Εθνικό Μετσόβιο Πολυτεχνείο

Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Τομέας Επικοινωνιών, Ηλεκτρονικής και Συστημάτων Πληροφορικής
Εργαστήριο Δικτύων Υπολογιστών

Υλοποίηση ιδιωτικής, ασφαλούς υπηρεσίας νέφους αποθήκευσης

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

ΑΝΔΡΕΑ ΤΣΑΓΚΑΡΟΠΟΥΛΟΥ

Επιβλέπων: Ευστάθιος Συκάς
Καθηγητής Ε.Μ.Π

Αθήνα, Σεπτέμβριος 2016

Εθνικό Μετσόβιο Πολυτεχνείο

Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Τομέας Επικοινωνιών, Ηλεκτρονικής και Συστημάτων Πληροφορικής
Εργαστήριο Δικτύων Υπολογιστών

Υλοποίηση ιδιωτικής, ασφαλούς υπηρεσίας νέφους αποθήκευσης

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

ΑΝΔΡΕΑ ΤΣΑΓΚΑΡΟΠΟΥΛΟΥ

Επιβλέπων: Ευστάθιος Συκάς
Καθηγητής Ε.Μ.Π

Εγκρίθηκε από τη τριμελή εξεταστική επιτροπή τη 1η Σεπτεμβρίου 2016.

(Υπογραφή)	*(Υπογραφή)*	*(Υπογραφή)*
…….	…….	……...…….
Ευστάθιος Συκάς	Μιχαήλ Θεολόγου	Γεώργιος Στασινόπουλος
Καθηγητής Ε.Μ.Π	Καθηγητής Ε.Μ.Π	Καθηγητής Ε.Μ.Π

Αθήνα, Σεπτέμβριος 2016

(Υπογραφή)

……

ΑΝΔΡΕΑΣ Α. ΤΣΑΓΚΑΡΟΠΟΥΛΟΣ

Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών

Περίληψη

Σκοπός της παρούσας διπλωματικής εργασίας, είναι η ανάπτυξη μιας σύγχρονης υπηρεσίας νέφους αποθήκευσης η οποία θα παρέχει μέγιστη ασφάλεια. Η χρήση του νέφους αποθήκευσης σήμερα είναι δεδομένη, και ο χρήστης μπορεί να έχει πρόσβαση στις πληροφορίες του από παντού, αλλά οι εγγυήσεις για την ασφάλεια πολλές φορές δεν είναι ικανοποιητικές. Οι περισσότερες από τις εταιρείες που προσφέρουν τις υπηρεσίες τους στον τομέα, δεν δημοσιοποιούν τον κώδικά που χρησιμοποιούν και έτσι δεν υπάρχει η δυνατότητα ελέγχου της διαχείρισης των δεδομένων του χρήστη. Αυτό δημιουργεί επιπλέον αμφιβολίες για την ασφάλεια των δεδομένων και την προστασία της ιδιωτικότητας του χρήστη– και εν τέλει της δυνατότητας για ελεύθερη χρήση της υπηρεσίας. Αντίθετα, για την ανάπτυξη της υπηρεσίας μας βασιζόμαστε κατά κόρον σε λογισμικό ανοικτού κώδικα, εφαρμόζουμε την υπάρχουσα τεχνολογία ασφαλούς http σύνδεσης, και απαγορεύουμε τη πρόσβαση στα δεδομένα του χρήστη, εκτός των περιπτώσεων που αποστέλλονται ή παραλαμβάνονται αρχεία, αποκλείοντας έτσι τις περιπτώσεις επιθέσεων ωμής βίας. Η υπηρεσία βασίζεται σε τρία συστήματα: Τον Owncloud-Server, ο οποίος παρέχει τις δυνατότητες πρόσβασης στα αποθηκευμένα δεδομένα, τον Mqtt Broker ο οποίος επιτρέπει τη χρήση ενός ελαφρού και αποδοτικού πρωτόκολλο επικοινωνίας, και τον Android Client που είναι υπεύθυνος για το συγχρονισμό και τη παρουσίαση των πληροφοριών του χρήστη. Και τα τρία συστήματα, λειτουργούν με τις τελευταίες εκδόσεις που είναι διαθέσιμες, σε πολλά λειτουργικά συστήματα. Στη περίπτωση που κάποιο σύστημα δεν μπορεί να συνεργαστεί, προβλέπονται διαδικασίες που εξασφαλίζουν την υπηρεσία σε συνθήκες λειτουργίας.

Λέξεις-κλειδιά:

Internet, Domain, Privacy, Cloud, Open Source, Ιδιωτικότητα, Οικιακή Υπηρεσία, Νέφος Αποθήκευσης, Android, Linux, Raspberry Pi, MQTT, Client-Server, Ασφάλεια, Προσωπικά δεδομένα

Abstract

The purpose of this thesis is to implement a contemporary, secure cloud storage service, providing maximum security. The use of cloud storage nowadays is ubiquitous, so users can access their data from everywhere; however, security is not warranted in a satisfactory manner. Most of the companies specializing in the sector do not open source their products, and so the manipulation of the user data cannot be checked. This raises questions concerning the security of the data and user privacy – and of the possibility to use the service freely after all. Instead, to develop our service we use chiefly open source components, we implement the HTTPS specification and prohibit access to user data, except when uploading or downloading, thereby excluding the case of a successful brute-force attack. The service is based in three components: the Owncloud-Server, which provides storage and processing, the Mqtt Broker which enables the use of a light and efficient communication protocol, and the Android Client which is responsible for the synchronization of the data, and its presentation to the user. All three components can run in their latest versions, in many operation systems. In case a component cannot cooperate, we take measures to reassure the security of the service under operation.

Keywords:

Internet, Domain, Privacy, Cloud Storage, Open Source, Home Service, Android, Linux, Raspberry Pi, MQTT, Client-Server, Security, Private Data

Ευχαριστίες

Θα ήθελα να ευχαριστήσω στο σημείο αυτό τους γονείς και τα αδέλφια μου, χωρίς την υποστήριξη των οποίων δεν θα ήταν δυνατό να φτάσω μέχρι το σημείο αυτό. Επίσης, θα ήθελα να ευχαριστήσω ιδιαίτερα το συμφοιτητή μου Νικόλαο Παπάδη για όλη την συνεργασία και συναναστροφή στη διάρκεια των σπουδών μου, καθώς και όλους τους άλλους συμφοιτητές μου που με βοήθησαν να φτάσω μέχρι εδώ. Επιπλέον, θα ήθελα να ευχαριστήσω τον κ. Γιώργο Λυμπερόπουλο για τη βοήθειά του κατά την εκπόνηση της παρούσας εργασίας. Τέλος, θα ήθελα να ευχαριστήσω τους καθηγητές μου Ευστάθιο Συκά, Μιχαήλ Θεολόγου και Γεώργιο Στασινόπουλο για όλη τη βοήθειά τους στην προσέγγισή μου της έννοιας των δικτύων και του εφαρμοσμένου Internet.

Πίνακας Περιεχομένων

1 Εισαγωγή

1.1 Σκοπός της διπλωματικής εργασίας

Ο σκοπός αυτής της διπλωματικής εργασίας, είναι να δημιουργήσουμε μια σύγχρονη, ιδιωτική και πολύ ασφαλή υπηρεσία νέφους αποθήκευσης, με το ελάχιστο δυνατό κόστος. Η υπηρεσία θα απευθύνεται σε οικιακούς χρήστες, και σε πολύ μικρές επιχειρήσεις.

Οι υπηρεσίες νέφους είναι διαδικτυακές υπηρεσίες, που παρέχουν στους χρήστες τους πρόσβαση στα αρχεία τους από οπουδήποτε, αρκεί να υπάρχει σύνδεση στο διαδίκτυο και να κατέχουν μια συσκευή με πρόσβαση σε αυτό. Πλέον είναι πολύ εύκολο να στείλει ένας χρήστης ένα υπερσύνδεσμο ενός διαδικτυακού αντιγράφου των αρχείων του, τα οποία ένας άλλος μπορεί να κατεβάσει στη δική του συσκευή. Αυτό έχει μεγάλο αντίκτυπο στους τομείς της δημιουργίας αντιγράφων ασφαλείας, της συνεργασίας με κοινόχρηστα έγγραφα, και επίσης έχει δώσει στις κινητές συσκευές πολύ μεγάλη αυτονομία στον αποθηκευτικό τους χώρο.

Πολλές από τις υπηρεσίες προωθούνται από εταιρείες που έχουν αποκτήσει μεγάλη υπόληψη στο χώρο της τεχνολογίας όπως τη Google, τη Microsoft, τη Dropbox και άλλες. Το γεγονός αυτό έχει συντελέσει κατά μεγάλο λόγο στη δημοτικότητά τους, αλλά υπάρχουν αρκετές αδυναμίες και περιορισμοί που συνοδεύουν τη χρήση τους.

Δυστυχώς, για διάφορους λόγους πολλές από τις υπηρεσίες νέφους, δεν είναι σε θέση να προστατέψουν τα προσωπικά στοιχεία των χρηστών τους, και έχουν καταγραφεί περιπτώσεις πολύ μεγάλων επιτυχημένων επιθέσεων. Επίσης, γενικά παρέχουν πολύ μικρό αποθηκευτικό χώρο και πολύ βασικές υπηρεσίες σε όσους δεν έχουν συνδρομή, αλλά και όσοι έχουν συνδρομή πληρώνουν μεγαλύτερο κόστος από την αξία του προϊόντος που τους διατίθεται. Αρκετές φορές, έχουν σταματήσει τη λειτουργία τους, και έχουν αναγκάσει τους χρήστες να αναζητήσουν αλλού χώρο για τα δεδομένα τους. Τέλος οι εφαρμογές που σχετίζονται με μια τέτοια υπηρεσία εν γένει δεν μπορούν να χρησιμοποιηθούν σε κάποια άλλη, και είναι αδύνατη η συνάθροιση αποθηκευτικού χώρου από διάφορες υπηρεσίες σε μία.

Τα παραπάνω μειονεκτήματα οπωσδήποτε προβληματίζουν κάθε χρήστη αυτών των υπηρεσιών, και θέτουν ακόμη και τη περίπτωση της μη χρήσης τους ως μία ρεαλιστική επιλογή. Στα επόμενα κεφάλαια, θα αναλύσουμε τον τρόπο με τον οποίο μπορούμε να αναπτύξουμε μια σύγχρονη, ασφαλή και ιδιωτική υπηρεσία νέφους, η οποία θα έχει τη δυνατότητα υποστήριξης πολύ μεγάλου αποθηκευτικού χώρου με οικονομικά συμφέροντα τρόπο, θα λειτουργεί επ' αόριστον, και θα μπορεί να περιορίζει αποτελεσματικά τη πρόσβαση στα δεδομένα του χρήστη.

1.2 Δομή της διπλωματικής εργασίας

Στο κεφάλαιο 2 γίνεται μια συγκριτική μελέτη των λύσεων που υφίστανται μέχρι στιγμής και των πλεονεκτημάτων της λύσης που προτείνουμε σε αυτή την εργασία.

Στο κεφάλαιο 3 αναφερόμαστε στην αρχιτεκτονική του συστήματος που θα υλοποιήσουμε, δηλαδή τα μέρη από τα οποία αυτό αποτελείται, και πώς αλληλεπιδρούν. Στο ίδιο κεφάλαιο, αναφερόμαστε σε όλα τα ειδικά στοιχεία που χρησιμοποιούνται για την ανάπτυξη της λύσης.

Στο κεφάλαιο 4 αναφέρουμε λεπτομερώς τις διαδικασίες που απαιτούνται προκειμένου να εγκαταστήσουμε τον βασικό εξυπηρετητή της υπηρεσίας μας, και να γίνει προσβάσιμος από το Internet.

Στο κεφάλαιο 5 θέτουμε τα θεμέλια της ασφαλούς εφαρμογής-πελάτη Android που θα δημιουργήσουμε. Αρχικά παρούσιάζουμε γενικές αρχές του προγραμματισμού και της λειτουργίας του Android, και στη συνέχεια εισάγουμε τα βασικά τμήματα που θα ενωθούν κατάλληλα για να απαρτίσουν την εφαρμογή.

Στο κεφάλαιο 6 παρουσιάζουμε βήμα-βήμα τις αλλαγές που πρέπει να πραγματοποιήσουμε στον κώδικα που ήδη έχουμε εισάγει στο κεφάλαιο 5, για να έχουμε αποκλειστική πρόσβαση στα δεδομένα μας, και να εμποδίσουμε την πρόσβαση τρίτων.

Στο κεφάλαιο 7 παραθέτουμε μερικά μικρά προγράμματα που θα εκτελούνται στον εξυπηρετητή, και μας εξασφαλίζουν ότι τα δεδομένα μας δεν θα διαρρεύσουν ακόμη και αν ο πελάτης δεν λειτουργήσει όπως αναμένουμε.

Στο κεφάλαιο 8 παρουσιάζουμε επιπλέον προτάσεις για την επέκταση της παρούσας εργασίας, και τα συμπεράσματα που προέκυψαν από την εκπόνησή της.

Το κεφάλαιο 9 συνοψίζει τις παραπομπές του κυρίου μέρους της εργασίας.

Τα κεφάλαια 10,11,12, αποτελούν παραρτήματα, και περιέχουν τον κώδικα που χρειάστηκε να αλλάξουμε για την ανάπτυξη της εφαρμογής Android, το πλήρες αρχείο προσαρμογής του openssl στον server, και πληροφορίες σχετικές με το πρωτόκολλο MQTT.

2 Υπηρεσίες Νέφους αποθήκευσης

2.1 Αναγκαιότητα και τρόπος λειτουργίας

Από την αρχική εμφάνιση των υπολογιστικών συστημάτων, υπήρχε το εμφανές πρόβλημα της συγκέντρωσης μεγάλης υπολογιστικής ισχύος σε περιορισμένο χώρο. Σήμερα, 50 περίπου χρόνια αργότερα, παρά τη τεράστια αύξηση των διαθέσιμων επιδόσεων και τη μεγάλη πτώση των τιμών το πρόβλημα εξακολουθεί να υφίσταται για το μέσο χρήστη- όχι τόσο στο τομέα των υπολογιστικών πόρων (επεξεργαστικής ισχύος, μνήμης RAM) όσο στον τομέα του αποθηκευτικού χώρου και του διαμοιρασμού των αρχείων.

Πλέον είναι πολύ εύκολο για τον μέσο χρήστη να δημιουργήσει μεγάλες ποσότητες ψηφιακού υλικού, και μέσα απο το Internet είναι δυνατή η απόκτηση πολύ μεγάλου όγκου έτοιμου ψηφιακού υλικού. Υπάρχουσες λύσεις, όπως οι εξωτερικοί σκληροί δίσκοι ή οι δίσκοι USB, δεν λύνουν το πρόβλημα της χωρητικότητας και του διαμοιρασμού ικανοποιητικά διότι η πρόσβαση στα δεδομένα εξαρτάται από τη κατοχή του φυσικού μέσου, επομένως ο διαμοιρασμός δεν μπορεί να γίνει σε πραγματικό χρόνο.

Για την κάλυψη αυτής της ανάγκης, τα τελευταία χρόνια έχουν αναπτυχθεί υπηρεσίες που απευθύνονται σε οικιακούς χρήστες, εκ μέρους μεγάλων εταιρειών όπως η Microsoft, η Google, η Box, η Dropbox κ.α .Οι υπηρεσίες αυτές αναλαμβάνουν να αποθηκεύσουν τα δεδομένα των χρηστών τους στα δικά τους κέντρα δεδομένων τα οποία πλαισιώνονται από εξειδικευμένο προσωπικο και εξοπλισμό, και είναι σε θέση να συντηρούν πολύ μεγάλες χωρητικότητες αποθήκευσης με μεγάλη αξιοπιστία. Προκειμένου να γίνει κάποιος χρήστης μιας τέτοιας υπηρεσίας πρέπει να αποκτήσει ένα μοναδικό συνδυασμό ονόματος χρήστη και κωδικού, στοιχεία που πρέπει να δίνει κάθε φορά που την επισκέπτεται.

Με τη βοήθεια αυτών των υπηρεσιών ο χρήστης μπορεί είτε να στείλει δεδομένα (upload) στο χώρο αποθήκευσης που του διατίθεται, είτε να λάβει ξανά κάποια από αυτά τα δεδομένα (download) σε μια τοπική συσκευή αποθήκευσης, όπως είναι για το κινητό τηλέφωνο η κάρτα μνήμης. Από τη στιγμή που τα δεδομένα μεταφερθούν στις αποθήκες δεδομένων της υπηρεσίας, λειτουργούν ως ένα δυνητικό αντίγραφο ασφαλείας για το χρήστη, το οποίο όμως – αντίθετα από ένα παραδοσιακό αντίγραφο σε κάποιο δίσκο – είναι διαθέσιμο από οπουδήποτε, αρκεί να υπάρχει σύνδεση στο Internet.

Ο πιο κοινός τρόπος για να αποκτήσει ξανά πρόσβαση στα δεδομένα του ο χρήστης, είναι να πλοηγηθεί στην ιστοσελίδα της υπηρεσίας, η οποία παρέχει κατάλληλο γραφικό μενού για να μπορεί να έχει εποπτεία των αρχείων όπως και σε ένα συνηθισμένο διαχειριστή αρχείων (file manager). Συνήθως αυτός ο τρόπος δίνει πρόσβαση στη πλήρη λειτουργικότητα της υπηρεσίας, διότι ο χρήστης μπορεί να καθορίζει άδειες για τα αρχεία του, ή να τα διαμοιραστεί και με άλλους χρήστες. Αρκετές φορές όμως, ο χρήστης θέλει να έχει μόνιμα κάποια αρχεία προσβάσιμα και από το τοπικό σύστημα αποθήκευσής του και από το διαδικτυακό, και μάλιστα όλες οι αλλαγές σε

κάποιο από τα δύο μέσα να αντικατοπτρίζονται και στο άλλο. Στις περιπτώσεις αυτές που ο χρήστης θέλει μια πιο μόνιμη και ανθεκτική σε λάθη επικοινωνία με τη διαδικτυακή υπηρεσία, συνήθως παρέχεται ένα ειδικό προγράμμα-πελάτης (client). Ο client είναι εγκατεστημένος σε κάποια ηλεκτρονική συσκευή με πρόσβαση στο Internet και μπορεί να υποστηρίζει το συγχρονισμό αρχείων υπό προϋποθέσεις (π.χ φίλτρα ονόματος, ημερομηνίας τροποποιησης), τη συνέχιση του ανεβάσματος των αρχείων από το τελευταίο σημείο ακόμη και όταν η διαδικασία διακοπεί από κάποιο πρόβλημα κ.α.

Από τα παραπάνω φαίνεται ότι ανοίγονται τα τελευταία χρόνια σημαντικές δυνατότητες αποθηκευτικής αυτονομίας ακόμη και για συσκευές περιορισμένης ισχύος όπως είναι τα κινητά τηλέφωνα. Όμως, παρά τα πολύ σημαντικά πλεονεκτήματα που παρουσιάσαμε, οι υπηρεσίες νέφους υπόκεινται στους κινδύνους παραβίασης της ιδιωτικότητας που αναφέραμε παραπάνω. Αρκετές φορές οι χρήστες επιλέγουν να ανεβάσουν υλικό που θεωρούν ότι είναι πολύ προσωπικό, όπως αντίγραφα δημοσίων εγγράφων που τους αφορούν, φορολογικά στοιχεία, ακόμη και αντίγραφα της ταυτότητάς τους. Υπό αυτό το πρίσμα, μια διαρροή μπορεί να αποκτήσει πολύ μεγάλες διαστάσεις, γιατί οι πληροφορίες αυτές σε λάθος χέρια μπορούν να αποτελέσουν υλικό ακόμη και για τον εκβιασμό του χρήστη.

Η υπηρεσία που υλοποιούμε στη συνέχεια, παρέχει όλα τα πλεονεκτήματα που έχει μια υπηρεσία νέφους, και επιπλέον ενισχυμένη ασφάλεια και ιδιωτικότητα ώστε ο χρήστης μόνο να έχει πρόσβαση στα δεδομένα όταν χρειάζεται, και κανείς άλλος.

2.2 Μειονεκτήματα των υπαρχουσών υπηρεσιών

- ***Προβλήματα ιδιωτικότητας:*** Επειδή ο κώδικας των εμπορικών υπηρεσιών είναι μυστικό της εταιρείας, δεν είναι δυνατόν να γνωρίζουμε ποιός είναι ο ακριβής τρόπος με τον οποίο χειρίζονται τα δεδομένα μας. Έχουν καταγραφεί περιπτώσεις [1], κατά τις οποίες μεγάλες εταιρείες υποχρεώθηκαν να πληρώσουν πρόστιμα για την μαζική παραβίαση της ιδιωτικότητας των πελατών τους. Φυσικά, εκτός από αυτές τις περιπτώσεις, οι περισσότερες από τις εταιρείες έχουν συμβάσεις παροχής υπηρεσιών, που δηλώνουν ρητά ότι τα στοιχεία που συλλέγονται από την υπηρεσία, θα χρησιμοποιηθούν κατά την κρίση της εταιρείας προκειμένου να προωθήσει τα συμφέροντά της. Χαρακτηριστικά αναφέρουμε για την περίπτωση της Google, ένα απόσπασμα από τη συμφωνία χρήσης (terms of service) [2] :

  Some of our Services allow you to upload, submit, store, send or receive content. [...] When you upload, submit, store, send or receive content to or through our Services, you give Google (and those we work with) a worldwide license to use, host, store, reproduce, modify, create derivative works (such as those resulting from translations, adaptations or other changes we make so that your content works better with our Services), communicate, publish, publicly perform, publicly display and distribute such content. The rights you grant in this license are for the limited purpose of operating, promoting, and improving our Services, and to develop new ones. This license continues even if you stop using our Services[...].

Για την περίπτωση της Dropbox ισχύουν τα εξής [3] :

Our Services also provide you with features like photo thumbnails, document previews, email organization, easy sorting, editing, sharing and searching. These and other features may require our systems to access, store and scan Your Stuff. You give us permission to do those things, and this permission extends to our affiliates and trusted third parties we work with.

Επιπλέον, υπάρχει και ο κίνδυνος της μεμονωμένης υπέρβασης καθηκόντων εκ μέρους του προσωπικού της εκάστοτε εταιρείας. Για παράδειγμα, το 2010 η Google απέλυσε έναν μηχανικό υπολογιστών που ήταν υπεύθυνος για τη συντήρηση ιστοσελίδων των χρηστών της, διότι κατηγορήθηκε ότι απέκτησε πρόσβαση στα δεδομένα ενός χρήστη της [4].

- ***Πολύ μεγάλα προβλήματα από εισβολείς:*** Τα τελευταία χρόνια έχουν παρατηρηθεί πολλές επιτυχημένες εισβολές σε δημοφιλείς διαδικτυακές υπηρεσίες. Μάλιστα, ο αριθμός των λογαριασμών που επηρεάστηκαν ήταν πολύ μεγάλος. Ενδεικτικά παραθέτουμε τον παρακάτω πίνακα:

Υπηρεσία	Χρονολογία	Λογαριασμοί που επηρεάστηκαν
Yahoo	1/2012	450,000
MySpace	5/2016	360,000,000
Evernote	3/2013	50,000,000
Adobe	10/2013	38,000,000
LinkedIn	2012	167,000,000
Tumblr	2013	65,000,000

Μάλιστα, σε αρκετές από τις παραπάνω περιπτώσεις τα στοιχεία που διέρρευσαν έγιναν αντικείμενο αγοραπωλησίας σε σχετικές ιστοσελίδες του Dark Web [5]. Γενικά, μια κεντρική υπηρεσία που εξυπηρετεί εκατομμύρια χρήστες, είναι πολύ πιο πιθανό να προσελκύσει επιθέσεις σε σχέση με μια ιδιωτική υπηρεσία.

- ***Περιορισμένος αποθηκευτικός χώρος*:** Οι περισσότερες υπηρεσίες νέφους δίνουν από 2.5 μέχρι 15GB αποθηκευτικού χώρου στους χρήστες της δωρεάν υπηρεσίας τους. Από την άλλη πλευρά η υπηρεσία μας έχει πρακτικά άπειρο αποθηκευτικό χώρο, αφού ο χρήστης μπορεί να συνδέσει όσο μεγάλο αποθηκευτικό μέσο επιθυμεί.

- ***Περιορισμένες δυνατότητες*:** Πολλές φορές τα καλύτερα σημεία της εφαρμογής είναι προσβάσιμα μόνο από όσους έχουν συνδρομή σε κάποιο πρόγραμμα επι πληρωμή.

- ***Υψηλό κόστος*:** Η χρήση μιας τυπικής υπηρεσίας νέφους, κοστίζει από 70-100€ ετησίως για το ένα TB. Με δεδομένο ότι ένας χρήστης μπορεί να έχει περιεχόμενο αρκετά μεγαλύτερο από αυτό το μέγεθος, το κόστος γίνεται σημαντικός παράγοντας.

- ***Έλλειψη συνεργασίας - Ιδιοτέλεια*:** Κάθε υπηρεσία διαθέτει διαφορετικές εφαρμογές, που δεν συνεργάζονται τις περισσότερες φορές με προγράμματα άλλων εταιρειών, ή συνεργάζονται πολύ περιορισμένα. Επίσης, όσες εφαρμογές αναπτύσσονται για μια εφαρμογή νέφους επί πληρωμή, δεν μπορούν να χρησιμοποιηθούν για άλλες υπηρεσίες, ούτε να βοηθήσουν στην ανάπτυξη ανεξάρτητων εφαρμογών.

Αντίθετα, με τη χρήση της υπηρεσίας που αναπτύσσουμε τα παραπάνω προβλήματα εξασθενούν σημαντικά ή παύουν να υφίστανται, ενώ τη θέση τους παίρνουν αντίστοιχα πλεονεκτήματα:

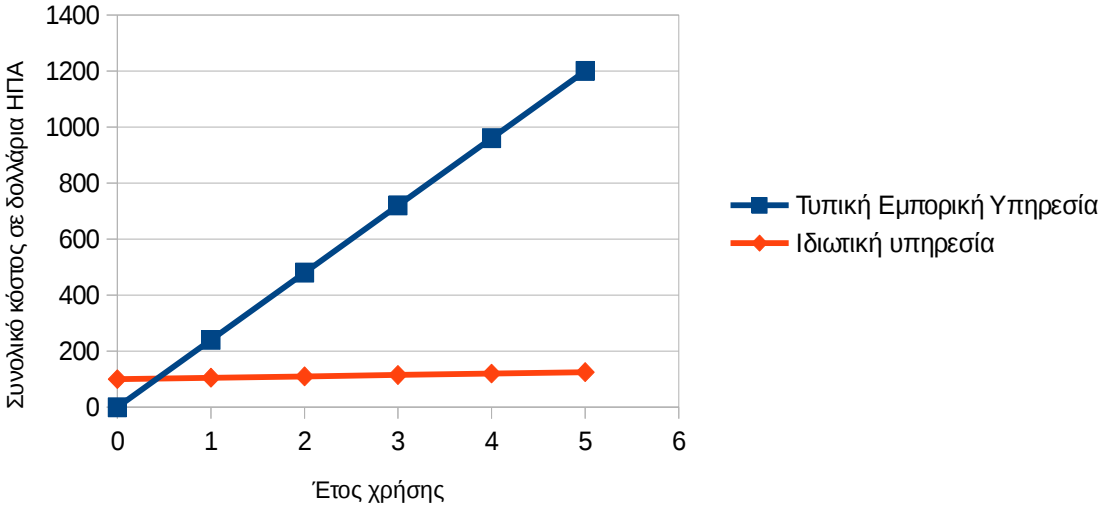
2.3 Πλεονεκτήματα από τη χρήση της ιδιωτικής υπηρεσίας νέφους αποθήκευσης

- ***Εξαιρετική ασφάλεια:*** Χάρη στην προσθήκη που πραγματοποιούμε στα πλαίσια αυτής της εργασίας, είναι εφικτό να αποκλείουμε την πρόσβαση τρίτων στα δεδομένα μας όταν δεν τα χρησιμοποιούμε και ταυτόχρονα να εκμηδενίζουμε τη πιθανότητα μιας επιτυχημένης επίθεσης ωμής βίας.

- ***Πρακτικά άπειρος αποθηκευτικός χώρος και συνεργασία:*** Κατά τη δημιουργία της υπηρεσίας μας, μπορουμε να ορίσουμε η αποθήκευση των δεδομένων να γίνεται όπου εμείς θέλουμε. Αυτό μας επιτρέπει να συνδέσουμε ένα οσοδήποτε μεγάλο αποθηκευτικό μέσο επιθυμούμε ως χώρο αποθήκευσης των δεδομένων, ενώ η υπηρεσία μας μπορεί να συνεργαστεί και με άλλες υπηρεσίες νέφους στις οποίες ο χρήστης έχει λογαριασμό, συναθροίζοντας τον αποθηκευτικό χώρο, και παρέχοντας μια ενιαία άποψη των αποθηκευμένων αρχείων.

- ***Μεγάλο οικονομικό όφελος:*** Αμέσως μετά την αρχική επένδυση, υπάρχει πολύ μεγάλο όφελος από τη χρήση μιας ιδιωτικής υπηρεσίας νέφους. Αυτό οφείλεται στο χαμηλό κόστος του ηλεκτρικού ρεύματος που μπορούμε να επιτύχουμε χρησιμοποιώντας μια κατάλληλη πλατφόρμα στο ρόλο του εξυπηρετητή.

Σύγκριση κόστους μεταξύ Ιδιωτικής Υπηρεσίας και Τυπικής Εμπορικής Υπηρεσίας



- ***Πλήρης έλεγχος του κώδικα και του τρόπου χρήσης των δεδομένων:*** Ακριβώς επειδή η υπηρεσία μας βασίζεται σε τεχνολογίες ανοικτού κώδικα, μπορούμε να επιθεωρήσουμε τον κώδικα, και να τον τροποποιήσουμε όπως θέλουμε. Είμαστε σε θέση να γνωρίζουμε ακριβώς τον τρόπο με τον οποίο συμπεριφέρεται η εφαρμογή μας κάθε χρονική στιγμή, και δεν υπάρχει η δυνατότητα μη εξουσιοδοτημένης πρόσβασης στα δεδομένα μας από τρίτους.

- ***Δυνατότητα εγκατάστασης οπουδήποτε και οποτεδήποτε, για οποιονδήποτε αριθμό χρηστών, για οποιοδήποτε σκοπό, για πάντα:*** Οι όροι με τους οποίους προσφέρεται το λογισμικό που αποτελεί τον πυρήνα της υπηρεσίας μας, διασφαλίζουν την ελευθερία του τελικού χρήστη να πραγματοποιήσει όσες εγκαταστάσεις θέλει για οποιοδήποτε σκοπό, χωρίς κανένα περιορισμό. Η κάθε εγκατάσταση της υπηρεσίας μπορεί να εξυπηρετεί τα μέλη μιας ολόκληρης οικογένειας καθώς υποστηρίζει τον ορισμό αυτόνομων χρηστών με προσωπικό συνθηματικό.

- ***Δωρεάν ενημερώσεις:*** Όλα τα παραπάνω θα ήταν ικανοποιητικά για λίγο καιρό αν δεν υπήρχε κάποια προοπτική για το μέλλον. Εκτός όμως από τη δωρεάν απόκτηση του λογισμικού, και οι ενημερώσεις του είναι δωρεάν για όλους τους χρήστες χωρίς κάποιο κόστος. Έτσι οι χρήστες όχι μόνο έχουν πρόσβαση στις τελευταίες δυνατότητες της τεχνολογίας, αλλά είναι ασφαλισμένοι και απέναντι στις τελευταίες επιθέσεις.

2.4 Επιπλέον δυνατότητες της ιδιωτικής υπηρεσίας νέφους αποθήκευσης

Είδαμε παραπάνω τα σημαντικότερα πλεονεκτήματα που σχετίζονται με τη χρήση μιας ιδιωτικής υπηρεσίας νέφους. Εκτός όμως από αυτά, που θεωρούνται απαραίτητα για να είναι ανταγωνιστική οποιαδήποτε τέτοια υπηρεσία, μας παρέχονται και επιπλέον δυνατότητες που καθιστούν την υπηρεσία μας εξαιρετικά ανταγωνιστική. Τέτοιες είναι οι παρακάτω:

- ***Ύπαρξη προγράμματος-πελάτη για πολλά λειτουργικά συστήματα:*** Πολλές από τις εμπορικές υπηρεσίες που υπάρχουν στην αγορά εστιάζουν τη προσοχή τους σε χρήστες Windows ή και Mac OS. Η υπηρεσία μας, μπορεί και λειτουργεί σε πολλά λειτουργικά συστήματα Linux απευθείας, ενώ επειδή είναι ανοικτού κώδικα μπορεί να μεταφερθεί και σε άλλα.

- ***Δημιουργία χρηστών με περιορισμένα ή πλήρη δικαιώματα:*** Ένας χρήστης με διαχειριστική πρόσβαση στην υπηρεσία, μπορεί ο ίδιος να ορίσει και άλλους χρήστες, και να τους απονείμει προνόμια.

- ***Προβολή όλων των δραστηριοτήτων που σχετίζονται με ένα αρχείο και σχολιασμός:*** Οι χρήστες μπορούν να γνωρίζουν πώς ακριβώς χρησιμοποιήθηκαν τα αρχεία τους, να τα σχολιάσουν, ή και να προσθέσουν ετικέτες (tags) για εύκολη αναζήτηση.

- ***Σύστημα ημερολογίου και επαφών:*** Ο κάθε χρήστης μπορεί να έχει το προσωπικό του ημερολόγιο και τις επαφές του, προσβάσιμα από οπουδήποτε.

- ***Αυτόματη αποστολή φωτογραφιών και βίντεο στον εξυπηρετητή:*** Μόλις μια κινητή συσκευή με ενεργή την εφαρμογή της υπηρεσίας τραβήξει μία φωτογραφία ή ένα βίντεο, αμέσως τα αποστέλλει μέσω Internet στον εξυπηρετητή για αποθήκευση. Μόλις τελειώσει η διαδικασία, είναι σε θέση να διαγράψει τα πρωτότυπα αρχεία που βρίσκονται στον αποθηκευτικό χώρο της συσκευής, ώστε να εξοικονομήσει χώρο.

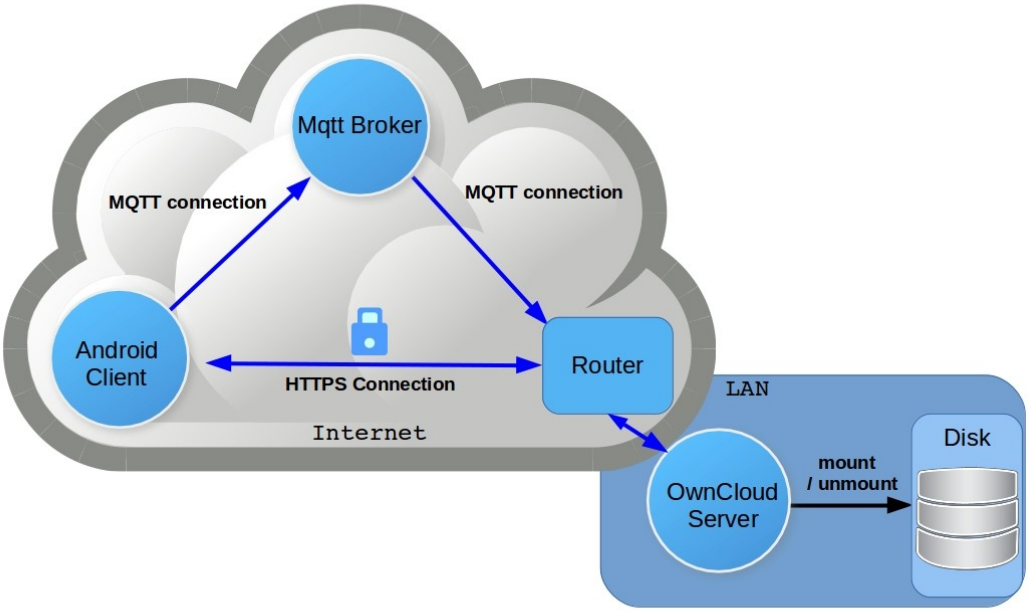
- ***Αναίρεση αλλαγών:*** Η υπηρεσία έχει ενσωματωμένο σύστημα καταγραφής των αλλαγών των αρχείων, οπότε στη περίπτωση λάθους, οι χρήστες μπορούν να επιστρέψουν σε ένα παλαιότερο αντίγραφο που αυτή έχει αποθηκεύσει.

- ***Πληθώρα εφαρμογών:*** Πολλές εφαρμογές, από την αναπαραγωγή ήχου μέχρι την επεξεργασία κειμένου, είναι διαθέσιμες δωρεάν για το χρήστη της εφαρμογής. Όλες οι εφαρμογές (250+) που υπάρχουν μέχρι τώρα είναι διαθέσιμες ως ελεύθερο λογισμικό.

3 Σχεδιασμός ασφαλούς υπηρεσίας αποθήκευσης στο νέφος

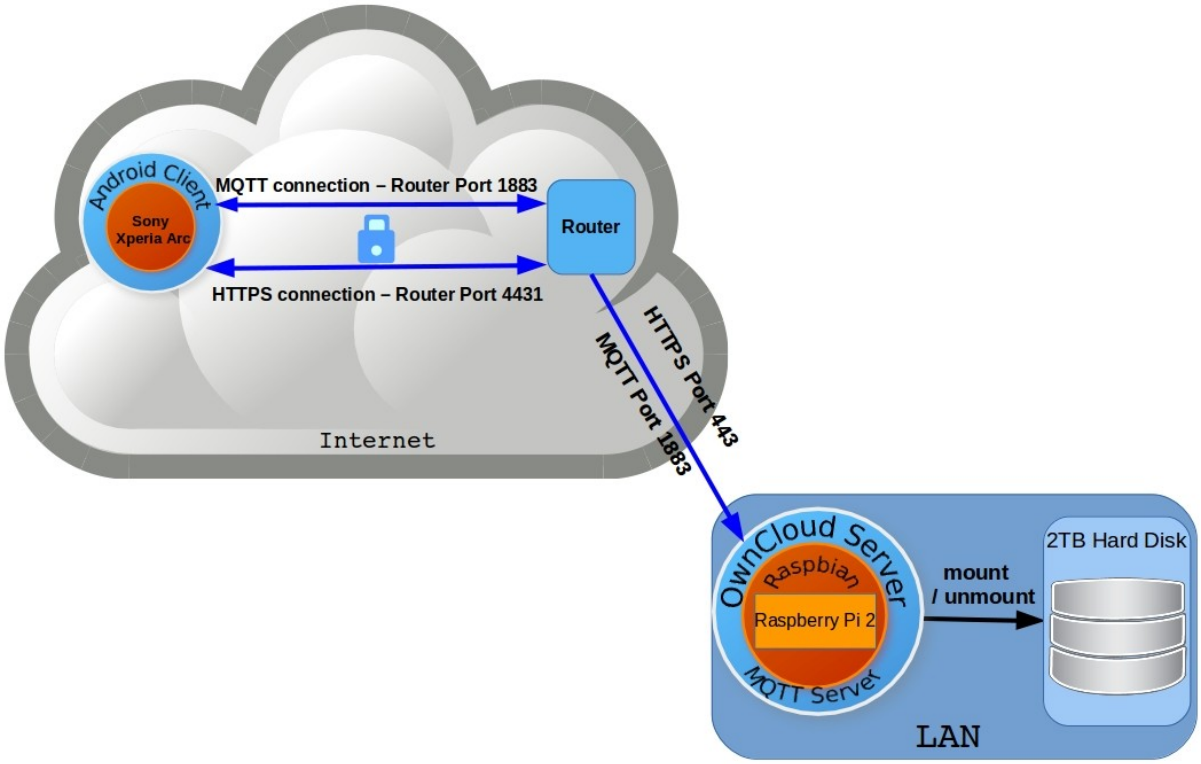
3.1 Γενικό αρχιτεκτονικό μοντέλο

Η λύση που αναπτύσσουμε, βασίζεται γενικά στην αρχιτεκτονική τύπου client – server. Αυτό είναι απαραίτητο από τη στιγμή που επιθυμούμε πρόσβαση στα δεδομένα μας με την ελάχιστη δυνατή πολυπλοκότητα και το μέγιστο δυνατό έλεγχο. Κάποια άλλη αρχιτεκτονική, όπως η peer – to – peer θα ήταν εφικτή, αλλά στην περίπτωση που το δίκτυο καταρρεύσει, ο χρήστης μπορεί να χάσει την πρόσβαση στα αρχεία του. Εάν μάλιστα το δίκτυο εκμεταλλεύεται τον ελεύθερο χώρο και στους δίσκους των άλλων ομότιμων για την παροχή αποθηκευτικού χώρου, υπάρχει η πιθανότητα της διαρροής δεδομένων – ακόμη και στη περίπτωση που είναι κρυπτογραφημένα – εξ' αιτίας κάποιας ατέλειας στον κώδικα του προγράμματος.
Για την υλοποίηση του πυρήνα της διάταξης, θα χρειαστεί να έχουμε κάποιο υπολογιστικό σύστημα με πρόσβαση στο Internet μέσω κάποιου δρομολογητή, το οποίο θα ονομάζουμε εξυπηρετητή (server). Στον server θα πρέπει να έχουμε εγκατεστημένο κάποιο (ανοικτού κώδικα) λειτουργικό σύστημα τύπου Linux, πάνω στο οποίο θα εγκαταστήσουμε το (ανοικτού κώδικα) λογισμικό Owncloud-Server που παρέχει λειτουργικότητα στην υπηρεσία μας. Έτσι, προκύπτει μία διάταξη που είναι διαφανής από άκρη σε άκρη, με μηδενικό κόστος επένδυσης σε άδειες λογισμικού. Ένα πολύ σημαντικό επιπρόσθετο πλεονέκτημα είναι ότι όλα τα προϊόντα επωφελούνται των τελευταίων αναβαθμίσεων από πλευράς ενημερώσεων ασφαλείας, κάτι που διασφαλίζει τη ποιότητα της υπηρεσίας μακροπρόθεσμα, οι οποίες διορθώνουν τα τυχόν λάθη στις τρέχουσες υλοποιήσεις.
Στην άλλη πλευρά της διάταξης, είναι ο χρήστης του πυρήνα – το πρόγραμμα πελάτης (client) – που είναι μία συσκευή android. Στη συσκευή θα εγκαταστήσουμε την ανοικτού κώδικα εφαρμογή Owncloud, στην οποία προσθέτουμε δυνατότητα επικοινωνίας με τον πυρήνα με χρήση του πρωτοκόλλου MQTT. Η χρήση του πρωτοκόλλου δεν προσθέτει σημαντικό βάρος στην εφαρμογή, και εγγυάται την ποιότητα της επικοινωνίας ακόμη και σε δύσκολες συνθήκες. Μια αρμόδια υπηρεσία θα χρησιμοποιεί τη σύνδεση που θα επιτυγχάνεται ώστε να επισημαίνει την ανάγκη χρήσης του δίσκου με κατάλληλα μηνύματα. Όταν ο χρήστης εκκινεί την εφαρμογή android θα αποστέλλεται ένα μήνυμα στον MQTT Broker, το οποίο θα πληροφορεί κάποια υπηρεσία του συστήματος να συνδέσει τον δίσκο, και όταν η εφαρμογή αδρανεί, θα αποστέλλεται μήνυμα το οποίο οδηγεί στην αποσύνδεση του δίσκου. Στη γενική περίπτωση ο MQTT Broker θα βρίσκεται σε διαφορετικό μηχάνημα από τον Owncloud Server.
Παρακάτω βλέπουμε μια σχηματική απεικόνιση των παραπάνω. Η φορά των βελών δείχνει τη κατεύθυνση των μηνυμάτων κατά τη δημιουργία και λειτουργία μιας καινούργιας σύνδεσης.



Όπως παρατηρούμε, η παραπάνω τοπολογία δεν εξαρτάται καθόλου από το χρησιμοποιούμενο υλικό, ούτε έχει μεγάλες απαιτήσεις για τη λειτουργία της. Έτσι, έχουμε ευελιξία στον τρόπο με τον οποίο μπορούμε να την υλοποιήσουμε. Για παράδειγμα, σε ένα δίκτυο αισθητήρων, ο owncloud server μπορεί να στηθεί ακόμη και επάνω σε μια ενσωματωμένη συσκευή, και ο σκληρός δίσκος μπορεί να είναι ακόμη και μια μνήμη flash. Σε ένα πιο απαιτητικό σχήμα, ο Router θα μπορούσε να αντιπροσώπευε μια υψηλών δυνατοτήτων προγραμματιζόμενη συσκευή διασύνδεσης με το Internet, ώστε να επιτυγχάνεται μεγαλύτερος έλεγχος στην επικοινωνία με το εσωτερικό δίκτυο, ενώ ο δίσκος θα μπορούσε να ήταν SSD, ώστε να παρέχει υψηλές ταχύτητες πρόσβασης στα δεδομένα.

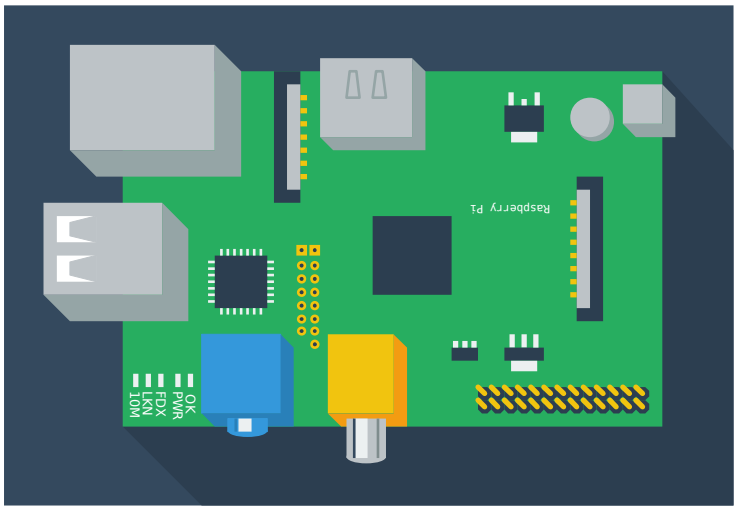
Στη συγκεκριμένη διπλωματική εργασία, η τοπολογία μας έχει τη παρακάτω μορφή:



3.2 Το υλικό και το λογισμικό που χρησιμοποιήθηκε

Η εργασία μας δεν είναι δυνατό να πραγματοποιηθεί χωρίς την δυνατότητα εξασφάλισης κατάλληλου υλικού και λογισμικού. Στις σελίδες που ακολουθούν, παραθέτουμε τα απαραίτητα κομμάτια που χρησιμοποιούμε στη πορεία ανάπτυξης της εργασίας, ξεκινώντας από τα στοιχεία υλικού:

Raspberry Pi 2



Στην υλοποίησή μας, η συσκευή Pi 2 λειτουργεί ως εξυπηρετητής. Αποτελείται από ένα τετραπύρηνο επεξεργαστή ARM Cortex-A7 στα 900MHz, και έχει 1GB RAM. Μπορεί να τρέξει αρκετά ελαφρά λειτουργικά συστήματα που έχουν αναπτυχθεί ή προσαρμοστεί για τη συσκευή, μεταξύ των οποίων εκδοχές των Debian, Arch Linux, Windows 10 αλλά και συστήματα που λειτουργούν αποκλειστικά ως media server, όπως το openelec.
Στην υλοποίησή μας έχει εγκατεστημένο το λειτουργικό σύστημα Raspbian, που προέρχεται από το Debian, και ενδείκνυται ως επιλογή από το Raspberry Pi Foundation. Το λειτουργικό σύστημα της συσκευής γράφεται σε μια microSD card, και το μηχάνημα εκκινεί από αυτήν. Υπάρχει απαίτηση για την ταχύτητα της sdcard, διότι από αυτή θα φορτώνεται το λειτουργικό σύστημα στην εκκίνηση. Στην υλοποίησή μας χρησιμοποιήσαμε μια κάρτα χωρητικότητας 8GB, κλάσης 10.
Η ενεργειακή κατανάλωση της συσκευής είναι 4W, και το χαμηλό κόστος που αυτή συνεπάγεται (~5€ τη στιγμή παρουσίασης της διπλωματικής) την καθιστά ιδανική για λειτουργία ως server. Η τιμή της είναι πολύ προσιτή, και στην ελληνική αγορά είναι περίπου 40€.
Η απόδοση της συσκευής σχετικά με τις προδιαγραφές της είναι ικανοποιητική για ένα οικιακό χρήστη.

Seagate Momentus 7200.4

Ως εξωτερικό μέσο αποθήκευσης, χρησιμοποιήθηκε ο HDD Seagate Momentus 7200.4, αλλά θα μπορούσε να χρησιμοποιηθεί οποιοσδήποτε σκληρός δίσκος με δυνατότητα διασύνδεσης μέσω USB. Η ταχύτητες που επιτυγχάνει ένας μέσου κόστους σκληρός δίσκος είναι προς το παρόν πολλαπλάσιες της μέγιστης ταχύτητας που μπορεί να επιτύχει οποιαδήποτε σύνδεση Internet σε επίπεδο τελικού καταναλωτή, επομένως ακόμη και ένας πολύ αργός δίσκος μπορεί να αποτελέσει ένα καλό μέσο αποθήκευσης. Ο δίσκος θέλουμε να έχει εγκατεστημένο το σύστημα αρχείων ntfs, ώστε να έχουμε μέγιστη συμβατότητα με όλα τα λειτουργικά συστήματα, και ειδικά στην περίπτωση που θα χρειαστεί να αποκτήσουμε άμεση πρόσβαση στα αρχεία του από συσκευή Windows. Για την εξασφάλιση αρκετού ρεύματος στον δίσκο, είναι πιθανόν να απαιτείται η εισαγωγή της εντολής max_usb_current=1 'στο αρχείο /boot/config.txt του λειτουργικού συστήματος Raspbian, ώστε να διπλασιαστεί η μέγιστη παροχή ρεύματος από τα 0.6 A στα 1.2 A.

Sony Xperia Arc

Για την δοκιμή και ανάπτυξη του Android Client, χρησιμοποιήθηκε η κινητή συσκευή Sony Xperia Arc, με λειτουργικό σύστημα CyanogenMod 9, βασισμένο στο Android 4.0. Η συσκευή αν και κρίνεται χαμηλών προδιαγραφών με τα σημερινά δεδομένα (1.0GHz single-core CPU, 512MB RAM), μπόρεσε να αντεπεξέλθει εύκολα στις απαιτήσεις της εφαρμογής. Οι περισσότερες συσκευές που έχουν κυκλοφορήσει από το 2013 και έπειτα, μπορούν να ανταπεξέλθουν στις απαιτήσεις της.

Από τη πλευρά του λογισμικού, χρησιμοποιήθηκαν τα παρακάτω προγράμματα:

Android – Emulator

Η εφαρμογή που αναπτύχθηκε είναι συμβατή με τη πιο νέα έκδοση android (android 6.0 MarshMallow) , όπως μπορεί να ελεγχθεί με τη χρήση emulator. Το περιβάλλον προγραμματισμού που επιλέχθηκε είναι το Android Studio 2.1.0. Η εφαρμογή που προέκυψε τελικά είναι το αποτέλεσμα της συνένωσης της πιο καινούργιας έκδοσης της εφαρμογής owncloud, με την εφαρμογή android-mqtt, με κομμάτια κώδικα που έγραψε ο συγγραφέας. Η χρήση του emulator είναι απαραίτητη για τον έλεγχο του κώδικα σε διάφορες συσκευές και λειτουργικά συστήματα.

Apache Server

Για τη φιλοξενία του owncloud-server, είναι απαραίτητη η ύπαρξη ενός εξυπηρετητή HTTP, καθώς η μεταφορά των αρχείων μεταξύ της εφαρμογής και του εξυπηρετητή γίνεται με την ασφαλή εκδοχή αυτού του πρωτοκόλλου. Η εφαρμογή owncloud από την πλευρά του εξυπηρετητή βασίζεται στον Apache – Server, και η έκδοση που χρησιμοποιήθηκε είναι η 2.4.10.

Owncloud Server

Πρόκειται για την εφαρμογή φιλοξενίας των αρχείων μας.Συγκεκριμένα, χρησιμοποιήσαμε την ανοικτού κώδικα εφαρμογή της Owncloud, η οποία διανέμεται υπό τους όρους της AGPL, καθώς υπάρχει και τμήμα της εφαρμογής που είναι κλειστού κώδικα, και διανέμεται επί πληρωμή. Η εφαρμογή υποστηρίζει τις πλέον χρήσιμες δυνατότητες που υπάρχουν και στις δημοφιλείς ανταγωνιστικές υπηρεσίες, ενώ έχει πελάτες υπολογίσιμους φορείς όπως το CERN.
Κατά τη διάρκεια της υλοποίησης της εργασίας, χρησιμοποιήσαμε τις εκδόσεις 7.0.6,8.2.3, και 9.0.2 με διαδοχικές αναβαθμίσεις, χωρίς η έκδοση του server δεν επηρεάσει την ανάπτυξη της εφαρμογής. Ο Owncloud Server, μπορεί να χρησιμοποιήσει αρκετά περιβάλλοντα βάσεων δεδομένων, αλλά και σε διαφορετικούς από τον Apache εξυπηρετητές. Για τη συγκεκριμένη υλοποίηση όμως, χρησιμοποιήσαμε την MySQL, έκδοση 14.14.

MQTT Server

Χρησιμοποιήθηκε η έκδοση 1.4.9 του Mosquitto open-source MQTT Server. Με βάση το σχεδιασμό της υπηρεσίας, η εφαρμογή μπορεί να εκτελείται σε κάποιον dedicated server, ακόμη και εκτός οικιακού δικτύου. Στην υλοποίησή μας, η εγκατάσταση έγινε στο ίδιο μηχάνημα που φιλοξενεί τον Owncloud Server, και η απόδοσή ήταν εξαιρετική.

4 Ρύθμιση της υπηρεσίας Owncloud-Server

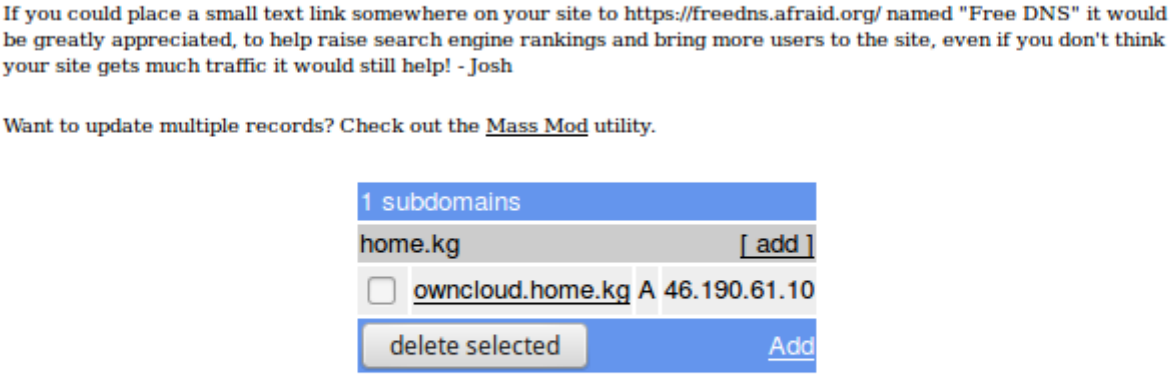
Κατά την παρουσίαση αυτής της διπλωματικής εργασίας, θα προχωρήσουμε βηματικά ως προς κάθε κομμάτι που αναλύσαμε παραπάνω, εγκαθιστώντας αρχικά τις βασικές εκδόσεις όλων των προγραμμάτων και υπηρεσιών, και στη συνέχεια προχωρώντας προς τις πιο περίπλοκες.

4.1 Λειτουργικό σύστημα και συνδεσιμότητα

Για τη διπλωματική αυτή εργασία, θεωρούμε ότι ο χρήστης έχει κατεβάσει το λειτουργικό σύστημα Raspbian, και το έχει εγκαταστήσει στη συσκευή του. Επίσης, θα θεωρήσουμε ότι έχουμε τη δυνατότητα επικοινωνίας με τη συσκευή, είτε μέσω ενός σταθερού συστήματος (οθόνη, ποντίκι, πληκτρολόγιο) είτε μέσω σύνδεσης SSH. Δεν κάνουμε καμία άλλη παραδοχή για το υλικό και το λογισμικό που είναι εγκατεστημένο στη συσκευή.

4.2 Ρύθμιση Domain Name και σύνδεση στον οικιακό εξυπηρετητή από το Internet

Για να μπορούμε να συνδεθούμε με την υπηρεσία μας από το Internet, θα πρέπει να γνωρίζουμε την IP διεύθυνσή της, ή να της έχουμε αποδώσει ένα domain name συσχετίζοντας το με την IP διεύθυνσή της, με τη βοήθεια ενός DNS εξυπηρετητή. Επειδή για τους περισσότερους οικιακούς καταναλωτές οι διευθύνσεις IP είναι δυναμικές, η σύνδεση στην υπηρεσία με τον πρώτο τρόπο είναι αδύνατη – επομένως θα πρέπει να της αποδώσουμε ένα domain name.
Αρχικά επισκεπτόμαστε έναν ιστότοπο παροχής δωρεάν ονόματος DNS, ο οποίος θα κάνει την αντιστοίχηση μεταξύ του domain name της υπηρεσίας μας, και της οικιακής IP την οποία στη συνέχεια θα συσχετίσουμε με τη συσκευή μας. Υπάρχουν αρκετοί τέτοιοι ιστότοποι, ένας από τους οποίους είναι ο https://freedns.afraid.org. Αφού δημιουργήσουμε ένα χρήστη και μπούμε στο σύστημα, μας παρουσιάζεται το παρακάτω μενού στο κέντρο της οθόνης:



Από το πεδίο Add μπορούμε να προσθέσουμε κάποιο άλλο domain name, το οποίο επιλέγεται από μία λίστα δωρεάν ονομάτων. Αφού πλέον επιλέξουμε κάποιο όνομα, πρέπει να ρυθμίσουμε την ανανέωση της αντιστοίχησης μεταξύ της διεύθυνσης IP και του domain name. Για το λόγο αυτό, επιλέγουμε το πεδίο Dynamic IP addressing, στο αριστερό τμήμα της ιστοσελίδας, από το οποίο οδηγούμαστε σε μια νέα ιστοσελίδα που μας παρουσιάζει κάποια scripts που μπορούμε να χρησιμοποιούμε για να ανανεώνουμε την αντιστοίχηση. Με τη χρήση ενός script σε τακτικά χρονικά διαστήματα, μπορούμε να εξασφαλίσουμε ότι η αντιστοίχηση θα είναι πάντα επίκαιρη.
Το λειτουργικό σύστημα Raspbian όπως και τα περισσότερα λειτουργικά συστήματα Linux, διαθέτει μια υπηρεσία εκτέλεσης εντολών σε συγκεκριμένα χρονικά διαστήματα που ονομάζεται cron. Ως χρήστες του συστήματος, μπορούμε να μεταβάλλουμε τα περιεχόμενα του αρχείου της υπηρεσίας και να επιβάλλουμε την εκτέλεση των εντολών που επιθυμούμε. Για να πετύχουμε λοιπόν την εκτέλεση ενός από τα scripts που προτείνονται (έστω του curl script), ανοίγουμε ένα τερματικό στον εξυπηρετητή και πληκτρολογούμε,

sudo crontab -e

Αφού επιλέξουμε τον επεξεργαστή κώδικα που θα χρησιμοποιούμε, αν δεν το έχουμε κάνει ήδη, ανοίγουμε το αρχείο και εκεί προσθέτουμε την εξής γραμμή:

```
*/5 *   *   *   *    curl http://freedns.afraid.org/dynamic/update.php?
TTcwZ3U0Y3hZNEFzTUVEelpIYVVRa2xCOjE1MzQyNDI2 >>
/home/pi/Desktop/IPlog/freedns_owncloud_home_kg.log 2>&1 &
```

Η εντολή αυτή θα εκτελείται κάθε 5 λεπτά, και θα αποθηκεύει την έξοδό της (stdout) , καθώς και το stderr της σε ειδικό αρχείο καταγραφής, για σκοπούς ανίχνευσης λαθών (debugging). Εκτός από τις στατιστικές πληροφορίες, η έξοδος περιέχει μηνύματα όπως το παρακάτω:

```
ERROR: Address 46.190.61.10 has not changed.
```

ή, στην περίπτωση που έχει υπάρξει αλλαγή διεύθυνσης,

```
Updated 1 host(s) owncloud.home.kg to 46.190.61.10 in 0.049 seconds.
```

Η εντολή συσχετίζει τη διεύθυνση http που επισκεπτόμαστε με την IP του δρομολογητή μας, και αλλάζει ανάλογα την εγγραφή DNS τύπου A που διατηρεί για τη διεύθυνση owncloud.home.kg .

4.3 Εγκατάσταση και ρύθμιση της πλατφόρμας Owncloud-Server

Στη συνέχεια, θα πρέπει να εγκαταστήσουμε τον Owncloud-Server στη συσκευή μας, ώστε να ενεργοποιήσουμε την αυθεντική έκδοση της υπηρεσίας, και να δοκιμάσουμε τη λειτουργικότητά της. Εκτελούμε λοιπόν την εντολή:

sudo apt-get install owncloud

Στο σημείο αυτό παρατηρούμε ότι η εγκατάσταση του πακέτου owncloud, εγκαθιστά μαζί με αυτό και διάφορα άλλα προγράμματα που απαιτούνται, όπως τον apache server, και την php. Όμως, δεν εγκαθίσταται η mysql, η οποία είναι αναγκαία για τη διαχείριση των αρχείων που θα αποθηκευτούν στην υπηρεσία μας. Για το σκοπό αυτό θα πρέπει να εκτελέσουμε την εντολή:

sudo apt-get install mysql-server

Στη συνέχεια, θα πρέπει να διασφαλίσουμε ότι ο Server θα αποκρίνεται στις αιτήσεις μας. Για το λόγο αυτό, θα πρέπει να προσθέσουμε το όνομα του Server, τόσο στο τοπικό δίκτυο, αν επιθυμούμε να έχουμε πρόσβαση και από αυτό, όσο και στο δημόσιο δίκτυο, στο /etc/owncloud/config.php που διατηρεί η εφαρμογή Owncloud-Server.

Εκτελούμε την εντολή:

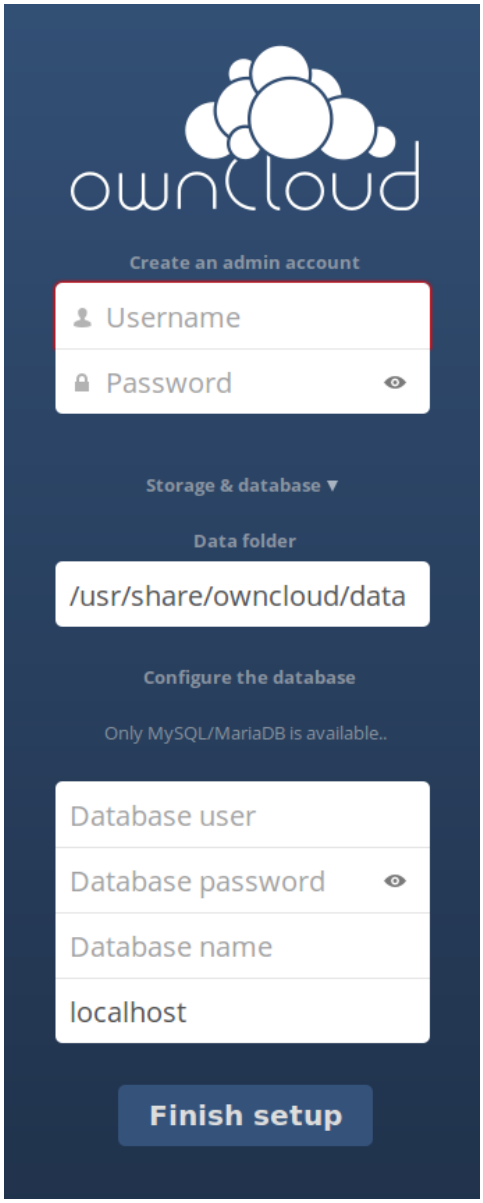
sudo nano /etc/owncloud/config.php

Στη συνέχεια εντοπίζουμε το πεδίο trusted domains, και φροντίζουμε να έχει περιεχόμενα όμοια με αυτά που παρατίθενται παρακάτω:

```
'trusted_domains' =>
  array (
    0 => 'localhost',
    1 => '192.168.1.6',
    2 => 'owncloud.home.kg',
  )
```

Το αναγνωριστικό 0 χρησιμοποείται για την πρόσβαση από τον ίδιο τον εξυπηρετητή της υπηρεσίας, το αναγνωριστικό 1 για πρόσβαση από τους υπολογιστές του εσωτερικού δικτύου, ενώ το αναγνωριστικό 2 για τους υπολογιστές στο Internet. Η αρίθμηση δεν είναι απόλυτη, και μπορούμε να προσθέσουμε και άλλα αναγνωριστικά αν θέλουμε (π.χ όνομα της συσκευής στο τοπικό δίκτυο που μπορεί να υποκαταστήσει την IP διεύθυνση της, ή τη δημόσια στατική IP του server, αν υπάρχει). Επίσης πρέπει να αλλάξουμε τα αναγνωριστικά που αναφέρονται παραπάνω ώστε να ταιριάζουν αφενός με την διεύθυνση της συσκευής στο τοπικό δίκτυο (όπως στο αναγνωριστικό 1), και αφετέρου με το domain name της υπηρεσίας μας (όπως στο αναγνωριστικό 2).
Αμέσως μετά τις παραπάνω τροποποιήσεις είμαστε σε θέση να ελέγξουμε την εγκυρότητα του

σχήματός μας, πληκτρολογώντας σε ένα browser localhost – αν βρισκόμαστε στον ίδιο τον server – ή τη τοπική IP διεύθυνση της συσκευής από ένα υπολογιστή που βρίσκεται στο ίδιο LAN. Τότε, μας παρουσιάζεται η πιο κάτω οθόνη:



Αρχικά, πληκτρολογούμε ένα όνομα χρήστη για την υπηρεσία νέφους, και ένα κωδικό. Τα στοιχεία αυτά θα είναι απαραίτητα για κάθε επόμενη φορά που θα θέλουμε να αποκτήσουμε πρόσβαση στην υπηρεσία, είτε διαδικτυακά, είτε μέσω ενός client.
Έπειτα, θα πρέπει να αλλάξουμε την τοποθεσία αποθήκευσης των δεδομένων, ώστε αυτά να αποθηκεύονται στο σκληρό δίσκο και όχι στον περιορισμένο χώρο της sdcard. Θα δημιουργήσουμε τον φάκελο /var/www/owncloud/data, στον οποίο θα συνδέεται (mount) ο σκληρός δίσκος – ή και ένα μόνο διαμέρισμά του (partition) που θέλουμε να περιέχει τα δεδομένα. Αν σκοπεύουμε να χρησιμοποιήσουμε κάποιο διαμέρισμα θεωρούμε ότι σε αυτό το σημείο ο δίσκος έχει διαμεριστεί, και ότι γνωρίζουμε το διαμέρισμα που θα χρησιμοποιηθεί για τα δεδομένα της υπηρεσίας μας – έστω ότι αυτό είναι το /dev/sda2. Τότε, θα προσθέσουμε την εξής γραμμή στο αρχείο /etc/fstab

(όπως αναφέραμε και παραπάνω, υποθέτουμε δίσκο μορφοποιημένο κατά NTFS) :

```
/dev/sda2       /var/www/owncloud/data ntfs-3g
rw,umask=007,auto,owner,users,uid=33,gid=33,utf8      0       0
```

Συνοπτικά αυτό σημαίνει ότι ο ιδιοκτήτης του δίσκου – που εδώ έχει label /dev/sda2 – καθώς και όλων των φακέλων σε αυτόν, θα είναι ο χρήστης www-data. Επίσης ο δίσκος θα συνδέεται στο λειτουργικό σύστημα (mount) κάθε φορά που αυτό εκκινεί, και μόνο ο χρήστης www-data (που υπαγορεύεται από το uid=33) και όσοι ανήκουν στο group του (gid=33) θα έχουν δικαιώματα προβολής, επεξεργασίας και εκτέλεσης του αρχείου. Τέλος, δεν θα γίνονται αντίγραφα ασφαλείας του δίσκου από το πρόγραμμα dump, ούτε θα ελέγχεται για σφάλματα κατά την εκκίνηση της συσκευής. Επιπλέον, για να αποτρέψουμε τη χρήση του δίσκου από άλλες υπηρεσίες ή χρήστες, θα δώσουμε αποκλειστικό δικαίωμα πρόσβασης στον δίσκο δεδομένων στο χρήστη www-data, με τις παρακάτω εντολές:

sudo mkdir -p /var/www/owncloud/data #Δημιουργούμε το directory εάν δεν υπάρχει.
sudo chown -R www-data:www-data /var/www/owncloud/data
sudo mount -a #, ώστε να συνδέσουμε όλα τα συστήματα αρχείων που ορίζονται στο /etc/fstab.

Αφού ολοκληρώσουμε τη παραπάνω διαδικασία μπορούμε να αλλάξουμε την τοποθεσία αποθήκευσης σε /var/www/owncloud/data, και στο γραφικό μενού.
Ο δίσκος μας πλέον μπορεί να φιλοξενεί τα δεδομένα στη τοποθεσία που καθορίσαμε, όμως αυτά δεν είναι οργανωμένα ακόμη ώστε να μπορούν να χρησιμοποιούνται. Για το λόγο αυτό πρέπει να ρυθμίσουμε το σχήμα της βάσης δεδομένων της υπηρεσίας.
Από ένα τερματικό, εκκινούμε την υπηρεσία mysql με την εντολή,

mysql -u root -p
Δίνουμε τον κωδικό ασφαλείας, και στη συνέχεια τις εξής εντολές:

```
CREATE DATABASE owncloud_data;

CREATE USER 'owncloud_user'@'localhost' IDENTIFIED BY 'password'; -- όπου
password ο κωδικός που θέλουμε να έχει ο χρήστης owncloud της βάσης δεδομένων,
όχι της υπηρεσίας.
GRANT ALL PRIVILEGES ON owncloud_data.* TO 'owncloud_user'@'localhost';
FLUSH PRIVILEGES;
```

Στη συνέχεια κλείνουμε τη σύνδεση με τη βάση, και στο γραφικό μενού στον browser, συμπληρώνουμε τα πεδία που αντιστοιχούν στη βάση δεδομένων μας ως εξής:

Πεδίο	**Τιμή**
Username	*owncloud_user*
Database	*owncloud_data*
Database password	Εδώ συμπληρώνουμε τον κωδικό που έχει ο χρήστης owncloud_user της βάσης δεδομένων.
location	*localhost* (είναι ήδη συμπληρωμένο το πεδίο αυτό, και δεν το αλλάζουμε)

Τώρα η υπηρεσία Owncloud είναι έτοιμη για χρήση. Πιέζουμε το κουμπί Finish Setup, και εμφανίζεται η αρχική οθόνη χρήστη. Μπορούμε πλέον να ανεβάσουμε αρχεία, να συγχρονίσουμε φακέλους με κάποιον άλλο υπολογιστή ή με κάποια κινητή συσκευή. Η διεύθυνση της υπηρεσίας μας είναι hostname/**owncloud**, όπου hostname το domain name που επιλέξαμε στα παραπάνω βήματα.

4.4 Προώθηση θυρών στον οικιακό δρομολογητή (Port Forwarding)

Προκειμένου να είναι προσβάσιμη η υπηρεσία μας από το διαδίκτυο, θα πρέπει να προωθούμε τα πακέτα που φτάνουν μέχρι τον δρομολογητή του σπιτιού στον server. Για να αποφύγουμε τα πιο συνηθισμένα port scans που απευθύνονται προς ολόκληρα blocks διευθύνσεων, με στόχο να εντοπίσουν ενεργούς servers – στόχους, θα χρησιμοποιήσουμε non-standard ports, τα οποία δεν έχουν συσχετισθεί με κάποια άλλη υπηρεσία [6] . Για την εξυπηρέτηση αιτημάτων HTTP θα χρησιμοποιήσουμε το port 8000 και για την εξυπηρέτηση αιτημάτων HTTPS το port 4431. Με τη βοήθεια του δρομολογητή προωθούμε τα εισερχόμενα πακέτα προς τον server, στα ports 80 και 443.
Εφόσον θέλουμε να καταστήσουμε δυνατή τη πρόσβαση στη συσκευή μας μόνο μέσω του HTTPS, μπορούμε να ζητήσουμε από τον εξυπηρετητή να ανακατευθύνει τα αιτήματα HTTP προς το port 4431. Με αυτό τον τρόπο, πετυχαίνουμε επίσης να διαχωρίσουμε τις ασφαλείς συνδέσεις που ξεκινούν από το εσωτερικό δίκτυο (οι οποίες τελικά καταλήγουν στο port 4431) από τις ασφαλείς συνδέσεις που ξεκινούν από το εξωτερικό δίκτυο με προορισμό το port 8000, αλλάζουν στη συνέχεια το προορισμό με τη παρέμβαση του εξυπηρετητή στο port 4431, και τελικά ανακατευθύνονται από το δρομολογητή στο port 443. Έτσι, αν χρησιμοποιούμε την υπηρεσία και από το οικιακό δίκτυο και από εξωτερικά δίκτυα, μπορούμε να εφαρμόσουμε διαφορετικές πολιτικές χρήστης της υπηρεσίας μας ανάλογα με το port που χρησιμοποιείται.
Για να υλοποιήσουμε τα παραπάνω, πρέπει αφενός να δημιουργήσουμε κανόνες για τη προώθηση θυρών στο δρομολογητή, και αφετέρου να εισάγουμε τις σωστές οδηγίες προς τον εξυπηρετητή Apache, στο αρχείο virtual_hosts.

Η εικόνα των κανόνων στον δρομολογητή είναι η εξής:

Enable	Name	WAN Host Start IP Address	WAN Start Port	LAN Host Start Port	WAN Connection	Modify	Delete
	Protocol	WAN Host End IP Address	WAN End Port	LAN Host End Port	LAN Host Address		
☑	Owncloud		8000	80	vdsl_internet		
	TCP		8000	80	192.168.1.6		
☑	Broker		1883	1883	vdsl_internet		
	TCP		1883	1883	192.168.1.6		
☑	Https-Owr		4431	443	vdsl_internet		
	TCP		4431	443	192.168.1.6		

Ο δεύτερος κανόνας επιτρέπει τη διέλευση μηνυμάτων MQTT, η οποία γίνεται από τη θύρα 1883. Η σκοπιμότητα των μηνυμάτων αυτών αναλύεται στη συνέχεια, όμως προς το παρόν θα εισάγουμε ένα σχετικό κανόνα στο δρομολογητή, ώστε τα μηνύματα που θα αποστέλλονται προς την IP μας, να προωθούνται μέχρι τον MQTT Broker ο οποίος στην παρούσα υλοποίηση φιλοξενείται εντός του οικιακού δικτύου.
Αφού δημιουργήσουμε τους κανόνες, δοκιμάζουμε να πλοηγηθούμε στην ιστοσελίδα μας μέσω ενός άλλου δικτύου (εκτός οικίας), ή δοκιμάζουμε να πλοηγηθούμε χρησιμοποιώντας κάποια υπηρεσία Proxy όπως για παράδειγμα τον Tor Browser. Δεν δοκιμάζουμε από το δίκτυό μας, διότι αρκετοί δρομολογητές παραπέμπουν στη δική τους ιστοσελίδα διαχείρισης όταν δοκιμάσουμε να επισκεφτούμε μια ιστοσελίδα που έχει συσχετιστεί με τη δημόσια IP τους.
Η σύνδεσή μας πρέπει να είναι επιτυχής, και μας παρουσιάζεται η αρχική σελίδα του owncloud, για να εισέλθουμε στην υπηρεσία.
Στη συνέχεια θα εξασφαλίσουμε ότι η κίνηση που θα ανταλλάσουμε με τον κατά τη λήψη και αποστολή αρχείων είναι ασφαλής. Αρχικά θα δημιουργήσουμε τα απαραίτητα πιστοποιητικά που θα αυθεντικοποιούν τον εξυπηρετητή ώστε να μπορεί να διαχειρίζεται τις αιτήσεις των client προς αυτόν. Έπειτα, θα ενεργοποιήσουμε την υποστήριξη HTTPS από τον εξυπηρετητή και θα προσθέσουμε επιπλέον ρυθμίσεις που θα μας βοηθήσουν να αποφύγουμε γνωστές επιθέσεις υποβάθμισης του HTTPS σε HTTP.
Υποθέτοντας μια σύνδεση τερματικού στον εξυπηρετητή, εκτελούμε τις παρακάτω εντολές:

sudo a2dissite 000-default
sudo a2enmod headers && sudo a2enmod ssl # ενεργοποίηση δυνατότητας χρήσης ssl και εισαγωγής κεφαλίδων τύπου HSTS στη συνέχεια.
sudo service apache2 restart

cd /etc/apache2/
sudo cp /etc/ssl/openssl.cnf /etc/ssl/openssl.cnf.bk #δημιουργία backup του αρχικού αρχείου
sudo nano /etc/ssl/openssl.cnf

Στη συνέχεια αλλάζουμε τις εξής γραμμές του αρχείου:

```
dir = /root/SSLCertAuth
default_days = 3650        # 10ετές πιστοποιητικό
default_bits = 2048        # bit ασυμμετρικού κλειδιού της αρχικής φάσης της
                           #μετάδοσης. Θεωρούνται ασφαλή με τα σημερινά
                    #(2016) δεδομένα, από το NIST των ΗΠΑ. Το μέγιστο
#που έχει σπάσει μέχρι στιγμής είναι το κλειδί RSA μήκους 768 bits.

countryName_default = GR
0.organizationName_default = identifier    #αναγνωριστικό του οργανισμού, π.χ
                                           #Owncloud@Home, Ntua κλπ.
```

Το πλήρες αρχείο openssl.cnf παρατίθεται στο κεφάλαιο 10, στο παράρτημα κώδικα του εξυπηρετητή.

Για τη δημιουργία του κλειδιού κρυπογράφησης, θα πρέπει να δημιουργήσουμε την δική μας αρχή πιστοποίησης (CA - Certificate Authority) εκτελώντας τις παρακάτω εντολές:

sudo mkdir /root/SSLCertAuth && sudo chmod 700 /root/SSLCertAuth
sudo -i# πρόσβαση διαχειριστή ώστε να δημιουργήσουμε φακέλους κάτω από τον root φάκελο
cd /root/SSLCertAuth
sudo mkdir certs private newcerts
sudo echo 1000 > serial # θα μπορούσε να είναι οποιοσδήποτε αριθμός, προσδιορίζει μοναδικά κάθε κλειδί.
sudo touch index.txt

Συνεχίζουμε με τη δημιουργία του κλειδιού και την υπογραφή του:

cd SSLCertAuth

openssl req -new -x509 -days 3650 -extensions v3_ca -keyout private/cakey.pem -out cacert.pem -config /etc/ssl/openssl.cnf #δημιουργία CA Certificate, και CA private key που αντιστοιχεί σε αυτό. Τα ονόματά τους προσδιορίζονται στο openssl.cnf

openssl req -new -nodes -out apache-req.pem -keyout private/apache-key.pem -config /etc/ssl/openssl.cnf #δημιουργία Apache Certificate Request (περιέχει το public key), και Apache private key που αντιστοιχεί σε αυτό. Η επιλογή των ονομάτων στις παραμέτρους out και keyout είναι ελεύθερη.

openssl ca -config /etc/ssl/openssl.cnf -out apache-cert.pem -infiles apache-req.pem
#δημιουργία Apache Certificate (περιέχει το public key), από το προηγούμενο request, με τη βοήθεια του CA Certificate που δημιουργήσαμε δύο εντολές πιο πάνω, και του config file.

mkdir /etc/ssl/crt
mkdir /etc/ssl/key
cp /root/SSLCertAuth/apache-cert.pem /etc/ssl/crt
cp /root/SSLCertAuth/private/apache-key.pem /etc/ssl/key
mkdir /var/www/logs #δημιουργία ειδικών φακέλων για τα κλειδιά καθώς και φακέλου για καταγραφή της ροής ssl. Θα χρησιμοποιηθούν στο apache virtual-hosts file.
cp /etc/apache2/sites-available/default-ssl.conf /etc/apache2/sites-available/default-ssl.conf.bk #αντίγραφο ασφαλείας
nano /etc/apache2/sites-available/default-ssl.conf

Στη συνέχεια, αλλάζουμε τα περιεχόμενα του αρχικού αρχείου, προσθέτοντας τα απαραίτητα redirections ώστε να υλοποιείται το σχήμα port forwarding που αναλύσαμε παραπάνω και τις παραπομπές προς το πραγματικό αρχείο private key και certificate. Επίσης, προσθέτουμε την οδηγία ***Header always set Strict-Transport-Security "max-age=15768000; includeSubDomains; preload"*** προκειμένου να αποφευχθούν επιθέσεις υποβάθμισης του https σε απλό http.

Τα περιεχόμενα του αρχείου είναι τα παρακάτω:

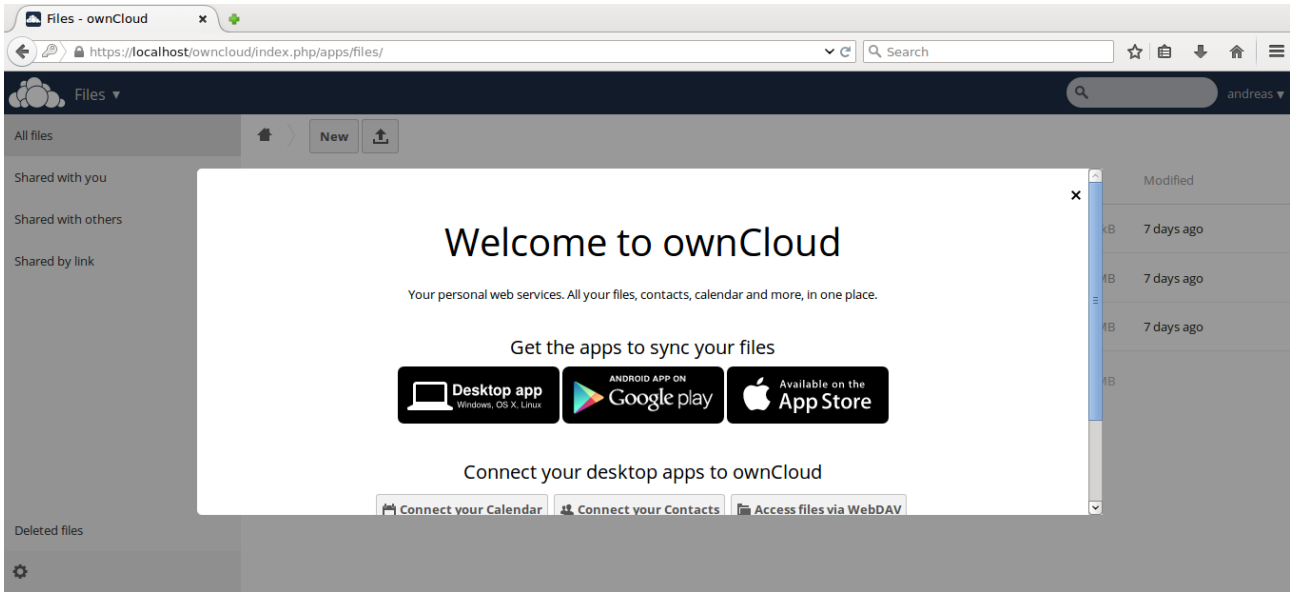
```
<IfModule mod_ssl.c>
        <VirtualHost *:80>
                DocumentRoot /var/www/owncloud
                ServerName owncloud.home.kg
                Redirect /owncloud https://owncloud.home.kg:4431/owncloud
                Redirect / https://owncloud.home.kg:4431/owncloud
        </VirtualHost>
```

```
        <VirtualHost *:4431>
                SSLEngine on
                ServerName 192.168.1.6
                ServerAlias owncloud.home.kg
                SSLCertificateFile /etc/ssl/crt/apache-cert.pem
                SSLCertificateKeyFile /etc/ssl/key/apache-key.pem
                DocumentRoot /var/www/owncloud

                <IfModule mod_headers.c>
                        Header always set Strict-Transport-Security "max-
age=15768000; includeSubDomains; preload"
                </IfModule>


                <FilesMatch "\.(cgi|shtml|phtml|php)$">
                    SSLOptions +StdEnvVars
                </FilesMatch>
                <Directory /usr/lib/cgi-bin>
                    SSLOptions +StdEnvVars
                </Directory>

                BrowserMatch "MSIE [2-6]" \
                nokeepalive ssl-unclean-shutdown \
                downgrade-1.0 force-response-1.0
                BrowserMatch "MSIE [17-9]" ssl-unclean-shutdown

         </VirtualHost>

         <VirtualHost _default_:443>
                ServerAdmin webmaster@localhost
                ServerName 192.168.1.6
                ServerAlias owncloud.home.kg
                SSLEngine on
                SSLCertificateFile /etc/ssl/crt/apache-cert.pem
                SSLCertificateKeyFile /etc/ssl/key/apache-key.pem
                DocumentRoot /var/www/owncloud

                <IfModule mod_headers.c>
                                Header always set Strict-Transport-Security
"max-age=15768000; includeSubDomains; preload"
                </IfModule>



                CustomLog /var/www/logs/ssl-access_log vhost_combined
                ErrorLog /var/www/logs/ssl-error_log

                <FilesMatch "\.(cgi|shtml|phtml|php)$">
                                SSLOptions +StdEnvVars
                </FilesMatch>
                <Directory /usr/lib/cgi-bin>
                                SSLOptions +StdEnvVars
                </Directory>

                BrowserMatch "MSIE [2-6]" \
                                nokeepalive ssl-unclean-shutdown \
                                downgrade-1.0 force-response-1.0
                BrowserMatch "MSIE [17-9]" ssl-unclean-shutdown

        </VirtualHost>
</IfModule>
```

Τέλος, εκτελούμε τις εντολές
a2ensite default-ssl.conf
service apache reload
exit

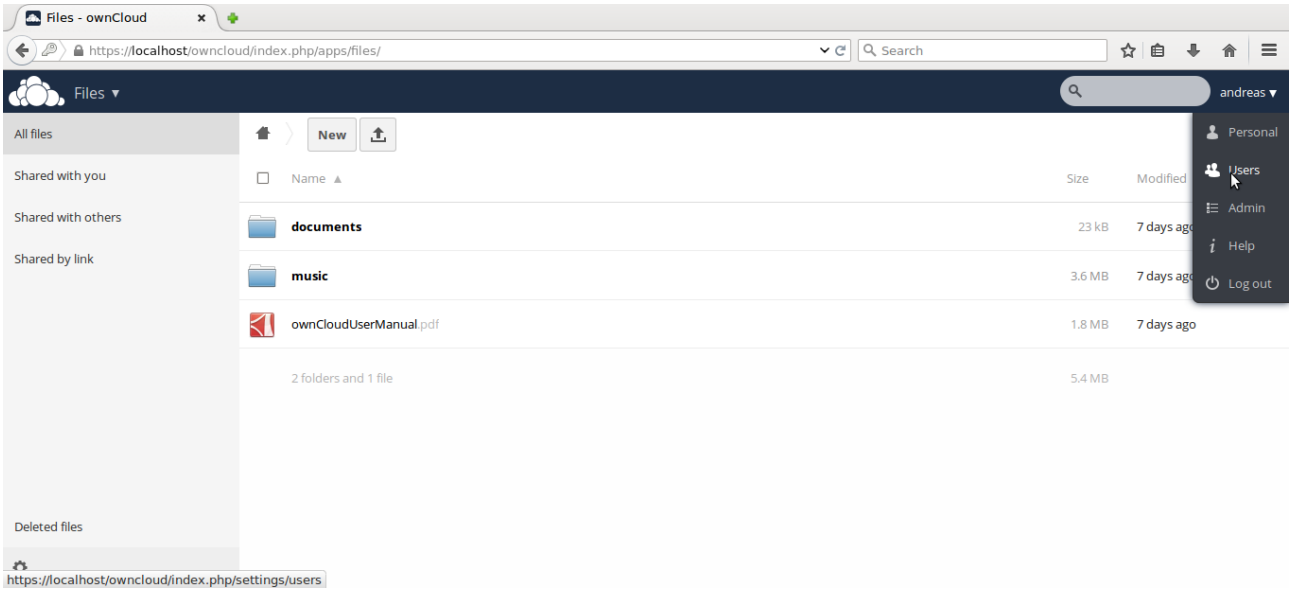
Στο σημείο αυτό μπορούμε να ελέγξουμε την υπηρεσία μας, πληκτρολογώντας τη διεύθυνσή της από ένα οποιονδήποτε browser, αποδεχόμενοι το πιστοποιητικό που δημιουργήσαμε και υπογράψαμε οι ίδιοι. Παρατηρούμε ότι η σύνδεση γίνεται με τη χρήση HTTPS.

Η εικόνα μετά τη πρώτη σύνδεση είναι η παρακάτω:

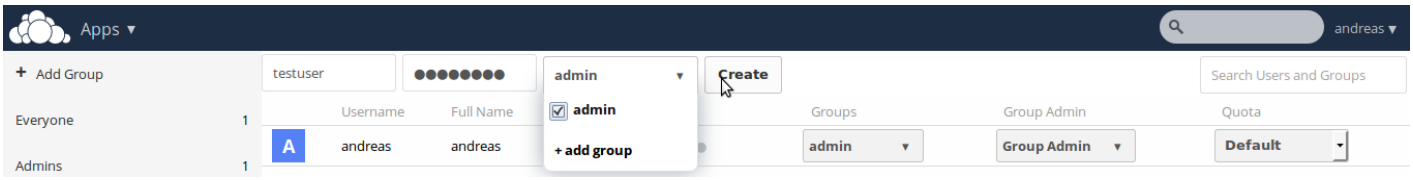


Στη συνέχεια, θα προσθέσουμε στην υπηρεσία μας ακόμη ένα χρήστη που θα έχει διαχειριστικά δικαιώματα, δηλαδή θα μπορεί να ελέγχει τους άλλους χρήστες και να αλλάζει τις ρυθμίσεις του Owncloud-Server.

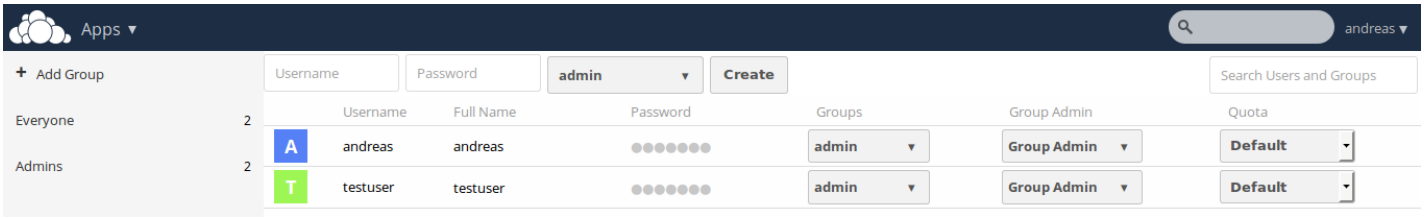
Αρχικά ανοίγουμε το μενού δεξιά από το όνομα του χρήστη, και επιλέγουμε Users:



Στη συνέχεια, προσθέτουμε το επιθυμητό όνομα χρήστη και ένα κωδικό, προσδιορίζουμε ότι θέλουμε να ανήκει στην ομάδα των διαχειριστών (μπορούμε να το κάνουμε, γιατί ο πρώτος λογαριασμός είναι λογαριασμός διαχειριστή) και πιέζουμε Create:



Τελικά μπορούμε να δούμε ότι δημιουργήθηκε ο δεύτερος χρήστης:



Με τον τρόπο που περιγράφηκε παραπάνω, μπορούμε να δημιουργήσουμε χρήστες για όλα τα μέλη ενός σπιτιού, ή για μια μικρή ομάδα.

5 Εφαρμογές Android και βασικός κώδικας της εφαρμογής Client

5.1 Βασικές Παρατηρήσεις για το προγραμματισμό σε Android

Για μια εφαρμογή Android, ιδιαίτερο ρόλο έχουν οι υπηρεσίες (services) και οι δραστηριότητες (activities) που είναι ανά πάσα στιγμή ενεργές. Οι υπηρεσίες χρησιμοποιούνται για εργασίες που μπορούν να εκτελούνται στο παρασκήνιο, συνήθως για αρκετά μεγάλο χρονικό διάστημα, ενώ οι δραστηριότητες για εργασίες που αφορούν την αλληλεπίδραση χρήστη και συσκευής.
Τόσο οι υπηρεσίες όσο και οι δραστηριότητες υλοποιούνται με κλάσεις παρόμοιες με αυτές που συναντούμε στη Java, οι οποίες όμως εκτός των κανονικών μεθόδων τους, υποχρεούνται να υλοποιήσουν συγκεκριμένες μεθόδους στις οποίες στηρίζεται η εκτέλεσή της εφαρμογής. Η εκτέλεση των μεθόδων αυτών έχει ασύγχρονο χαρακτήρα και βασίζεται στα γεγονότα που συμβαίνουν κατά την αλληλεπίδραση με τον χρήστη. Κάποια από τα πιο βασικά γεγονότα για μια υπηρεσία ή δραστηριότητα είναι η δημιουργία της, η προσωρινή παύση, η επαναφορά και ο τερματισμός [7].
Κατά παρόμοιο τρόπο δομούνται και οι υπηρεσίες. Μια υπηρεσία μπορεί να προγραμματιστεί ώστε να εκτελέσει ειδικό κώδικα όταν δημιουργείται, όταν εκκινεί, πριν σταματήσει αλλά και όταν κάποια άλλη δραστηριότητα ζητήσει να συνδεθεί (bind) μαζί της [8].
Η επικοινωνία μεταξύ υπηρεσιών και δραστηριοτήτων επιτυγχάνεται με μια ειδική δομή, τον Broadcast Receiver, ο οποίος λαμβάνει intents, που είναι μηνύματα που μπορούν να περιλαμβάνουν και δεδομένα. Με βάση τα μηνύματα που λαμβάνει, ο Broadcast Receiver εκτελεί τον ανάλογο κώδικα.
Στο τομέα της αλληλεπίδρασης του χρήστη με το λειτουργικό σύστημα, το κύριο μέσο για τη διέγερση της προσοχής του μετά από κάποια σημαντικά γεγονότα είναι το γραφικό στοιχείο Toast. Πρόκειται για μια πολύ ευέλικτη κλάση ειδοποιήσεων, που επιτρέπει να εμφανίζεται στην οθόνη του χρήστη, κείμενο, εικόνα, ακόμη και γραφικά στοιχεία. Μία ακόμη χρήσιμη δυνατότητα αλληλεπίδρασης είναι οι στατικές ή οι δυναμικές διατάξεις (layouts), που μπορούν να περιλαμβάνουν κουμπιά, κυλιόμενα μενού κλπ. Στην εφαρμογή μας θα χρησιμοποιήσουμε μια στατική διάταξη, που θα ενημερώνει το χρήστη για την ανάγκη επανασύνδεσης του δίσκου μετά από μια περίοδο αδρανείας.
Το λειτουργικό σύστημα ενημερώνεται για την ύπαρξη υπηρεσιών και δραστηριοτήτων από το ειδικό αρχείο ρυθμίσεων, AndroidManifest.xml, το οποίο βρίσκεται στον βασικό φάκελο (root folder) της κάθε εφαρμογής.

5.2 Ροή εκτέλεσης μιας εφαρμογής Android

Η ροή της εκτέλεσης σε μια εφαρμογή android, ξεκινά από την κλάση application, όπως αυτή δηλώνεται στο AndroidManifest.xml της εφαρμογής. Αμέσως μετά την εκτέλεση της μεθόδου onCreate αυτής της κλάσης, εκκινεί η μέθοδος onCreate της δραστηριότητας περιλαμβάνει τη δήλωση για τη δράση android.intent.action.MAIN και την κατηγορία

android.intent.category.LAUNCHER [9]. Στο παρακάτω παράδειγμα κώδικα, η δραστηριότητα που εκκινεί είναι η MainActivity, που βρίσκεται μάλιστα στο βασικό φάκελο της εφαρμογής:

```
<activity android:name=".MainActivity" android:label="@string/app_name">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

Από το σημείο αυτό και στη συνέχεια, καλούνται όλες οι υπόλοιπες δραστηριότητες και υπηρεσίες που απαρτίζουν την εφαρμογή, και αποκρίνονται στις δεδομένα που προκύπτουν από τις δραστηριότητες του χρήστη.

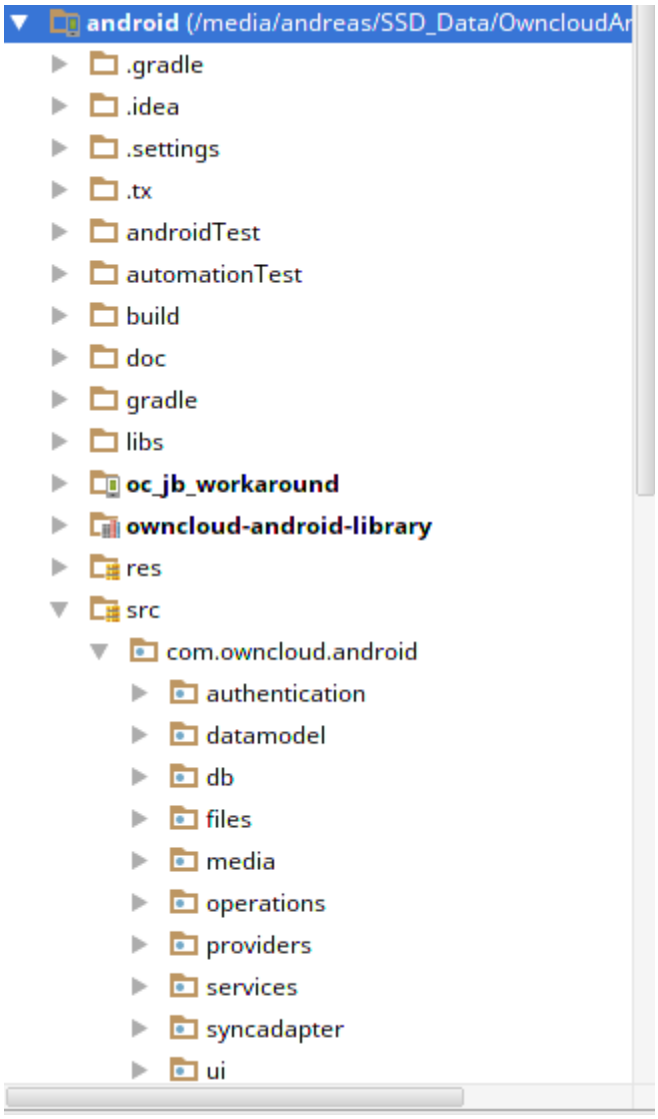
5.3 Εγκατάσταση της εφαρμογής owncloud-android

Για την υλοποίηση του ασφαλούς client, μία από τις βασικές εφαρμογές που θα χρησιμοποιήσουμε, είναι η υπάρχουσα έκδοση της εφαρμογής owncloud-android. Η εφαρμογή είναι υπεύθυνη για το συγχρονισμό των αρχείων μεταξύ μιας συσκευής android και του Owncloud-Server. Περιλαμβάνει κατάλληλο γραφικό περιβάλλον, και ένα βασικό περιηγητή για τα αρχεία που είναι αποθηκευμένα στον εξυπηρετητή, ώστε ο χρήστης να μπορεί να κατεβάζει αρχεία στη συσκευή και να ανεβάζει αρχεία στο φάκελο που επιθυμεί. Επιπλέον υποστηρίζει και άλλες δυνατότητες των σύγχρονων client, όπως τη δυνατότητα να ανεβάζει αμέσως φωτογραφίες και βίντεο που παίρνει ο χρήστης στον Owncloud-Server.
Αρχικά, θα πρέπει να κατεβάσουμε τον πηγαίο κώδικα της εφαρμογής, προκειμένου να διαπιστώσουμε ότι λειτουργεί σωστά με την υπηρεσία μας. Για να το πετύχουμε αυτό, από ένα νέο τερματικό εκτελούμε τις εξής εντολές:

```
sudo apt-get install git
cd directory_of_project    # ο φάκελος που θέλουμε να περιέχει το project
git clone --recursive  https://github.com/owncloud/android
```

Στη συνέχεια, θα πρέπει να εγκαταστήσουμε το Android Studio, καθώς και το Android SDK, από τη σχετική ιστοσελίδα της Google [10].
Μετά την εγκατάσταση, μπορούμε να εισάγουμε το project που κατεβάσαμε παραπάνω από το μενού import existing project. Έχουμε πλέον μπροστά μας το καινούργιο project με όλες τις κλάσεις που σχετίζονται με αυτό. Πιέζουμε το κουμπί Run, και η εφαρμογή μεταγλωττίζεται και εφόσον έχουμε συνδεδεμένο το κινητό μας ή κάποια άλλη εικονική συσκευή ξεκινά να τρέχει. Μπορούμε να παρατηρήσουμε στο κάτω μέρος του android studio, το αρχείο καταγραφής της εφαρμογής να ανανεώνεται καθώς αυτή λειτουργεί.
Ο κώδικας της εφαρμογής είναι οργανωμένος σε τρία πακέτα, το android, το οποίο είναι το εξωτερικό και τα oc_jb_workaround και owncloud-android-library, τα οποία περιλαμβάνονται σε αυτό. Στο φάκελο src, περιέχονται οι κλάσεις που θα χρειαστεί να τροποποιήσουμε, και είναι ο γενικός κώδικας της εφαρμογής. Εκεί υπάρχει και το πακέτο com.owncloud.android, το οποίο περιέχει τις κλάσεις της εφαρμογής κατά κατηγορίες.

Από το AndroidManifest.xml για το module android, που βρίσκεται στο βασικό φάκελο της εφαρμογής, βλέπουμε ότι αφού η εφαρμογή εκτελέσει τον κώδικα της MainApp (Application class), συνεχίζει με τη δραστηριότητα FileDisplayActivity.java. Η FileDisplayActivity είναι υπεύθυνη για την αρχικοποίηση του βασικού μενού της εφαρμογής, από το οποίο ο χρήστης βλέπει τον κατάλογο των αρχείων του.
Παρακάτω βλέπουμε τον κώδικα της εφαρμογής όπως φαίνεται αμέσως μετά την εισαγωγή του στο Android Studio.



5.4 Εφαρμογή mqtt-client

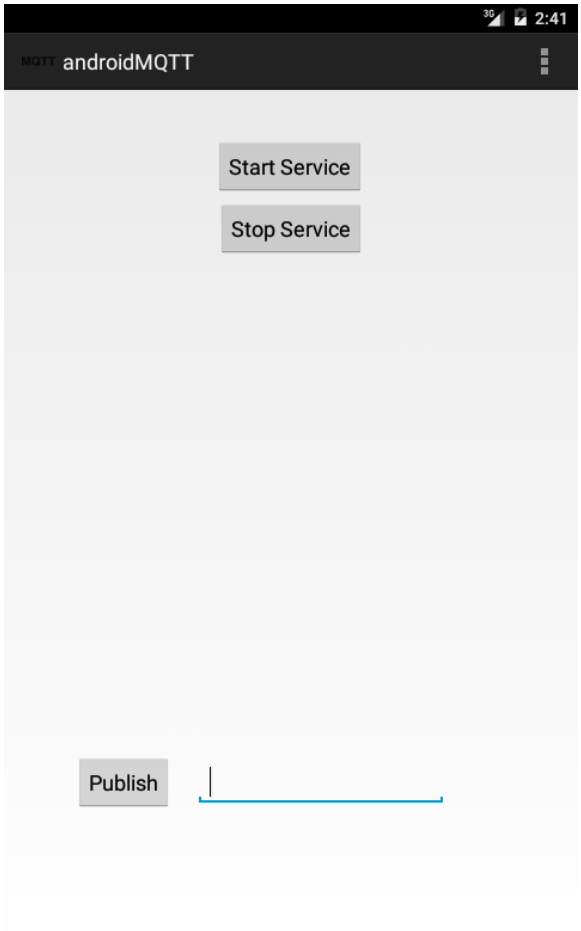
Εκτός από την εφαρμογή owncloud-android, που θα είναι υπεύθυνη για την επικοινωνία με τον εξυπηρετητή, προκειμένου να αποστέλλουμε μηνύματα στο αποθηκευτικό μέσο, θα χρειαστούμε τις δυνατότητες που προσφέρει ένας MQTT Client. Στη συγκεκριμένη εργασία θα χρησιμοποιήσουμε μια υλοποίηση που έχει γίνει από το χρήστη AshuJoshi του github:
https://github.com/AshuJoshi/androidMQTT/tree/master/androidMQTT
Η υλοποίηση βασίζεται σε ένα υποδειγματικό άρθρο του Dale Lane σχετικό με τη χρήση του mqtt στο Android [11].
Για να μπορέσουμε να χρησιμοποιήσουμε στη συνέχεια τα στοιχεία της εφαρμογής mqtt-client, είναι σημαντικό να επιβεβαιώσουμε την ορθή λειτουργία της στο περιβάλλον εργασίας μας.
Αρχικά θα κατεβάσουμε την εφαρμογή mqtt-client, εκτελώντας την εξής εντολή, στον φάκελο που θέλουμε:

git clone https://github.com/AshuJoshi/androidMQTT

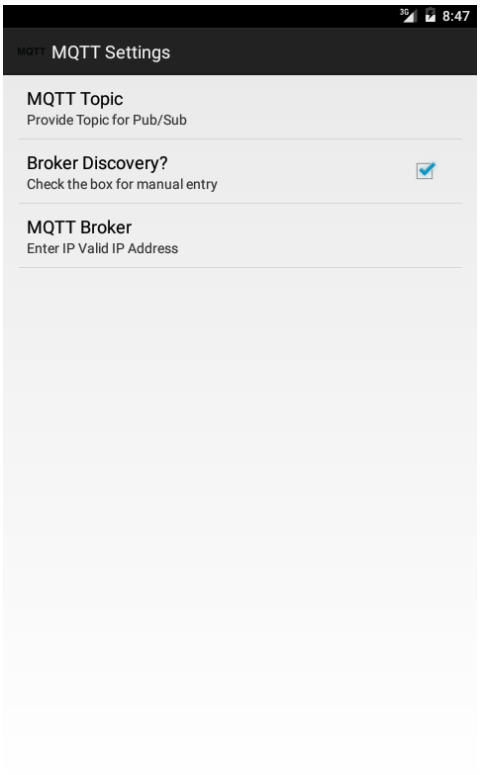
Στη συνέχεια, ανοίγουμε το πρόγραμμα Android Studio, και πηγαίνουμε στο μενού File → New →Import Project. Παρουσιάζεται μία σειρά διαλόγων, στην οποία πρέπει να επιλέξουμε ότι θέλουμε να εργαστούμε με τα υπάρχοντα αρχεία, να επιλέξουμε το σωστό SDK ανάλογα με τις συσκευές που θα χρησιμοποιηθούν, και να επιβεβαιώσουμε τη δομή του project.
Για να έχουμε πλήρη τη λειτουργικότητα της εφαρμογής, πρέπει να κατεβάσουμε τη βιβλιοθήκη wmqtt.jar, η οποία ανήκει στην IBM, και την οποία μπορούμε να βρούμε ως μέρος του πακέτου IA92.zip, από την εξής τοποθεσία:

ftp://public.dhe.ibm.com/software/integration/support/supportpacs/individual/ia92.zip

Η βιβλιοθήκη βρίσκεται στο φάκελο J2SE του συμπιεσμένου αρχείου. Την αντιγράφουμε στο βασικό φάκελο (root folder) της εφαρμογής (μαζί με τους φακέλους gen,res, και src) και κάνοντας δεξί κλικ στη βιβλιοθήκη επιλέγουμε “Add As Library…”. Αφού προσθέσουμε τη βιβλιοθήκη, ο MQTT Client είναι έτοιμος να μεταγλωττιστεί και να τρέξει.
Όπως μπορούμε να δούμε από το αρχείο AndroidManifest.xml, η εφαρμογή mqtt-client, εκκινεί τη δραστηριότητα MQTTActivity, η οποία περιλαμβάνει μια απλή διεπαφή για τη δημοσίευση των μηνυμάτων. Μετά την εκκίνηση μπορούμε από μια άλλη διεπαφή να καθορίσουμε τη διεύθυνση του Broker και το topic στο οποίο θα στέλνουμε μηνύματα, και για το οποίο θα ενημερωνόμαστε.
Πολύ σημαντικό ρόλο στη λειτουργία της εφαρμογής έχει η υπηρεσία MQTTService, που είναι υπεύθυνη για το χειρισμό όλων των γεγονότων MQTT.
Παρακάτω μπορούμε να δούμε μερικά στιγμιότυπα από τη χρήση της εφαρμογής:



Η βασική οθόνη της εφαρμογής, αποστέλλει μηνύματα πιέζοντας το Publish, αφού πρώτα έχουμε πιέσει το Start Service.



Η οθόνη προτιμήσεων της εφαρμογής

6 Δημιουργία ασφαλούς εφαρμογής Client

6.1 Σχεδιασμός της εφαρμογής – πελάτη (Client)

Για την εφαρμογή του μοντέλου για τον πελάτη που περιγράφηκε στην αρχή της εργασίας, θα χρησιμοποιήσουμε ως βάση την εφαρμογή android του Owncloud, και την παραπάνω υλοποίηση του MQTT Client. Επίσης, θα περιλαμβάνονται μέθοδοι που θα ρυθμίζουν τη σύνδεση και την αποσύνδεση του δίσκου με σκοπό την ασφάλεια. Θα προσθέσουμε τη λειτουργικότητα του MQTT Client επάνω στον Owncloud Client γιατί ο κώδικας του τελευταίου έχει πολύ μεγαλύτερη έκταση.

6.2 Εφαρμογή owncloud-android

Έχοντας πλέον εισάγει την εφαρμογή στο περιβάλλον ανάπτυξης, μπορούμε να μεταβάλλουμε τις κλάσεις της εφαρμογής, ώστε να υλοποιούν το σχέδιο που περιγράψαμε στη σχετική ενότητα. Θα πρέπει δηλαδή το αποθηκευτικό μέσο της υπηρεσίας να είναι προσβάσιμο μόνο όταν υπάρχει ανάγκη ανάγνωσης ή αποθήκευσης σε αυτό, και να είναι αποσυνδεδεμένο κάθε άλλη στιγμή.
Την απαίτηση αυτή θα την ικανοποιήσουμε κατά κύριο λόγο προσθέτοντας κώδικα στη δραστηριότητα FileDisplayActivity, η οποία όπως αναφέραμε παραπάνω παρουσιάζει στο χρήστη τα αρχεία που είναι αποθηκευμένα στην υπηρεσία. Η δραστηριότητα είναι υποχρεωμένη να αλληλεπιδρά με το αποθηκευτικό μέσο κάθε φορά που εκκινεί η εφαρμογή προκειμένου να λάβει τον κατάλογο των υπαρχόντων αρχείων. Όταν δεν χρησιμοποιείται, δεν υπάρχει ανάγκη να επιτρέπουμε την πρόσβαση στο αποθηκευτικό μέσο.
Οι παραπάνω απαιτήσεις ταιριάζουν με τις μεθόδους που εκτελούνται στα διάφορα στάδια του κύκλου ζωής της δραστηριότητας. Κατά την έναρξη ή τη συνέχιση της δραστηριότητας (εκτέλεση μεθόδου onResume) θα πρέπει να ειδοποιούμε με κάποιο τρόπο το αποθηκευτικό μέσο ώστε να συνδεθεί (μήνυμα MQTT ON), και όταν ο χρήστης δεν έχει πλέον τη προσοχή του στην εφαρμογή ή την τερματίζει (μέθοδος onPause, παρέλευση 5' χωρίς κάποια ενέργεια του χρήστη), θα ειδοποιούμε αντίστοιχα το αποθηκευτικό μέσο να αποσυνδεθεί, στέλνοντας ένα μήνυμα OFF με τη χρήση του MQTT. Οι μέθοδοι θα ενημερώνουν το χρήστη για τις αλλαγές αυτές ώστε να ρυθμίζει τη πρόσβαση στο περιεχόμενό του δρώντας κατάλληλα.

Ο πίνακας που ακολουθεί συνοψίζει τις κυριότερες λειτουργίες που θα αποκτήσει κάθε κλάση που θα αναλύσουμε παρακάτω:

Όνομα κλάσης	**Λειτουργία**
FileDisplayActivity.java	Η κλάση αυτή είναι υπεύθυνη για τη παρουσίαση των αρχείων στο χρήστη, και συντονίζει την αποστολή μηνυμάτων MQTT προς τον εξυπηρετητή ώστε να έχουμε ασφάλεια στα δεδομένα.

MQTTService.java	Η κλάση αυτή είναι υπεύθυνη για την αποστολή και τη λήψη μηνυμάτων MQTT, και για την ενημέρωση της FileDisplayActivity.
ConnectivityActionReceiver.java	Η κλάση αυτή είναι υπεύθυνη για την επανενεργοποίηση της σύνδεσης MQTT, όταν ανιχνευτεί ότι η σύνδεση στο διαδίκτυο επανέλθει.
DiskUsageService.java	Η κλάση αυτή εξετάζει κατά πόσο η εφαρμογή είναι αδρανής ή όχι, και αντίστοιχα επικοινωνεί με την FileDisplayActivity ώστε να πληροφορηθεί ο χρήστης
Preferences.java	Στη κλάση αυτή συμπεριλαμβάνονται μεταξύ άλλων και οι ρυθμίσεις που αφορούν την επικοινωνία μέσω MQTT.

Παρακάτω παραθέτουμε τον κώδικα της εφαρμογής ανά κλάση, αναλυτικά και με σχόλια. Παρουσιάζονται οι μεταβολές στον υπάρχοντα κώδικα αλλά και καινούργιος κώδικας, οι επιπλέον μεταβλητές που πρέπει να ορίσουμε και οι επιπλέον εισαγόμενες κλάσεις (import statements) για κάθε κλάση που μεταβάλλουμε ή ορίζουμε στην εφαρμογή.
Όπως θα γίνει αντιληπτό, επιζητούμε μέγιστη αυτονομία των κλάσεων στον κώδικά μας, ώστε να είναι εύκολα τροποποιήσιμος. Για το λόγο αυτό η επικοινωνία μεταξύ όλων των συστημάτων που εισάγουμε στην εφαρμογή, γίνεται με τη χρήση BroadcastReceiver και LocalBroadcastManager.
Εξαίρεση αποτελεί το πεδίο last_action_timestamp της κλάσης Log_OC.java, το οποίο χρησιμοποείται από άλλη κλάση με τη χρήση μιας μεθόδου λήψης (getter method) διότι θεωρούμε ότι αποτελεί ουσιαστικό στοιχείο της (τροποποιημένης) κλάσης Log_OC.
Τα πλήρη αρχεία που περιέχουν επιπλέον και τον αλλαγμένο κώδικα, παρατίθενται στο τέλος της εργασίας σε σχετικό παράρτημα.

6.3 Κλάση FileDisplayActivity.java.

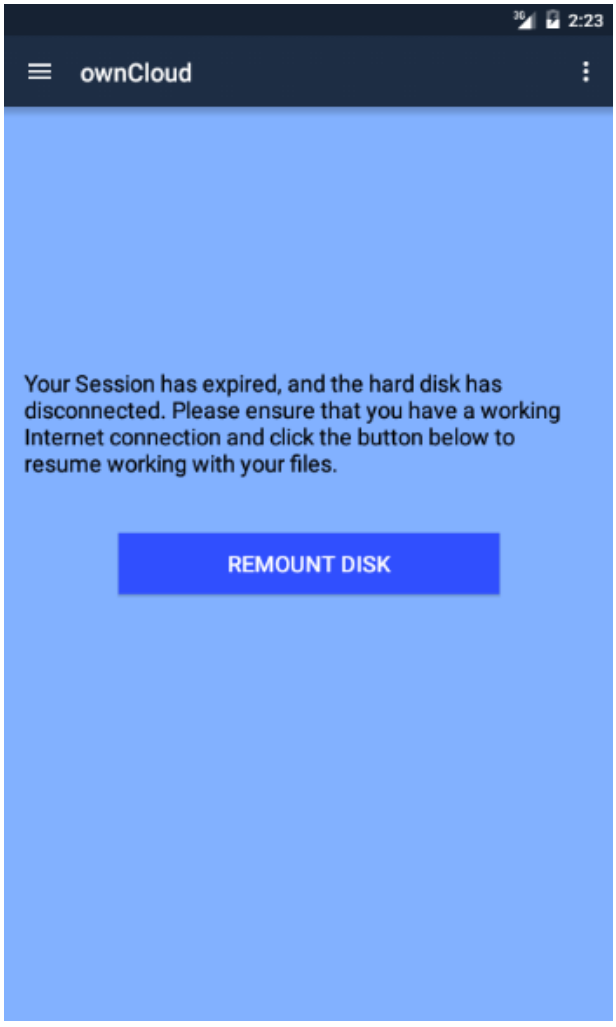
Γραφικό σκέλος αλλαγών στη δραστηριότητα FileDisplayActivity

Για την ενημέρωση του χρήστη στη περίπτωση που δεν παρατηρηθεί δραστηριότητα για αρκετά μεγάλο χρονικό διάστημα, η εφαρμογή θα πρέπει να εμφανίζει σχετική οθόνη, και παράλληλα να αποσυνδέει το δίσκο. Για την υλοποίηση της οθόνης, στο φάκελο res/layout της εφαρμογής, δημιουργούμε ένα καινούργιο αρχείο xml με όνομα hard_disk_reconnect.xml, με τα παρακάτω περιεχόμενα:

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/disk_layout"
    android:layout_width="wrap_content"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
```

```
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".ui.activity.FileDisplayActivity"
    android:background="#82b1ff">
    <Button
        android:id="@+id/remount_disk"
        android:layout_width="300dp"
        android:layout_height="wrap_content"
        android:text="@string/reconnect_drive"
        android:layout_centerVertical="true"
        android:layout_centerHorizontal="true"
        android:background="#304ffe"
        android:textColor="@color/white"
        android:textSize="@dimen/abc_text_size_medium_material" />
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceMedium"
        android:text="Your Session has expired, and the hard disk has disconnected.
Please ensure that you have a working Internet connection and click the button below to
resume working with your files."
        android:id="@+id/textView2"
        android:layout_above="@+id/remount_disk"
        android:layout_centerHorizontal="true"
        android:layout_marginBottom="43dp"
        android:textColor="@color/black" />
</RelativeLayout>
```

Η οθόνη που εμφανίζεται για να πληροφορήσει το χρήστη είναι η παρακάτω:



Για την εκτύπωση του κειμένου, θα ορίσουμε ένα string με αναγνωριστικό reconnect_drive στο αρχείο strings.xml στο μονοπάτι res/values/strings.xml, προσθέτοντας την εξής δήλωση:

```
<string name="reconnect_drive">Remount Disk</string>
```

Αλλαγές στις μεθόδους onStart, onCreate, onStop, και ορισμός της βοηθητικής μεθόδου publishMessage

Η μέθοδος onCreate της FileDisplayActivity εκτελείται πρώτη μετά την έναρξη της εφαρμογής, και την εκτέλεση της Application class MainApp.java. Στη FileDisplayActivity εκκινούμε τις υπηρεσίες MQTTService και DiskUsageService, οι οποίες είναι υπεύθυνες για την αποστολή και λήψη μηνυμάτων MQTT, και για τη τήρηση των χρονικών ορίων ασφαλείας κατά τη χρήση του δίσκου αντίστοιχα. Επίσης, δημιουργούμε ένα λήπτη μηνυμάτων (BroadCastReceiver) με όνομα receiver που λαμβάνει μηνύματα από τις παραπάνω υπηρεσίες.
Για την ορθή λειτουργία του λήπτη, θα πρέπει να τον δηλώσουμε στη μέθοδο OnStart, μαζί με τα μηνύματα που λαμβάνει, γράφοντας τον παρακάτω κώδικα:

```
IntentFilter actions = new IntentFilter("com.diskusageservice.remount_action");
actions.addAction("com.mqttservice.message_received");
LocalBroadcastManager.getInstance(this).registerReceiver((receiver),actions);
```

Επίσης, θα πρέπει να τον αποδεσμεύσουμε στη μέθοδο onStop, με τον παρακάτω κώδικα:

```
LocalBroadcastManager.getInstance(this).unregisterReceiver(receiver);
```

Όταν ο λήπτης λάβει μήνυμα επιτυχούς αποστολής μηνύματος από την mqttService, το συγκρίνει με το μήνυμα που εκκρεμεί (pending_message) και ανάλογα τυπώνει στο αρχείο καταγραφής κατάλληλο διαγνωστικό μήνυμα και ενημερώνει τη μεταβλητή mqtt_message_received. Επίσης όταν λαμβάνει μήνυμα που μας πληροφορεί για την αδράνεια του συστήματος για αρκετό χρόνο παρουσιάζει τη σχετική οθόνη που ορίσαμε παραπάνω στο χρήστη. Όπως παρατηρούμε, το layout που ορίσαμε περιλαμβάνει την επιλογή να επανασυνδέσει το δίσκο και να συνεχίσει τη χρήση της εφαρμογής. Για το λόγο αυτό διαχειριζόμαστε το γεγονός της πίεσης του κουμπιού REMOUNT DISK, επανεκκινώντας την δραστηριότητα FileDisplayActivity.

Για την υλοποίηση των παραπάνω θα εισάγουμε τις παρακάτω κλάσεις:

```
import android.support.v4.content.LocalBroadcastManager;
import com.owncloud.android.services.MQTTService;
import android.util.Log;
import com.owncloud.android.services.DiskUsageService;
import android.widget.Button;
```

Στο τμήμα δηλώσεων θα προσθέσουμε τις παρακάτω μεταβλητές:

```
private static boolean mBound;
private static MQTTService mService;
BroadcastReceiver receiver;
LocalBroadcastManager broadcaster;
private static Intent mqttService;
private static Intent diskusageService;
private boolean mqtt_message_received;
private static String pending_message;
public static ServiceConnection mConnection = new ServiceConnection() {

        // Περιγράφουμε τι θα γίνει όταν θα συνδεθεί η FileDisplayActivity με την
mqttService
        @Override
        public void onServiceConnected(ComponentName className,
                                       IBinder service) {
            Log.d(TAG, "Inside onServiceConnected");
            MQTTService.LocalBinder binder = (MQTTService.LocalBinder) service;
            mService = (MQTTService)binder.getService();
            mBound = true;
         // Η αποστολή μηνύματος για τη σύνδεση του δίσκου γίνεται στη μέθοδο onResume.
        }
        // Περιγράφουμε τι θα γίνει όταν αποσυνδεθεί η δραστηριότητα από την υπηρεσία
        // Δεν αποστέλλεται μήνυμμα OFF γιατί δεν ξέρουμε αν θα εκτελεστεί η μέθοδος
        // διότι η σύνδεση δραστηριότητας – υπηρεσίας γίνεται με την παράμετρο
        // Context.BIND_AUTO_CREATE
        @Override
        public void onServiceDisconnected(ComponentName arg0) {
            Log_OC.v(TAG, "MQTT Service disconnected");
            mBound = false;
        }
    };
```

Ακολούθως ορίζουμε τη μέθοδο publishMessage. Αν έχει συνδεθεί η δραστηριότητα (mBound = true) με το αντικείμενο-υπηρεσία mService, η μέθοδος προωθεί το μήνυμα για αποστολή με τις μεθόδους της MQTTService, διαφορετικά εκτυπώνει κατάλληλο διαγνωστικό μήνυμα.

```
public static void publishMessage(String message){
    if (mBound) {
        mService.publishMessageToTopic(message); //Αν η mBound είναι αληθής, τότε η
αναφορά στην mService θα είναι έγκυρη
    }
    else {
        Log.v(TAG, "Tried to publish but mBound is false");
    }
    pending_message = message;
}
```

Στη συνέχεια, στη μέθοδο onCreate, και πριν την εκτέλεση της εντολής super.onCreate, θα προσθέσουμε τον παρακάτω κώδικα:

```
        broadcaster = LocalBroadcastManager.getInstance(this);
        receiver = new BroadcastReceiver() {
            @Override
            public void onReceive(Context context, Intent intent) {
                if (intent.getAction().equals("com.mqttservice.message_received")) {
```

```
                    if
(pending_message.equals(intent.getStringExtra("message_received"))) {
                        mqtt_message_received = true;
                        Log.v(TAG, "The pending message has been received");
                    } else {
                        Log.v(TAG, "A message different from the pending has been
received");
                    }
                } else if
(intent.getAction().equals("com.diskusageservice.remount_action")) {
                    publishMessage("OFF");//λάβαμε το μήνυμα για να αποσυνδέσουμε το
δίσκο οπότε τον αποσυνδέουμε και δείχνουμε την οθόνη με πληροφορίες για την επανασύνδεση
                    setContentView(R.layout.hard_disk_reconnect);
                    Log.d(TAG, "message from notifyDiskDisconnected received - disk
unmounted");
                    final Intent reply_intent = new Intent(FileDisplayActivity.this,
DiskUsageService.class);

reply_intent.setAction("com.filedisplayactivity.broadcast_received");
                    broadcaster.sendBroadcast(reply_intent); //αποστολή μηνύματος στη
DiskUsageService, ότι ο χρήστης είναι ενήμερος για τη παρέλευση μεγάλου χρονικού
διαστήματος αδρανείας του δίσκου
                    Button RemountButton = (Button) findViewById(R.id.remount_disk);
                    RemountButton.setOnClickListener(new View.OnClickListener() {
                        @Override
                        public void onClick(View v) {
                            // TODO Auto-generated method stub
                            Log.d(TAG, "Remount Button Clicked");
                            stopService(diskusageService);
                            stopService(mqttService);
                            unbindService(mConnection);
                            mBound = false; //η onServiceDisconnected καλείται μόνο
όταν η mBound τερματίζει απρόοπτα, επομένως δεν μπορούμε να μεταφέρουμε εκεί αυτή την
εντολή.


reply_intent.setAction("com.filedisplayactivity.user_notified_reset");
                            broadcaster.sendBroadcast(reply_intent);
                            FileDisplayActivity.this.recreate(); // Επανέναρξη της
δραστηριότητας και αποστολή του μηνύματος ON για επανασύνδεση του δίσκου κατά τη κλήση
της onResume.
                        }
                    });
                }
            }
        };
        //Δημιουργία της υπηρεσίας mqttService, τύπου MQTTService
        mqttService = new Intent(this, MQTTService.class);
        startService(mqttService);
        //Δημιουργία της υπηρεσίας diskusageService, τύπου MQTTService
        Log.v(TAG, "Disk Service Starting");
        diskusageService = new Intent(this, DiskUsageService.class);
        startService(diskusageService);
        Log.v(TAG, "Disk Service Started");
```

Αλλαγές στη μέθοδο onResume

Η μέθοδος onResume διασφαλίζει ότι μετά από μια προσωρινή διακοπή, θα συνεχίσουμε να έχουμε πρόσβαση στα αρχεία με ασφαλή τρόπο. Πρώτα, θα ενημερώσουμε την υπηρεσία DiskUsageService, ότι η FileDisplayActivity βρίσκεται στο προσκήνιο της εκτέλεσης, ώστε να

μπορεί να ειδοποιεί σωστά τον χρήστη για την ανάγκη επανασύνδεσης του δίσκου αλλάζοντας το layout. Επειτα θα συνδέσουμε την υπηρεσία mqttService με τη δραστηριότητα εκτελώντας την εντολή bindService, διότι η mqttService αποσυνδέεται όπως θα δούμε παρακάτω, κάθε φορά που η υπηρεσία περνά στο παρασκήνιο. Η εντολή bindService έχει ως συνέπεια να δημιουργείται ένα αντικείμενο mConnection, και να ξεκινά η εκτέλεση της μεθόδου onServiceConnected του αντικειμένου. Η μέθοδος onResume προσφέρεται καλύτερα από την onCreate για την εκκίνηση της υπηρεσίας mqttService, διότι μας δίνει την ευελιξία να δημιουργούμε ή να καταργούμε τη σύνδεση όταν η εφαρμογή περνά στο παρασκήνιο ή στο προσκήνιο αντίστοιχα, και επίσης εκτελείται και κατά τη δημιουργία της υπηρεσίας.
Στη συνέχεια στέλνουμε μήνυμα ON στο Broker με την publishMessage. Η υπηρεσία mqttService αναλαμβάνει να δημοσιεύσει το μήνυμα, και όταν αυτό πραγματοποιηθεί θα ενημερώσει τη τιμή της μεταβλητής mqtt_message_received από false που είναι αρχικά, σε true. Εάν δεν καταφέρουμε να δημοσιεύσουμε το μήνυμα (π.χ. διότι το δίκτυο είναι υπερφορτωμένο, ή διότι δεν έχει ξεκινήσει η onServiceConnected άμεσα, οπότε η mBound είναι ακόμη false) ,θα αποστέλλουμε το μήνυμα mqtt, μέχρις ότου η μεταβλητή πάρει την τιμή true, ή φτάσουμε στο ανώτατο κατώφλι των 120 προσπαθειών. Αυτό υλοποιείται με μία χρονοπρογραμματιζόμενη εργασία, με τη βοήθεια των κλάσεων Timer και TimerTask.
Για να αντιληφθεί ο Timer τις αλλαγές στη μεταβλητή mqtt_message_received, δημιουργούμε μια ιδιωτική μέθοδο:

```
private Boolean getmqttmessagestatus(){return mqtt_message_received;}
```

Επίσης, θα πρέπει να εισάγουμε κάποιες επιπλέον κλάσεις στον κώδικα:

```
import java.util.TimerTask;
import java.util.Timer;
```

Στο τμήμα δηλώσεων θα προσθέσουμε τις εξής μεταβλητές:

```
private static final int MAX_MQTT_RETRIES = 120;
private int mqtt_publish_retries;
```

Ο πλήρης κώδικας που πραγματοποιεί τα παραπάνω και προσθέτουμε στη μέθοδο onResume είναι ο παρακάτω:

```
Intent focus_change = new Intent(this, DiskUsageService.class);
focus_change.setAction("com.filedisplayactivity.has_focus");
focus_change.putExtra("focus", true);
if (!mBound) {
    bindService(mqttService, mConnection, Context.BIND_AUTO_CREATE);
}
publishMessage("ON"); //αποστολή μηνύματος στο σκληρό δίσκο
Log_OC.v(TAG, "OnResume published ON message, but did it arrive?");
mqtt_message_received = false;
mqtt_publish_retries = 0;
final boolean mqtt_publish_success = false;
final Timer timer = new Timer();
//Αποστολή μηνυμάτων με αρχική καθυστέρηση 1000ms, κάθε 500ms, μέχρις ότου συνδεθεί
//ο δίσκος.
timer.schedule(new TimerTask() {
    @Override
    public void run() {
        if (!getmqttmessagestatus()) {
            if (mqtt_publish_retries < MAX_MQTT_RETRIES) {
                publishMessage("ON");
```

```
                mqtt_publish_retries++;
            }
        } else {
            Log.v(TAG, "Succeeded by persisting " + mqtt_publish_retries + " times");
            timer.cancel();
        }
    }
}, 1000, 500);
```

Αλλαγές στη μέθοδο onPause

Η εκτέλεση της μεθόδου onPause, σημαίνει ότι η δραστηριότητα περνάει στο παρασκήνιο. Επομένως, είναι απαραίτητο να αντιστρέψουμε το αποτέλεσμα των εντολών που εκτελούνται στη μέθοδο onResume που είδαμε προηγουμένως. Αρχικά, θα αποστείλουμε μήνυμα OFF στον δίσκο διότι δεν υπάρχει ανάγκη να είναι συνδεδεμένος. Στη συνέχεια, θα αποσυνδέσουμε την mqttService από τη δραστηριότητα, διότι πλέον η σύνδεση δεν είναι αναγκαία, και θα πληροφορήσουμε την υπηρεσία DiskUsageService ότι η FileDisplayActivity δεν βρίσκεται πλέον στο προσκήνιο. Προσθέτουμε λοιπόν στην αρχή της μεθόδου τον παρακάτω κώδικα:

```
 publishMessage("OFF");    //αποσύνδεση του δίσκου
 if (mBound) {
     unbindService(mConnection); //αποσύνδεση της υπηρεσίας
     mBound = false;
 }
 Log_OC.v(TAG, "Off message from onPause - Notifying of change of visibility status");
 Intent focus_change = new Intent(this,DiskUsageService.class);
 focus_change.setAction("com.filedisplayactivity.has_focus");
 focus_change.putExtra("focus",false);
 broadcaster.sendBroadcast(focus_change); //πληροφόρηση ότι η FileDisplayActivity είναι
στο παρασκήνιο
```

6.4 Κλάση MQTTService.java

Η υπηρεσία MQTTService.java ,όπως είδαμε παραπάνω, προέρχεται από την εφαρμογή mqtt-client και υλοποιεί τις μεθόδους που είναι απαραίτητες για την επικοινωνία μέσω του πρωτοκόλλου MQTT. Η λειτουργικότητά της υπηρεσίας βασίζεται στη βιβλιοθήκη wmqtt.jar την οποία αντιγράφουμε σε φάκελο της επιλογής μας στο project, π.χ στο μονοπάτι owncloud-android-library/libs, και από το μενού δεξί-κλικ επιλέγουμε “Add As Library“. Για να λειτουργήσει η υπηρεσία mqttService, πρέπει να την δηλώσουμε στο αρχείο AndroidManifest.xml, προσθέτοντας τον παρακάτω κώδικα, στο εσωτερικό της ετικέτας <application> στο AndroidManifest.xml.

```
<service android:name=".services.MQTTService" />
```

Η υπηρεσία MQTTService δημιουργήθηκε αρχικά προκειμένου να αλληλεπιδρά με τη δραστηριότητα MQTTActivity.java, και να παρουσιάζει αναλυτικά τη κατάσταση της εφαρμογής. Επειδή αυτό όμως δεν είναι απαραίτητο για την εφαρμογή μας, θα αντικαταστήσουμε τη notifyUser με μια πιο απλή έκδοση που θα βασίζεται στα μηνύματα Toast. Στο τμήμα δηλώσεων θα

προσθέσουμε τη παρακάτω μεταβλητή:

```
Handler handler = new Handler();
```

Στο τμήμα κώδικα αντικαθιστούμε τη notifyUser με την παρακάτω:

```
private void notifyUser(String alert, String title, String body)
{
     if (body.equals("ON")) body = "Hard disk has been mounted";
     else if (body.equals("OFF")) body = "Hard disk has been unmounted";
     final String parent_alert = body;
     handler.post(new Runnable() {
              public void run() {

Toast.makeText(getApplicationContext(),parent_alert,Toast.LENGTH_SHORT).show();
              }
     });
}
```

Θα χρησιμοποιήσουμε τη παραπάνω μέθοδο για να ειδοποιήσουμε το χρήστη για τη σύνδεση και την αποσύνδεση του δίσκου, και επίσης για να παροτρύνουμε το χρήστη να ρυθμίσει τη σύνδεση του προς το διαδίκτυο και τον MQTT Broker. Το πρώτο, πραγματοποιείται κατά τη κλήση της notifyUser από τη μέθοδο publishArrived της MQTTService, η οποία λαμβάνει όσα μηνύματα δημοσιεύονται στο θέμα που στέλνει η εφαρμογή μας. Στη περίπτωση αυτή, η notifyUser τυπώνει κατάλληλα μηνύματα όταν λάβει ON ή OFF, και σε κάθε άλλη περίπτωση τυπώνει το μήνυμα αυτούσιο. Το δεύτερο ζητούμενο πραγματοποείται μέσα στη μέθοδο connectToBroker όταν, σε περίπτωση αποτυχίας, εκτελούμε την εξής εντολή:

```
notifyUser("Unable to connect", "MQTT", "Unable to connect - will retry later. Please
ensure that you have inserted a valid MQTT broker IP address and a topic");
```

Η αρχική εφαρμογή MQTT, γράφτηκε με τέτοιο τρόπο που ο χρήστης μπορούσε να γνωρίζει αν η υπηρεσία είναι ενεργοποιημένη ανά πάσα στιγμή. Κάτι τέτοιο δεν είναι απαραίτητο για την εφαρμογή, επομένως διαγράφουμε τον παρακάτω κώδικα της μεθόδου handleStart:

```
NotificationManager nm = (NotificationManager) getSystemService(NOTIFICATION_SERVICE);
Notification notification = new Notification(R.drawable.ic_mqtt,
                                             "MQTT",
                                             System.currentTimeMillis());
notification.flags |= Notification.FLAG_ONGOING_EVENT;
notification.flags |= Notification.FLAG_NO_CLEAR;
Intent notificationIntent = new Intent(this, MQTTActivity.class);
PendingIntent contentIntent = PendingIntent.getActivity(this, 0,
                                                         notificationIntent,

PendingIntent.FLAG_UPDATE_CURRENT);
notification.setLatestEventInfo(this, "MQTT", "MQTT Service is running", contentIntent);
nm.notify(MQTT_NOTIFICATION_ONGOING, notification);
```

Η υπηρεσία MQTTService επικοινωνεί ακόμη με τις δραστηριότητες FileDisplayActivity και Preferences, και με την υπηρεσία ConnectivityActionReceiver.
Από τη δραστηριότητα Preferences λαμβάνει μήνυμα όταν αλλάξει η τιμή των πεδίων mBrokerAddress και mTopic. Τότε, θέτουμε την μεταβλητή RESTART σε true, και καλούμε διαδοχικά τις stopself και startservice (καλείται από την onDestroy) ώστε η υπηρεσία να

επανεκκινήσει με ανανεωμένα δεδομένα και μια νέα σύνδεση.
Από την υπηρεσία ConnectivityActionReceiver λαμβάνει μήνυμα όταν ανιχνευτεί σύνδεση στο Internet από κατάσταση μη σύνδεσης, και εκκινεί τις διαδικασίες που απαιτούνται για τη δημιουργία σύνδεσης με τον MQTT Broker. Στη δραστηριότητα FileDisplayActivity στέλνει μήνυμα η υπηρεσία ώστε να την ειδοποιήσει για την λήψη ενός νέου μηνύματος. Χρησιμοποιώντας το περιεχόμενο του μηνύματος, η FileDisplayActivity μπορεί να επιβεβαιώσει ότι τα μηνύματα ON και OFF που έχουν αποσταλεί έχουν φτάσει στον προορισμό τους.
Παράλληλα, στη περίπτωση που δεν έχει εξασφαλιστεί σύνδεση με τον Broker, η υπηρεσία φροντίζει να αποθηκευτεί στη μεταβλητή pending_message, και μόλις επιτευχθεί σύνδεση το στέλνει με τη μέθοδο send_pending_messages.

Για να πετύχουμε τα παραπάνω, πρέπει να εισάγουμε τις εξής επιπλέον κλάσεις:

```
import java.util.Timer;
import java.util.TimerTask;
import android.preference.PreferenceManager;
```

Στο τμήμα δηλώσεων να προσθέσουμε τις εξής μεταβλητές:

```
private BroadcastReceiver receiver;
private String pending_message = "none";
private Boolean RESTART = false;
```

Στη συνέχεια, θα ορίσουμε τη μέθοδο send_pending_messages:

```
private void send_pending_messages(){
    if (!pending_message.equals("none")) {
        Log.v(TAG,"Just Sent a pending message, "+pending_message);
        publishMessageToTopic(pending_message);
        pending_message="none";
    }
}
```

Στη μέθοδο onCreate, προσθέτουμε τον παρακάτω κώδικα:

```
        Log.v(TAG, "Current broker address is: " + brokerHostName);
        Log.v(TAG, "Current Mqtt topic is: " + topicName);
//Παρακάτω παίρνουμε τη διεύθυνση του Broker και το θέμα από τις ρυθμίσεις.
        PreferenceManager preferenceManager ;
        SharedPreferences prefs =
PreferenceManager.getDefaultSharedPreferences(getApplicationContext());
        String updatedBrokerHostName=prefs.getString("brokerHostName",brokerHostName);
        String updatedTopicName = prefs.getString("topicName",topicName);
        if (updatedBrokerHostName!=null && !
updatedBrokerHostName.equals(brokerHostName)) {
            Log.v(TAG, "The updated broker address is: " + updatedBrokerHostName);
            brokerHostName = updatedBrokerHostName;
        }
        if (updatedTopicName!=null && !updatedTopicName.equals(topicName)) {
            Log.v(TAG, "The updated Mqtt topic is: " + updatedTopicName);
            topicName = updatedTopicName;
        }
        receiver = new BroadcastReceiver() {
            @Override
```

```
            public void onReceive(Context context, Intent intent) {
                if (intent.getAction().equals("com.owncloud.preferences.update_mqtt"))
{ //Ανανεώθηκε η διεύθυνση του Broker ή το θέμα.
                        RESTART = true;
                        stopSelf();
                } else if (intent.getAction().equals("com.mqttservice.restart")) {
//Συνδεθήκαμε στο δίκτυο, και πρέπει να δοκιμάσουμε να συνδεθούμε ξανά.
                    Log.d(TAG, "Received Alert to subscribe once more");
                    Log.v(TAG, "these are the current values for broker: " +
brokerHostName + " and topic: " + topicName);
                    Timer timer = new Timer();
                    timer.schedule(new TimerTask() {
                        @Override
                        public void run() {
//Προσπαθούμε να δημιουργήσουμε σύνδεση με τις καινούργιες παραμέτρους, και να
αποστείλουμε τυχόν εκκρεμή μηνύματα.
                            Log.v(TAG, "Trying to connect with the help of Timer to
topic " + topicName + " and broker " + brokerHostName);
                            if (mqttClient == null) {
                                defineConnectionToBroker(brokerHostName);
                            }
                            if (connectToBroker()) {
                                subscribeToTopic(topicName);
                                send_pending_messages();
                                Log.v(TAG, "Connected with the help of Timer method
and sent pending messages");
                                cancel(); //Ακυρώνουμε τη χρονοπρογραμματισμένη
εργασία διότι στάλθηκαν τα εκκρεμή μηνύματα.
                            }
                        }
                    }, 10, 200);
                }
            }
        };
```

Μετά τις παραπάνω προσθήκες, η φόρτωση των παραμέτρων σύνδεσης δε χρειάζεται να επαναληφθεί, και για το λόγο αυτό διαγράφουμε τις εξής γραμμές από τον κώδικα:

```
SharedPreferences settings = getSharedPreferences(APP_ID, MODE_PRIVATE);
brokerHostName = settings.getString("broker", "");
topicName      = settings.getString("topic",  "");
```

Στη συνέχεια, εισάγουμε τη κλάση android.support.v4.content.LocalBroadcastManager και προσθέτουμε στην αρχή της μεθόδου onStartCommand τη δήλωση του λήπτη που χρησιμοποιήθηκε παραπάνω, μαζί με τα επιτρεπτά μηνύματα που μπορεί να λάβει:

```
import android.support.v4.content.LocalBroadcastManager;

IntentFilter allowed_intents = new IntentFilter("com.owncloud.preferences.update_mqtt");
allowed_intents.addAction("com.mqttservice.restart");
LocalBroadcastManager.getInstance(this).registerReceiver((receiver),
        new IntentFilter(allowed_intents));
```

Στη μέθοδο onDestroy, θα πρέπει να αναιρέσουμε τη δήλωση του λήπτη, προσθέτοντας στην αρχή της την παρακάτω εντολή:

```
LocalBroadcastManager.getInstance(this).unregisterReceiver(receiver);
```

Επίσης, στο τέλος της μεθόδου διαχειριζόμαστε το ενδεχόμενο να πρέπει να επανεκκινήσει η υπηρεσία για τους λόγους που περιγράφηκαν πιο πάνω, με τον εξής κώδικα:

```
if(RESTART){
     startService(new Intent(this,MQTTService.class));
     Log.v(TAG,"Mqtt service should restart");
}
```

Προκειμένου να μπορούμε να επικοινωνήσουμε με τη δραστηριότητα FileDisplayActivity, και να της στείλουμε το μήνυμα που λάβαμε για επιβεβαίωση, θα πρέπει να προσθέσουμε τις παρακάτω γραμμές στο τέλος της μεθόδου publishArrived:

```
Intent reception_of_message = new Intent(MQTTService.this, FileDisplayActivity.class);
reception_of_message.setAction("com.mqttservice.message_received");
reception_of_message.putExtra("message_received",messageBody);
LocalBroadcastManager.getInstance(this).sendBroadcast(reception_of_message);
```

Στη περίπτωση που η υπηρεσία δεν μπορέσει να αποστείλει κάποιο μήνυμα, όταν δεν υπάρχει σύνδεση με τον Broker, θα ενημερώσουμε τη τιμή του εκκρεμούς μηνήματος, στο εσωτερικό του ελέγχου `if (isAlreadyConnected() == false)` στη μέθοδο publishMessageToTopic:

```
if (isAlreadyConnected() == false)
{
   pending_message = message;
   Log.d(TAG, "mqtt, Unable to publish as we are not connected");
}
```

6.5 Κλάση ConnectivityActionReceiver.java

Η κλάση ConnectivityActionReceiver που επεκτείνει την BroadcastReceiver είναι υπεύθυνη για τη διαχείριση γεγονότων σχετικών με τη συνδεσιμότητα της εφαρμογής. Βρίσκεται στο μονοπάτι src/com/owncloud/android/files/services και εκκινείται από το αρχείο AndroidManifest.xml, με ειδική δήλωση τύπου receiver. Όταν από κατάσταση μη σύνδεσης παρατηρήθει σύνδεση στο Internet μέσω wifi, επανεκκινείται η υπηρεσία mqtt, και η κλάση αποστέλλει σχετικό μήνυμα το οποίο διαχειρίζεται η υπηρεσία MQTT. Στην υπηρεσία ConnectivityActionReceiver, εισάγουμε τις εξής επιπλέον κλάσεις:

```
import android.support.v4.content.LocalBroadcastManager;
import com.owncloud.android.services.MQTTService;
import android.util.Log;
```

Στο τμήμα δηλώσεων προσθέτουμε τις ακόλουθες μεταβλητές:

```
LocalBroadcastManager broadcaster;
Intent mqttservice;
```

Στην αρχή του κώδικα της μεθόδου wifiConnected θα πρέπει να προσθέσουμε τα εξής:

```
Log.v(TAG,"wifi connected, so mqtt is restarted");
mqttservice = new Intent(context.getApplicationContext(),MQTTService.class);
```

```
mqttservice.setAction("com.mqttservice.restart");
broadcaster = LocalBroadcastManager.getInstance(context);
broadcaster.sendBroadcast(mqttservice);
```

6.6 Κλάση DiskUsageService.java

Η υπηρεσία DiskUsageService λειτουργεί επικουρικά για τη δραστηριότητα FileDisplayActivity, εξασφαλίζωντας ότι η πρόσβαση στο δίσκο θα διακοπεί όταν παρέλθει κάποιο χρονικό διάστημα χωρίς δραστηριότητα. Όπως είδαμε παραπάνω, στη μέθοδο onResume η δραστηριότητα FileDisplayActivity ενημερώνει την υπηρεσία ότι βρίσκεται στο προσκήνιο, και μπορεί να δείχνει ειδοποιήσεις στο χρήστη. Ως αποτέλεσμα, η υπηρεσία θέτει τη μεταβλητή FILE_DISPLAY_ACTIVE στη τιμή true. Μόλις λοιπόν περάσουν λοιπόν περισσότερα από IDLE_SECONDS_LIMIT δευτερόλεπτα, υπολογισμένα με ακρίβεια +/- PRECISION_SECONDS δευτερόλεπτα, η υπηρεσία στέλνει ειδικό μήνυμα στη FileDisplayActivity, ενημερώνοντάς την για την ανάγκη αποσύνδεσης του δίσκου. Αμέσως μετά η FileDisplayActivity πρέπει να απαντήσει ότι το έλαβε και να δείξει τη σχετική οθόνη που παρουσιάσαμε παραπάνω. Αυτό χρειάζεται να γίνει μόνο μια φορά, και το επιτυγχάνουμε με τη βοήθεια της μεταβλητής USER_NOTIFIED.
Στη περίπτωση όμως που ο χρήστης παρακολουθεί κάποια άλλη δραστηριότητα της εφαρμογής, όπως την Preferences ή δεν χρησιμοποιεί την εφαρμογή, στέλνεται ειδικό μήνυμα από τη μέθοδο onPause που θέτει την FILE_DISPLAY_ACTIVE σε false και εμποδίζει την αποστολή επιπλέον μηνυμάτων. Αυτό συμβαίνει γιατί και στις δύο περιπτώσεις, όταν ο χρήστης θελήσει να δει τα αρχεία του, θα κληθεί η μέθοδος onResume της FileDisplayActivity, και θα αποσταλεί ούτως ή άλλως μήνυμα σύνδεσης στο δίσκο.
Αρχικά, για τη χρήση της υπηρεσίας θα πρέπει να τη δηλώσουμε στο αρχείο AndroidManifest.xml, στο εσωτερικό της ετικέτας <application> προσθέτοντας τη παρακάτω γραμμή:

```
<service android:name=".services.DiskUsageService" />
```

Επίσης, για να επιτύχουμε τη λήψη των μηνυμάτων όπως αναφέραμε παραπάνω, θα χρησιμοποιήσουμε ένα λήπτη, τον οποίο θα δηλώσουμε στη μέθοδο onStartCommand και θα αναιρέσουμε τη δήλωσή του στη μέθοδο onDestroy. Μαζί με τη δήλωσή του, θα πρέπει να δηλώσουμε και ποιά intents θα αποδέχεται.
Ο πλήρης κώδικας της υπηρεσίας είναι ο παρακάτω:

```
package com.owncloud.android.services;

import android.app.Service;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
import android.os.IBinder;
import android.support.annotation.Nullable;
import android.support.v4.content.LocalBroadcastManager;
import android.util.Log;
```

```
import com.owncloud.android.lib.common.utils.Log_OC;
import com.owncloud.android.ui.activity.FileDisplayActivity;
import java.text.SimpleDateFormat;
import java.util.Calendar;
import java.util.Timer;
import java.util.TimerTask;
import java.util.concurrent.TimeUnit;




public class DiskUsageService  extends Service {
    private static final int IDLE_SECONDS_LIMIT = 300; // μέγιστος επιτρεπτός χρόνος
αδράνειας της εφαρμογής
    private static final int PRECISION_SECONDS = 10; //μέγιστη επιτρεπτή απόκλιση από το
πάνω όριο
    private static boolean USER_NOTIFIED = false;
    private static boolean FILE_DISPLAY_ACTIVE = true; //είναι η προβολή αρχείων ενεργή;
    private BroadcastReceiver receiver;
    private static final String TAG = DiskUsageService.class.getSimpleName();
    private LocalBroadcastManager broadcaster;
    @Nullable
    @Override
    public IBinder onBind(Intent intent) {
        return null;
    }
    public void onCreate()
    {

        broadcaster = LocalBroadcastManager.getInstance(this);
        final SimpleDateFormat format = new SimpleDateFormat("yyyy/MM/dd HH:mm:ss");
        receiver = new BroadcastReceiver() {
            @Override
            public void onReceive(Context context, Intent intent) {
                if
(intent.getAction().equals("com.filedisplayactivity.user_notified_reset")) {
                    Log.v(TAG,"User responded with remounting");
                    USER_NOTIFIED = false; //αρχικοποίηση της μεταβλητής, ο χρήστης
απάντησε με επανασύνδεση του δίσκου
                }else
if(intent.getAction().equals("com.filedisplayactivity.broadcast_received")){
                    Log.v(TAG, "User was notified on his screen");
                    USER_NOTIFIED = true; //ο χρήστης ενημερώθηκε με κατάλληλη οθόνη
για την αποσύνδεση του δίσκου και τη δυνατότητα επανασύνδεσης
                }else
if(intent.getAction().equals("com.filedisplayactivity.has_focus")){
                    Log.v(TAG,"FileDisplayActivity has just either appeared or
disappeared");
                    FILE_DISPLAY_ACTIVE = intent.getBooleanExtra("focus", false);// Η
FileDisplayActivity ενημερώνει ότι πέρασε στο προσκήνιο ή στο παρασκήνιο αντίστοιχα.
                }
            }
        };

// Υπολογισμός του χρόνου μεταξύ της τελευταίας δραστηριότητας (όλες οι δραστηριότητες
λήψης ή ανεβάσματος αρχείων καταγράφονται) και της παρούσης στιγμής σε τακτική βάση.
        Timer timer =new Timer();
        timer.schedule(new TimerTask(){
        @Override
            public void run() {
            long time_idle=0; // πραγματικός χρόνος αδράνειας της εφαρμογής
                try {
```

```
                    if (Log_OC.getLast_action_timestamp() != null) {
                        time_idle = Calendar.getInstance().getTime().getTime() -
format.parse(Log_OC.getLast_action_timestamp()).getTime(); // υπολογισμός πραγματικού
χρόνου αδράνειας
                    } else {
                        Log.d(TAG,"Last Action time cannot be determined");
                    }
                }
                catch (Exception e){
                    Log.e(TAG,"Could not calculate time difference");
                }
                long time_idle_in_seconds = TimeUnit.MILLISECONDS.toSeconds(time_idle);
                if (time_idle_in_seconds>IDLE_SECONDS_LIMIT){
                    if(!USER_NOTIFIED && FILE_DISPLAY_ACTIVE){
                        notifyDiskDisconnected(); //Ενημερώνουμε τον χρήστη μόνο αν
έχουν περάσει περισσότερα δευτερόλεπτα από το όριο, και δεν τον έχουμε ενημερώσει, και
έχουμε ανοικτή την προβολή αρχείων (η FileDisplayActivity είναι ενεργή) .
                    }
                }
            }
        },10,PRECISION_SECONDS*1000);
    }
    public void notifyDiskDisconnected() {
// Πληροφόρηση της FileDisplayActivity για την έλλειψη δραστηριότητας.
        final Intent intent = new
Intent(DiskUsageService.this,FileDisplayActivity.class);
        intent.setAction("com.diskusageservice.remount_action");
        broadcaster.sendBroadcast(intent);
        Log.v(TAG,"notifyDiskDisconnected has sent broadcast");
    }
    @Override
    public int onStartCommand(final Intent intent, int flags, final int startId){
        IntentFilter filter = new
IntentFilter("com.filedisplayactivity.user_notified_reset");
        filter.addAction("com.filedisplayactivity.broadcast_received");
        filter.addAction("com.filedisplayactivity.has_focus"); //Αποδεκτά intents
        LocalBroadcastManager.getInstance(this).registerReceiver((receiver),filter);
        return  START_STICKY; //Επανέναρξη της υπηρεσίας εφόσον τερματιστεί σε
κατάσταση μεγάλης έλλειψης κύριας μνήμης
    }
    @Override
    public void onDestroy(){
        LocalBroadcastManager.getInstance(this).unregisterReceiver(receiver); //
αναίρεση δήλωσης του λήπτη κατά τον τερματισμό.
    }
}
```

Για να έχουμε πρόσβαση στη τελευταία καταγεγραμμένη δραστηριότητα του χρήστη, θα πρέπει να προσθέσουμε κάποιον κώδικα στην κλάση Log_OC.java που βρίσκεται στο μονοπάτι src/com/owncloud/android/lib/common/utils. Στο τμήμα δηλώσεων ορίζουμε την εξής μεταβλητή:

```
private static String last_action_timestamp;
```

Στη συνέχεια, στη μέθοδο appendLog, αμέσως μετά την εντολή

```
 String timeStamp = new
SimpleDateFormat(SIMPLE_DATE_FORMAT).format(Calendar.getInstance().getTime());
```

προσθέτουμε την ανάθεση:

```
last_action_timestamp = timeStamp;
```

Τη τιμή της μεταβλητής μπορούμε να την πάρουμε με την μέθοδο getLast_action_timestamp, που ορίζεται ως εξής:

```
public static String getLast_action_timestamp(){return last_action_timestamp; }
```

6.7 Κλάση Preferences.java

Η δραστηριότητα Preferences βρίσκεται στο μονοπάτι src/com/owncloud/android/ui/activity και διαχειρίζεται τις ρυθμίσεις του χρήστη. Η λειτουργία της βασίζεται στο γραφικό περιβάλλον που περιγράφεται στο αρχείο preferences.xml, στο μονοπάτι android/res/xml/preferences.xml. Για τους σκοπούς της εφαρμογής μας, η δραστηριότητα θα πρέπει να διαχειρίζεται τη δυνατότητα επιλογής της διεύθυνσης του Broker και του θέματος που θα παρακολουθεί. Το γραφικό σκέλος αυτής της απαίτησης μπορούμε να το ικανοποιήσουμε προσθέτοντας μια επιπλέον κατηγορία προτιμήσεων στο αρχείο preferences.xml, αμέσως μετά την κατηγορία accounts:

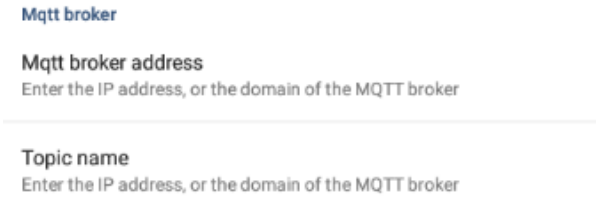
```
<PreferenceCategory android:title="@string/prefs_category_mqtt"
android:key="mqtt_category">
    <com.owncloud.android.utils.IPAddressPreference
        android:key="broker"
                    android:title="@string/broker_address"
                    android:summary="@string/broker_address_summary"
                    android:defaultValue="192.168.1.6" <!-- Διεύθυνση του εσωτερικού
υποδικτύου που μπορούμε να αλλάξουμε όπως επιθυμούμε -->
                    android:inputType="text"/>
    <EditTextPreference android:key="topic"
                    android:title="@string/topic_name"
                    android:summary="@string/broker_address_summary"
                    android:defaultValue="SYS/disk"
                    android:inputType="text" />
</PreferenceCategory>
```

Στο αρχείο strings.xml προσθέτουμε τις παρακάτω τιμές:

```
<string name="broker_address">Mqtt broker address</string>
<string name="broker_address_summary">Enter the IP address, or the domain of the MQTT
broker</string>
<string name="topic_name">Topic name</string>
<string name="topic_name_summary">MQTT topic that Owncloud will use</string>
<string name="prefs_category_mqtt">Mqtt broker</string>
```

Η εισαγωγή των παραπάνω εντολών έχει ως αποτέλεσμα την εμφάνιση των παρακάτω πεδίων στο

μενού της δραστηριότητας Preferences:



Στο προγραμματιστικό σκέλος πρέπει να διαβάζουμε τις τιμές που εισάγονται στα παραπάνω πεδία, και μόλις αντιλαμβανόμαστε κάποια αλλαγή σε ένα από αυτά, να στέλνουμε ένα μήνυμα στην υπηρεσία mqttService με τη χρήση ενός LocalBroadcastManager, που θα περιλαμβάνει τις επικαιροποιημένες τιμές.
Επομένως στο τμήμα δηλώσεων θα προσθέσουμε τα πεδία mPrefBroker και mPrefTopic – τα οποία θα έχουν τις τιμές που εισάγει ο χρήστης στο γραφικό περιβάλλον – και τα πεδία mBrokerAddress και mTopic που θα αποθηκεύουν τις μόνιμες τιμές της διεύθυνσης του Broker και του θέματος. Επειδή στην περίπτωση του mPrefBroker πρέπει να δεχόμαστε είσοδο με μορφή διευθύνσεων IP είναι αναγκαίο να αντιγράψουμε από τον κώδικα της εφαρμογής MQTT τη κλάση IPAddressPreference.java στο μονοπάτι android/src/com/owncloud/android/lib/common/utils.

Για την υλοποίηση όσων αναφέραμε, είναι απαραίτητο να εισάγουμε τις παρακάτω κλάσεις:

```
import android.support.v4.content.LocalBroadcastManager;
import android.preference.EditTextPreference;
import com.owncloud.android.utils.IPAddressPreference;
import android.util.Log;
import com.owncloud.android.services.MQTTService;
import android.app.Activity;
```

Προσθέτουμε τις εξής μεταβλητές στο τμήμα δηλώσεων:

```
private IPAddressPreference mPrefBroker;
private EditTextPreference mPrefTopic;
private String mBrokerAddress;
private String mTopic;
```

Στο τμήμα του κώδικα, στη μέθοδο onCreate, αμέσως μετά το κομμάτι κώδικα

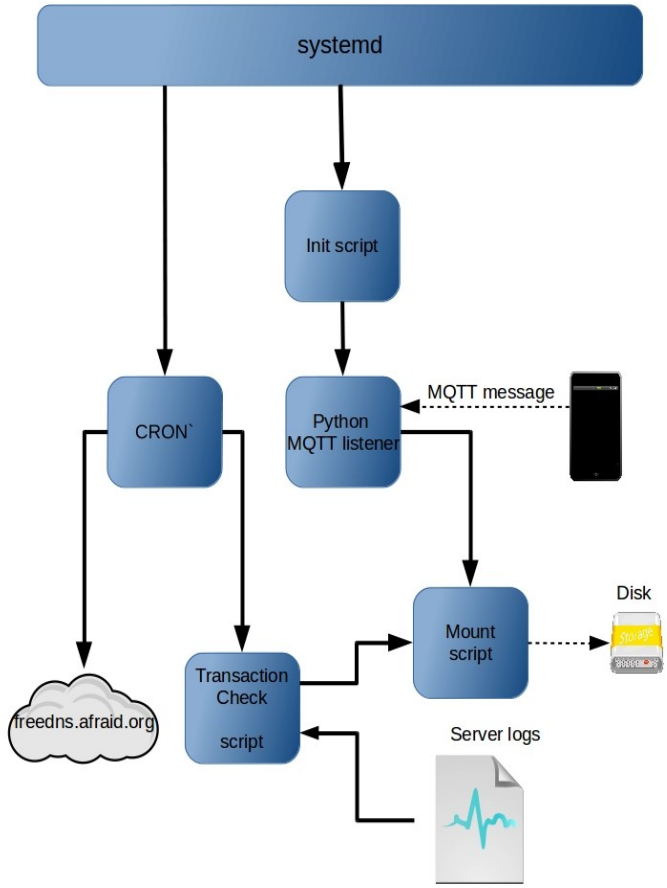
`if (mPrefInstantUploadPath != null){…}`, προσθέτουμε τις παρακάτω εντολές:

```
        final Activity current_preference_activity = this;

        mPrefBroker = (IPAddressPreference) findPreference("broker");
        mPrefBroker.setOnPreferenceChangeListener(new OnPreferenceChangeListener() {
            @Override
            public boolean onPreferenceChange(Preference preference,Object object) {
                mBrokerAddress = mPrefBroker.getEditText().getText().toString();
                mPrefBroker.setText(mBrokerAddress);
                SharedPreferences preferences =
PreferenceManager.getDefaultSharedPreferences(getApplicationContext());
                SharedPreferences.Editor editor = preferences.edit();
                editor.putString("brokerHostName", mBrokerAddress); // αποθήκευση της
αλλαγμένης διεύθυνσης του Broker.
                editor.commit();
```

```
                Intent intent = new Intent (Preferences.this, MQTTService.class);
                intent.setAction("com.owncloud.preferences.update_mqtt");
                intent.putExtra("brokerHostName",mBrokerAddress);
                intent.putExtra("topicName",mTopic);
                 LocalBroadcastManager.getInstance(current_preference_activity).
sendBroadcast(intent);

                Log.v(TAG,"Just sent the new broker address "+mBrokerAddress);
                return false;
            }
        });

        mPrefTopic = (EditTextPreference) findPreference("topic");
        mPrefTopic.setOnPreferenceChangeListener(new OnPreferenceChangeListener() {
            @Override
            public boolean onPreferenceChange(Preference preference,Object object) {
                //mPrefBroker.getSharedPreferences().getString(mBrokerAddress,null);
                mTopic = mPrefTopic.getEditText().getText().toString();
                mPrefTopic.setText(mTopic);
                SharedPreferences preferences =
PreferenceManager.getDefaultSharedPreferences(getApplicationContext());
                SharedPreferences.Editor editor = preferences.edit();
                editor.putString("topicName", mTopic); // Αποθήκευση του ανανεωμένου
θέματος MQTT
                editor.commit();
                Intent intent = new Intent (Preferences.this, MQTTService.class);
                intent.setAction("com.owncloud.preferences.update_mqtt");
                intent.putExtra("brokerHostName",mBrokerAddress);
                intent.putExtra("topicName",mTopic);
                broadcaster.sendBroadcast(intent);
                Log.v(TAG,"Just sent the topic, which is "+mTopic);
                return false;
            }
        });
```

7 Αλλαγές στο server και προγραμματισμός ασφαλών υπηρεσιών

Για να μπορεί ο εξυπηρετητής να αποκρίνεται στα μηνύματα MQTT που αποστέλλει ο client, αλλά και να αποσυνδέει το δίσκο αυτόματα μετά τη παρέλευση μεγάλου χρονικού διαστήματος, θα πρέπει να δημιουργήσουμε σχετικές υπηρεσίες στον εξυπηρετητή. Ένα γενικό διάγραμμα της ροής εκτέλεσης των υπηρεσιών που υπάρχουν στην υλοποίησή μας είναι το παρακάτω:



Η Systemd είναι η αρχική διαδικασία που τρέχει στη διανομή Linux κατά την εκκίνηση του εξυπηρετητή και αναλαμβάνει να εκκινήσει όλες τις διαδικασίες. Μεταξύ των άλλων, εκκινείται το init script που με τη σειρά του εκκινεί τον MQTT Listener, και η υπηρεσία Cron. Όλα τα προγράμματα με την εξαίρεση του MQTT Listener τρέχουν στο φλοιό bash, αλλά ο MQTT Listener είναι προγραμματισμένος σε python, και για το λόγο αυτό πρέπει να ξεκινήσει από μια υπηρεσία εκκίνησης (init script). Η εκτέλεση των ελέγχων για τη χρήση του αποθηκευτικού μέσου σε τακτικά διαστήματα βασίζεται στην υπηρεσία Cron, όπως και η εκτέλεση της αντιστοίχησης μεταξύ της δυναμικής IP και του Domain Name που είδαμε παραπάνω.
Παρακάτω θα υλοποιήσουμε κάθε πρόγραμμα (script) που αναφέρουμε παραπάνω, και τον MQTT Listener.

7.1 Υλοποίηση του MQTT Listener και εγκατάσταση του MQTT Broker

Επειδή όπως αναφέραμε παραπάνω το πρόγραμμα python δεν μπορεί να ξεκινήσει απευθείας από την Systemd, θα πρέπει να δημιουργήσουμε αρχικά την υπηρεσία εκκίνησης (Init script) η οποία θα φροντίσει να το ξεκινήσει. Γράφουμε λοιπόν από ένα νέο τερματικό τις παρακάτω εντολές:

cd /etc/init.d
nano python_mqtt_listener.sh #δημιουργία του αρχείου της υπηρεσίας εκκίνησης και άνοιγμα για επεξεργασία

Στη συνέχεια προσθέτουμε στο αρχείο τις παρακάτω γραμμές:

```
#!/bin/bash
date > /home/pi/Desktop/newsfile
python -u /home/pi/Desktop/mqtt_listener.py >> /home/pi/Desktop/newsfile
#Θέλουμε η έξοδος να μη λειτουργεί με buffering (παράμετρος u) και έτσι να
γίνεται διαθέσιμη αμέσως.
```

Κλείνουμε το αρχείο και γράφουμε τις εξής εντολές:

sudo chmod +x python_mqtt_listener.sh #μετατρέπουμε το αρχείο σε εκτελέσιμο
sudo update-rc.d python_mqtt_listener.sh defaults #πληροφόρηση λειτουργικού συστήματος ότι η υπηρεσία πρέπει να ξεκινά στην εκκίνηση.
cd ~/Desktop #δημιουργούμε το αρχείο που δηλώσαμε στο init script. Μπορούμε να το κάνουμε σε όποιο φάκελο θέλουμε, με κατάλληλη τροποποίηση του αρχείου /etc/init.d/python_mqtt_listener.sh

Ακολούθως, θα εγκαταστήσουμε το πρόγραμμα εγκατάστασης πακέτων της Python, και τις βιβλιοθήκες που είναι απαραίτητες για την επικοινωνία μέσω του MQTT:

sudo apt-get update && sudo apt-get install python-pip
sudo pip install paho-mqtt
nano mqtt_listener.py #δημιουργία του αρχείου του προγράμματος python και άνοιγμα για επεξεργασία

Το πρόγραμμα mqtt_listener.py που θα δημιουργήσουμε στο μονοπάτι /home/pi/Desktop/mqtt_listener.py, θα πρέπει να συνδέει και να αποσυνδέει το δίσκο ανάλογα με τα μηνύματα που αυτός λαμβάνει. Επίσης, θα διαχειρίζεται την περίπτωση που δεν είναι δυνατόν να επικοινωνήσει με τον MQTT server, ξαναπροσπαθώντας.
Το περιεχόμενο του αρχείου παρατίθεται παρακάτω:

```
import paho.mqtt.client as mqtt
import subprocess
import time
```

```
import datetime

external_address = "owncloud.home.kg"
internal_address = "192.168.1.6"

#FLAG
mqtt_address = internal_address

# Καλείται όταν αναγνωριστεί η σύνδεσή μας από τον server
def on_connect(client, userdata, rc):
    print("Connected with result code "+str(rc))
    # Εάν χαθεί η σύνδεση και αναγκαστούμε να επανασυνδεθούμε, χάρη στην
    # on_connect οι συνδρομές θα ανανεωθούν.
    client.subscribe("SYS/disk")


# Καλείται όταν αποσταλεί κάποιο νέο μήνυμα από τον server
def on_message(client, userdata, msg):
    print(msg.topic+" "+str(msg.payload))
    if (msg.payload=="ON"):
    subprocess.call(["/home/pi/Desktop/mountscript.sh
"+str(msg.payload)],shell=True)

    print("mount script contacted")

#Κύριο μέρος του προγράμματος, προσπαθούμε να δημιουργήσουμε ένα mqtt-client.
while True:
        try:
         client = mqtt.Client()
         client.on_connect = on_connect
         client.on_message = on_message
         client.connect(mqtt_address, 1883, 60)
         # Μόλις καλείται η παρακάτω εντολή ο client δρα μόνο προκειμένου να
         #απαντήσει στα μηνύματα που τον αφορούν.
         client.loop_forever()
        except Exception, e:
        #Χειρισμός της εξαίρεσης, στη περίπτωση που υπάρξει πρόβλημα κατά την
        #έναρξη
           print "Exception handled, reconnecting...\nDetail:\n%s" % e
           time.sleep(5)
```

Στη συνέχεια, στο μηχάνημα που θα φιλοξενεί τον mqtt-broker (στο παράδειγμά μας είναι ο ίδιος ο εξυπηρετητής), θα εγκαταστήσουμε τον Mosquitto broker, δίνοντας τη παρακάτω εντολή:

sudo apt-get install mosquitto mosquitto-clients

Ο Broker ξεκινά τη λειτουργία του αμέσως μετά την εγκατάσταση, χωρίς την ανάγκη να παρέμβουμε.

7.2 Πρόγραμμα σύνδεσης και αποσύνδεσης του δίσκου

Το επόμενο βήμα που θα ακολουθήσουμε για να λειτουργήσει η διάταξη, είναι να δημιουργήσουμε το αρχείο mountscript.sh – το οποίο μας επιτρέπει μια αφαίρεση της διαδικασίας σύνδεσης και αποσύνδεσης του δίσκου – στο μονοπάτι ~/Desktop. Εκτελούμε λοιπόν τις εξής εντολές σε ένα τερματικό:

cd ~/Desktop
nano mountscript.sh #δημιουργία του αρχείου του προγράμματος και άνοιγμα για επεξεργασία

Τα περιεχόμενα του αρχείου θα είναι τα παρακάτω:

```
#!/bin/bash
#Το πρόγραμμα πραγματοποιεί σύνδεση του δίσκου όταν εκτελείται με όρισμα ON, και
#αποσύνδεση του δίσκου όταν εκτελείται με όρισμα OFF, τυπώνοντας κατάλληλα
#διαγνωστικά μηνύματα στο αρχείο newsfile.

disk="/dev/sda2"

if [ "$1" == "ON" ];
then
    echo "mounting" >> /home/pi/Desktop/newsfile
    mount $disk  >> /home/pi/Desktop/newsfile
else
    if [ "$1" == "OFF" ];
    then
        echo "unmounting" >> /home/pi/Desktop/newsfile
        umount $disk
    else
    echo "Random message arrived"  >> /home/pi/Desktop/newsfile
    echo "$2"
    fi
fi
```

Στη συνέχεια κάνουμε το παραπάνω αρχείο εκτελέσιμο:

chmod +x mountscript.sh

7.3 Πρόγραμμα ελέγχου αδράνειας δίσκου

Όπως είδαμε παραπάνω, στον Android Client υπάρχουν διαδικασίες που προβλέπουν την αποσύνδεση του δίσκου όταν η εφαρμογή δεν χρησιμοποιηθεί για κάποιο χρονικό διάστημα από τον χρήστη. Αυτό είναι αρκετό, στη περίπτωση που έχουμε μια ιδανική σύνδεση δικτύου. Όμως, αν για κάποιο λόγο η σύνδεση χαθεί και το αποθηκευτικό μέσο είναι ενεργό, η πρόσβαση στις προσωπικές πληροφορίες του χρήστη είναι ανοικτή για κάθε εισβολέα. Όμως όπως υπογραμμίσαμε στο σχεδιασμό της υπηρεσίας, αυτό είναι απαράδεκτο. Για το λόγο αυτό, δημιουργούμε ένα εφεδρικό μηχανισμό με τη μορφή προγράμματος φλοιού, που θα αποσυνδέει το δίσκο στη περίπτωση που υπάρχει αδράνεια. Για την υλοποίηση του προγράμματος θα βασιστούμε στα αρχεία του Apache Server, που περιλαμβάνουν τη χρονική στιγμή της κάθε συναλλαγής. Το πρόγραμμα θα υπολογίζει τη διαφορά της τελευταίας συναλλαγής με τον Android Client, και αν αυτή είναι

μεγαλύτερη από τη μέγιστη τιμή που θα θέσουμε, ο δίσκος θα αποσυνδεθεί.
Θα δημιουργήσουμε το αρχείο του προγράμματος transactioncheck.sh, που θα υλοποιεί τις παραπάνω απαιτήσεις με τις παρακάτω εντολές από ένα νέο τερματικό:

nano transactioncheck.sh #δημιουργία του αρχείου του προγράμματος και άνοιγμα για επεξεργασία

```
#!/bin/bash
#This script will not work in Solaris machines, as the -d option in date is not
POSIX-compliant.
#Copyright Andreas Tsagkaropoulos 2016


#Υπολογισμός της χρονικής διαφοράς, και αν αυτή είναι μεγαλύτερη των
#MAX_SECONDS_INACTIVITY δευτερολέπτων, αποσύνδεση του δίσκου.

SCRIPT_ACTIVE="true"                        #Κατά πόσο είναι ενεργό το πρόγραμμα
MAX_SECONDS_INACTIVITY=$((10*60))
SCRIPT_LOG="/home/pi/Desktop/TIMEDATE"

APACHE_LOG="/var/www/logs/ssl-access_log" # Τα αρχεία του apache για ασφαλείς
συνδέσεις αποθηκεύονται στη τοποθεσία  /var/www/logs/ssl-access_log (όπως
απαιτήσαμε στο αρχείο ρυθμίσεων του apache).Αν δουλεύουμε με HTTP, η τοποθεσία
πρέπει να αλλάξει
HTTP_APACHE_LOG="/var/log/apache2/other_vhosts_access.log"
MASTER_CUT_COMMAND="cut -d'[' -f2 | cut -c 1-26" #Εφαρμόζεται σε ένα apache log
τύπου vhost_combined και λαμβάνει τους χαρακτήρες 1-26 του δεύτερου πεδίου όπως
διαχωρίζεται από το χαρακτήρα [ .

last_transaction=`cat $APACHE_LOG | grep ownCloud-android |  tail -1 | eval
$MASTER_CUT_COMMAND | tr '/' ' '`
# Παρακάτω μετατρέπουμε την χρονική στιγμή σε μορφή που μπορεί να αναγνωριστεί
από την εντολή date.

last_transaction_date=`echo $last_transaction | cut -c 1-11`
last_transaction_time=`echo $last_transaction | cut -c 13-26`

last_transaction="$last_transaction_date $last_transaction_time"

current_time=`date '+%d %b %Y %H:%M:%S %z'`
StartDate=$(date -u -d "$last_transaction" +"%s")
FinalDate=$(date -u -d "$current_time" +"%s")

echo "FinalDate is "$FinalDate "and StartDate is "$StartDate >> $SCRIPT_LOG

timediff=$(($FinalDate - $StartDate))

echo `date` "Script active and time difference is" $timediff >> $SCRIPT_LOG
#Συγκρίνουμε τη διαφορά με τα MAX_SECONDS_INACTIVITY δευτερόλεπτα, και αν είναι
μεγαλύτερη διακόπτουμε τη σύνδεση.

if [[ ( $timediff -gt $MAX_SECONDS_INACTIVITY ) && ( $SCRIPT_ACTIVE = "true" )
]] ; then                              #αποσύνδεση δίσκου και εκτύπωση μηνύματος
στο σχετικό αρχείο.
   /home/pi/Desktop/mountscript.sh OFF
   echo  "Process "$$" turned off hard disk due to inactivity for "$timediff
"seconds." >> $SCRIPT_LOG
   echo >> $SCRIPT_LOG                                                  #κενή γραμμή
   kill -9 $$
```

Αφού αποθηκεύσουμε το παραπάνω αρχείο, το κάνουμε εκτελέσιμο,

chmod +x transactioncheck.sh

και ζητούμε από την υπηρεσία cron να εκτελέσει τη παραπάνω υπηρεσία κάθε λεπτό δίνοντας την εντολή
sudo crontab -e

Στη συνέχεια, προσθέτουμε στο τέλος του αρχείου τη παρακάτω γραμμή:

```
*   *   *   *   *    /home/pi/Desktop/transactioncheck.sh
```

Αποθηκεύουμε και κλείνουμε το αρχείο. Πλέον, η υπηρεσία μας ελέγχει κάθε λεπτό για πιθανή υπέρβαση του χρόνου αδρανείας, και αν αυτό είναι αληθές αποσυνδέει το δίσκο.

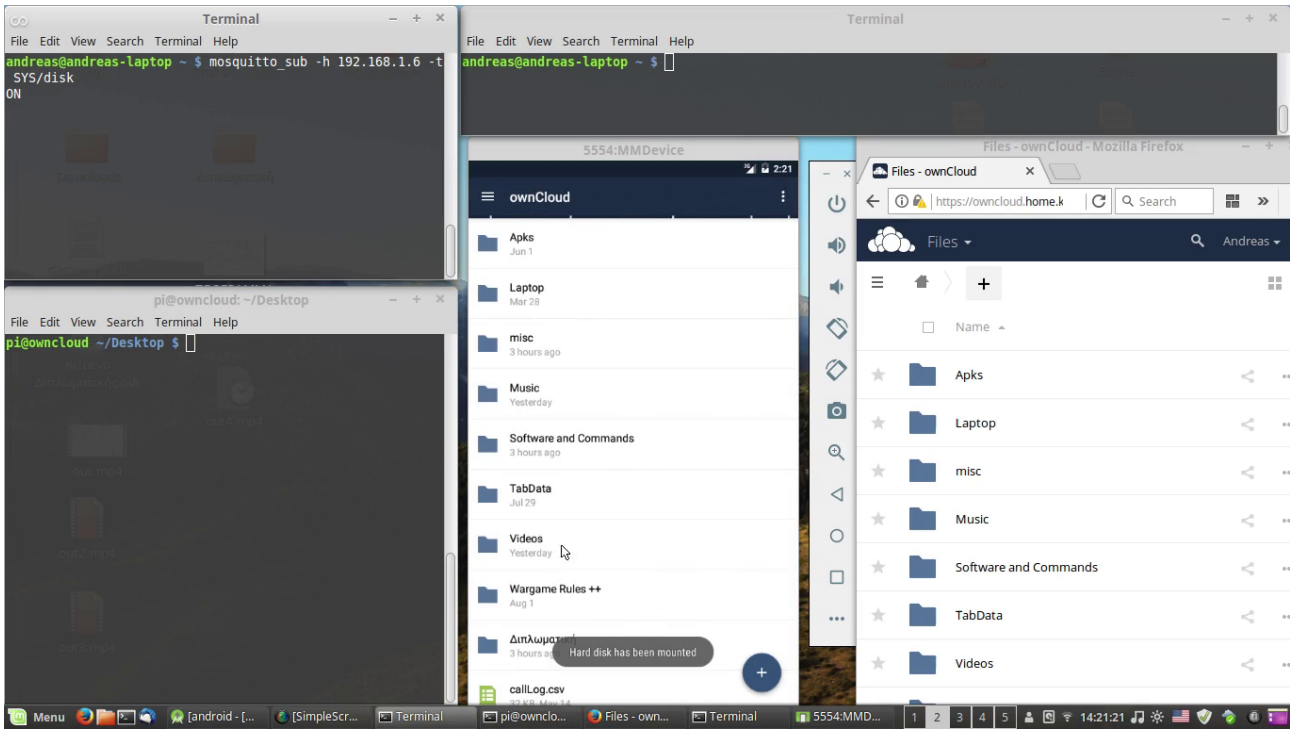
Τα αρχεία transactioncheck.sh, mqtt_listener.sh και mountscript.sh μπορούμε να τα ασφαλίσουμε από οποιαδήποτε προβολή, εκτέλεση και τροποποίηση των περιεχομένων τους από κάποιο πιθανό εισβολέα, εκτελώντας τις παρακάτω εντολές:

sudo chown root:root transactioncheck.sh
sudo chown root:root mqtt_listener.sh
sudo chown root:root mountscript.sh

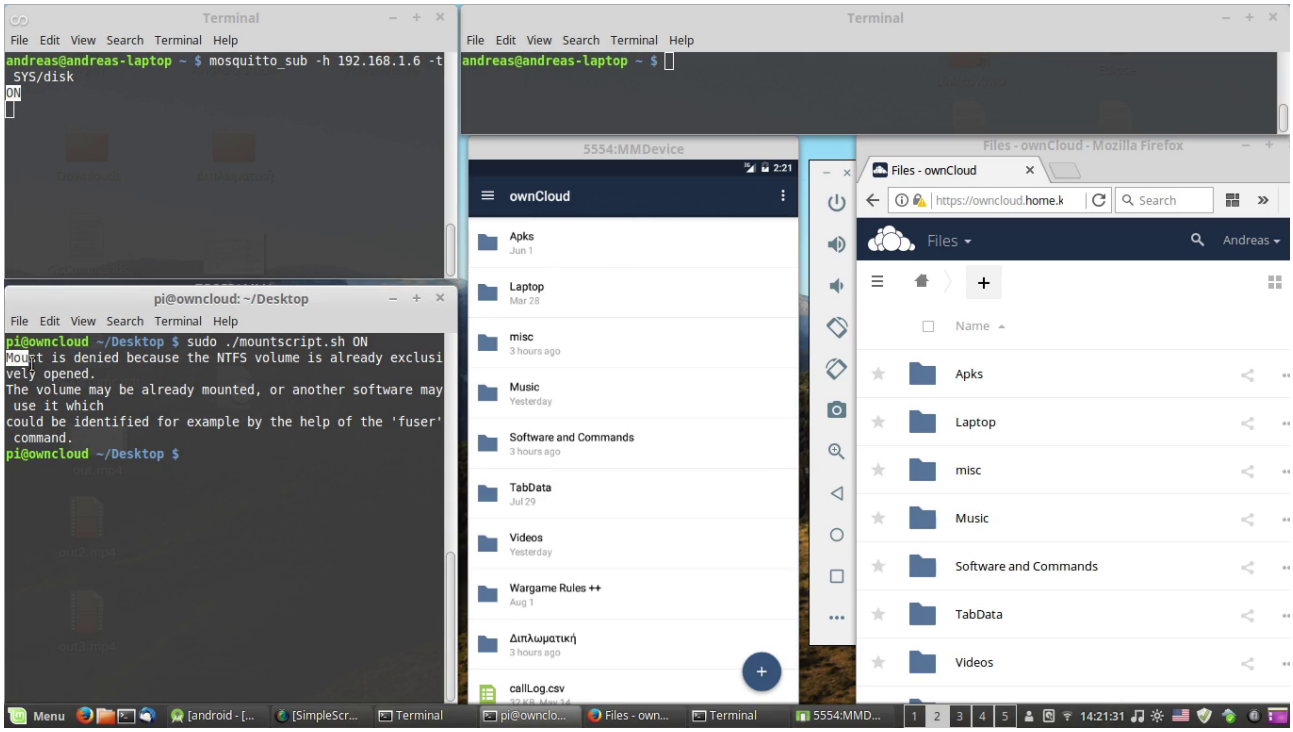
sudo chmod 700 transactioncheck.sh #Τα αρχεία μπορεί να τα προβάλει/επεξεργάζεται/εκτελεί μόνο ο διαχειριστής του συστήματος.
sudo chmod 700 mqtt_listener.sh
sudo chown 700 mountscript.sh

8 Σενάριο Χρήσης

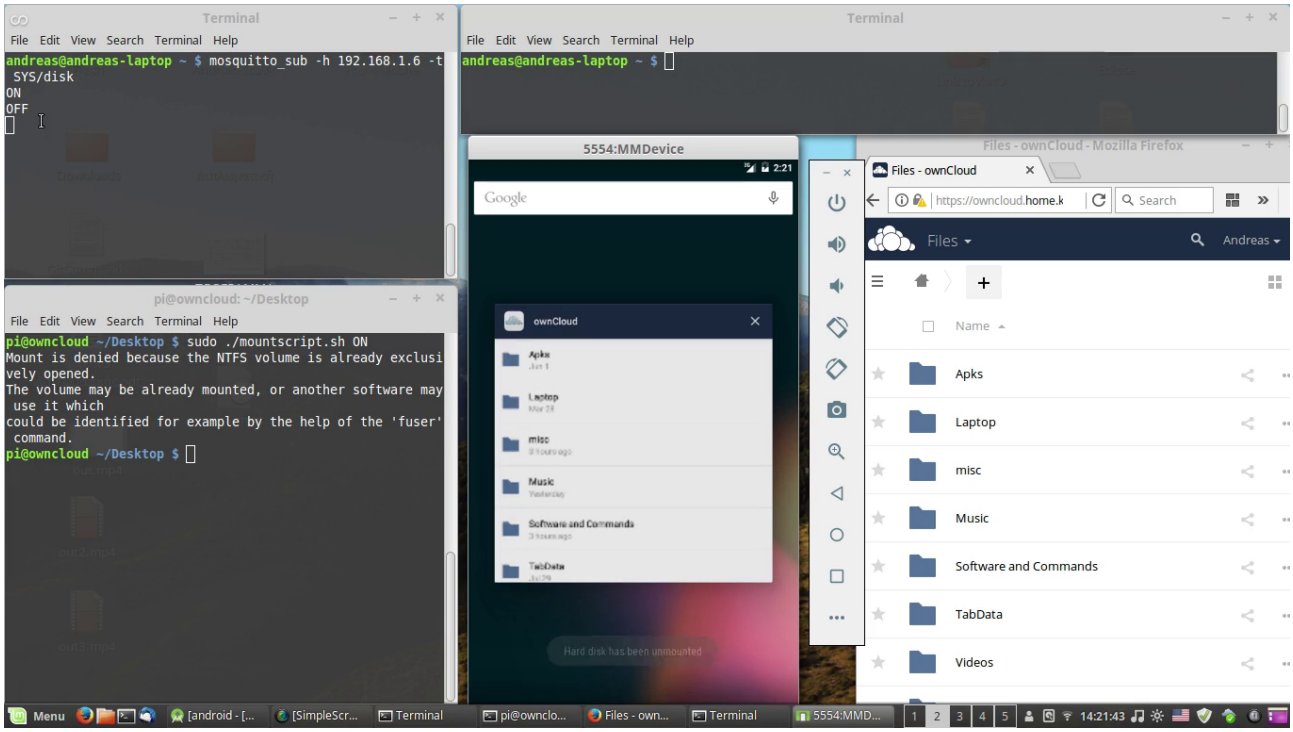
Παρακάτω, θα παρακολουθήσουμε ένα ολοκληρωμένο σενάριο χρήσης της υπηρεσίας, μέσα από κατάλληλα στιγμιότυπα. Ο χρήστης εκκινεί την εφαρμογή, στη συνέχεια την περνάει στο παρασκήνιο, και μετά την ενεργοποιεί εκ νέου. Τέλος, αφήνουμε την εφαρμογή Android σε κατάσταση αδρανείας οπότε εμφανίζεται η σχετική οθόνη, με τη βοήθεια της οποίας επανασυνδέουμε τον δίσκο.



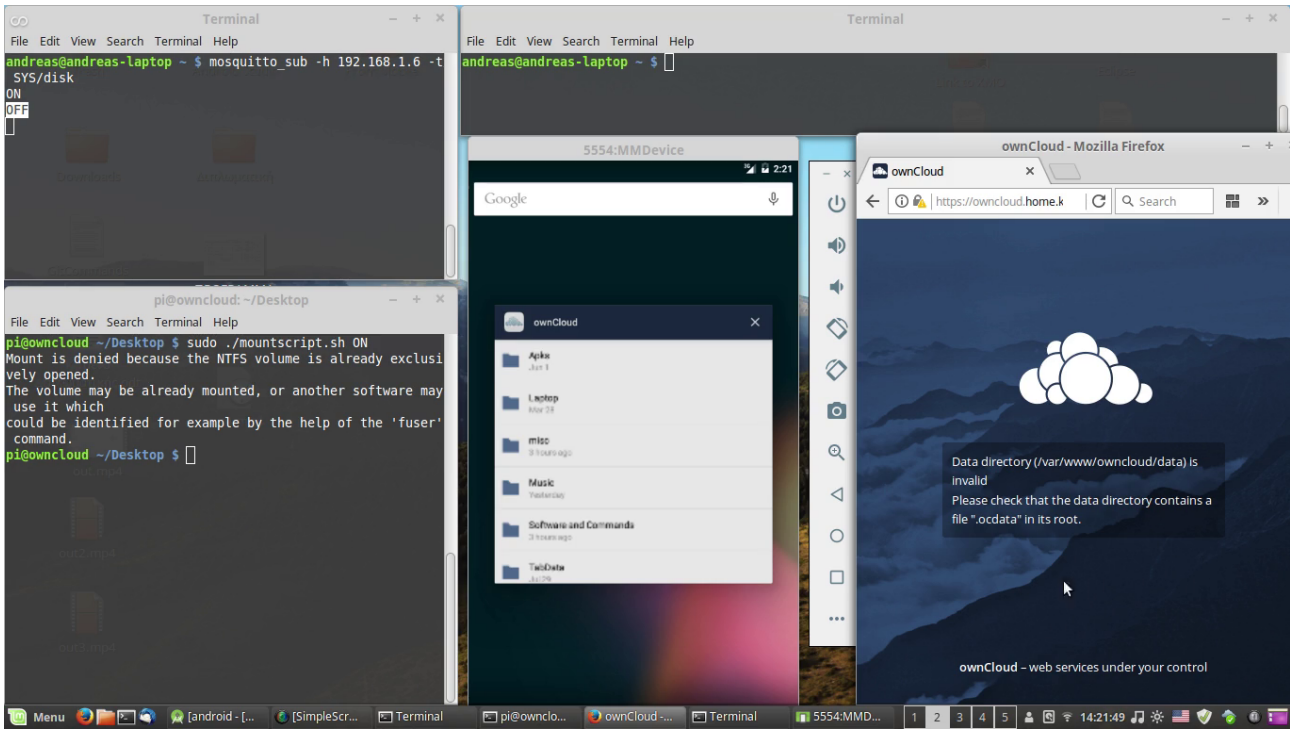
1. Αρχικά η εφαρμογή εκκινεί, και αποστέλλει μήνυμα "ON". Όπως παρατηρούμε στο παράθυρο του τερματικού πάνω αριστερά, το μήνυμα λαμβάνεται, και η εφαρμογή Android εμφανίζει αντίστοιχο μήνυμα με τη χρήση γραφικών Toast. Παρατηρούμε ότι ο Web Client είναι ενεργός στο κάτω-δεξιά παράθυρο.



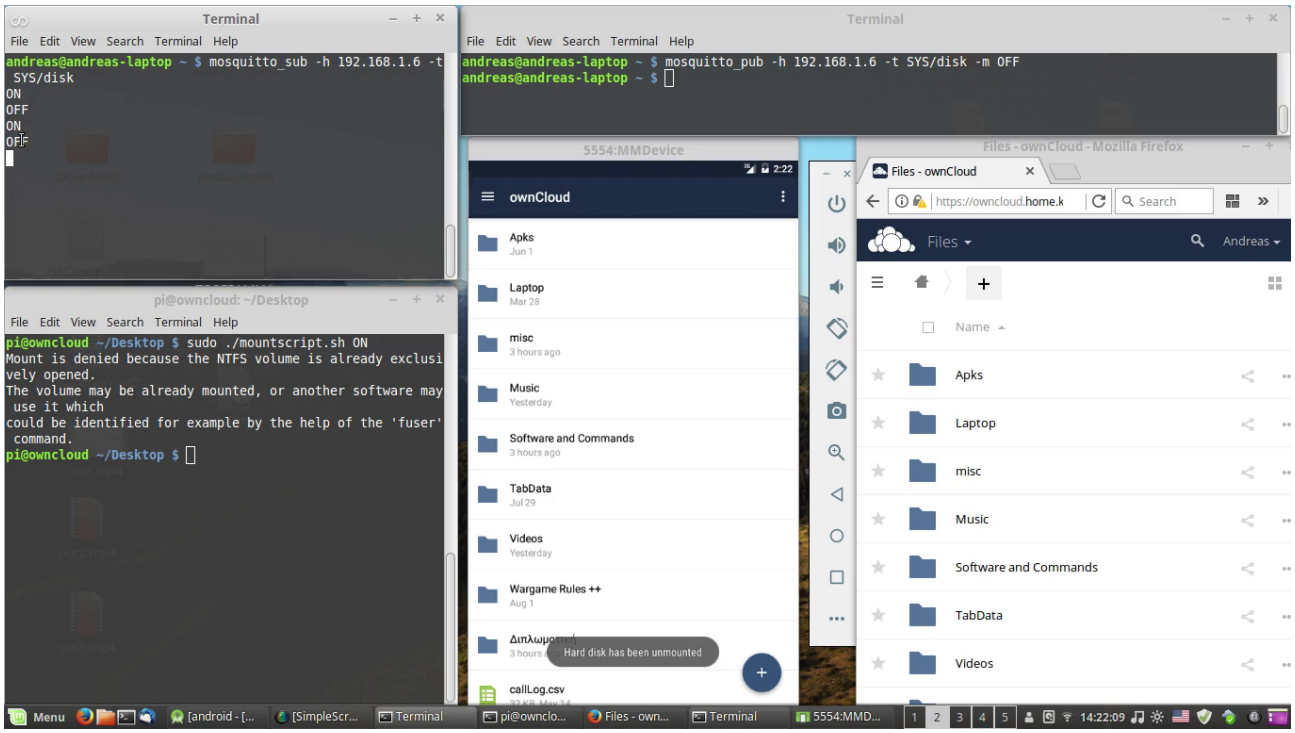
2. Παρατηρούμε ότι δεν μπορούμε να συνδέσουμε ξανά το δίσκο με τη χρήση του προγράμματος mountscript, επειδή ο δίσκος είναι συνδεδεμένος. Αυτό συμβαίνει επειδή εκτελείται το mountscript, μόλις λαμβάνεται μήνυμα ON από το mqttlistener στον server.



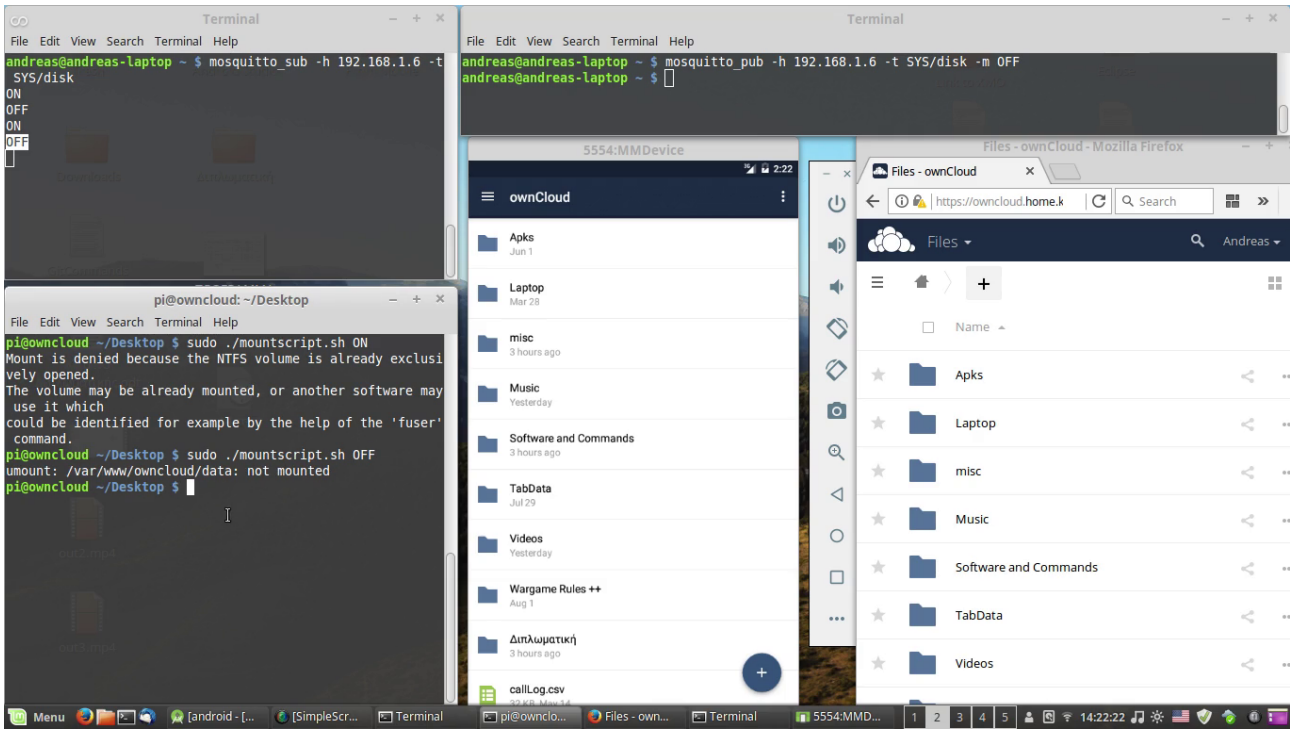
3. Τώρα, περνάμε την εφαρμογή στο παρασκήνιο της εκτέλεσης, και παρατηρούμε ότι στο παράθυρο της εφαρμογής Android, εμφανίζεται μήνυμα τύπου Toast που μας πληροφορεί για την αποσύνδεση του δίσκου. Την ίδια πληροφορία παίρνουμε και από το τερματικό πάνω-αριστερα, στη δεύτερη γραμμή μετά την εντολή.



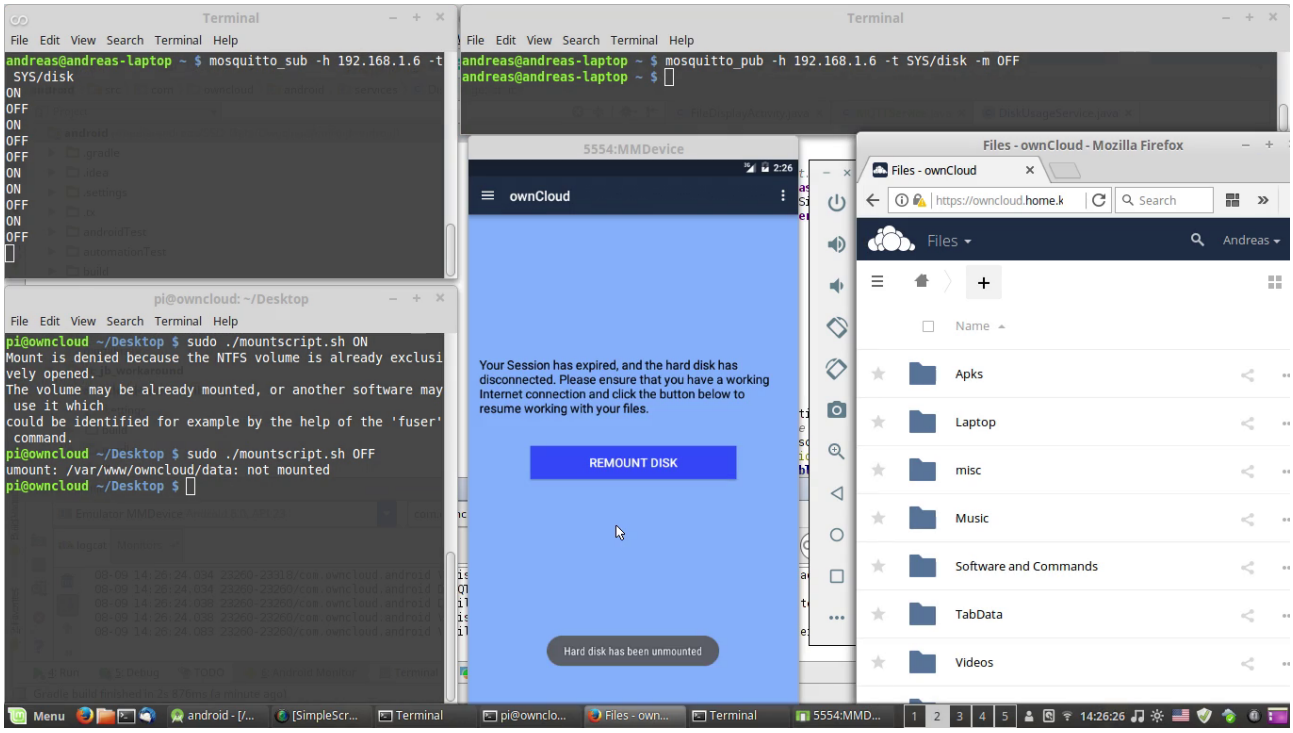
4. Επειδή η εφαρμογή πέρασε στο παρασκήνιο, πλέον δεν είναι προσβάσιμη η υπηρεσία ούτε μέσω του Web Client. Παρατηρούμε (κάτω-δεξιά παράθυρο) ότι δεν εμφανίζεται κάποιο πεδίο που επιτρέπει την είσοδο δεδομένων, και με τον τρόπο αυτό αποκλείονται οι επιθέσεις ωμής βίας.



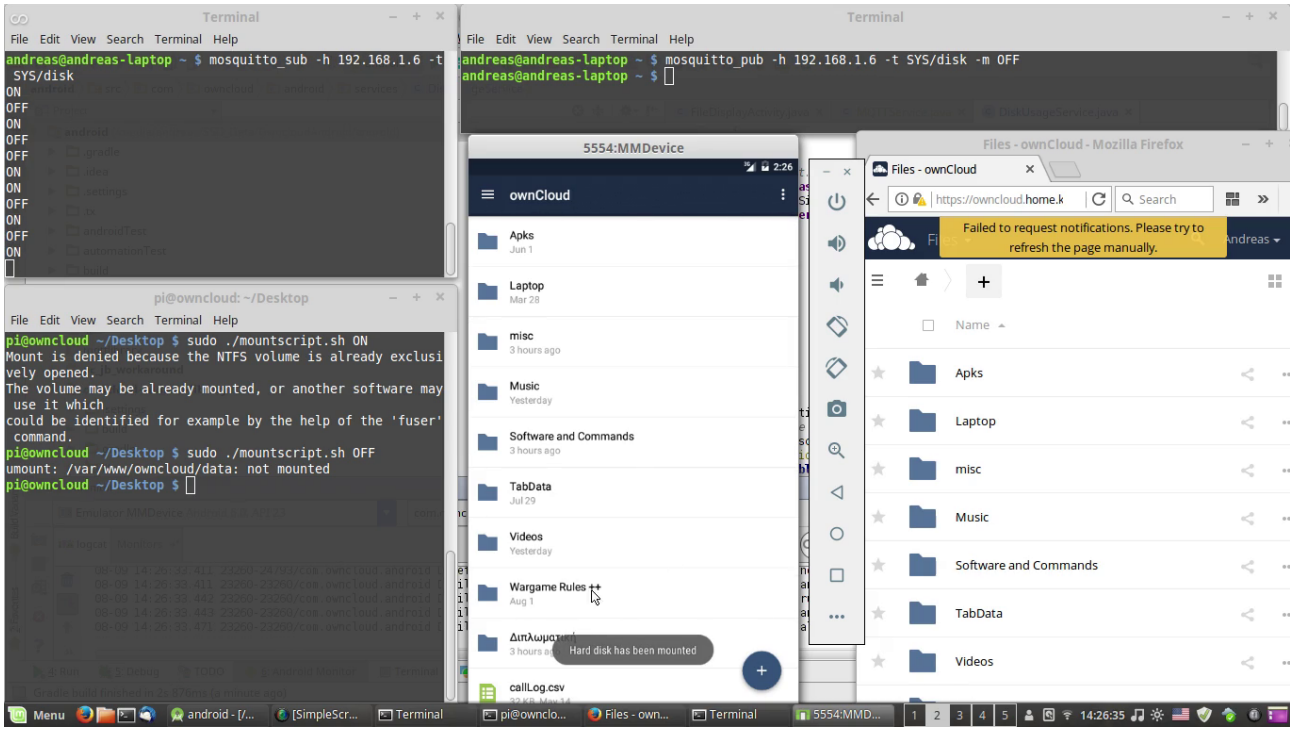
5. Αφού επανέλθουμε στην υπηρεσία ο δίσκος συνδέεται. Το στιγμιότυπο αυτό παρουσιάζει την περίπτωση που στέλνουμε ένα μήνυμα MQTT από ένα άλλο MQTT client (τον mosquitto_pub), ο οποίος τρέχει σε ένα άλλο υπολογιστή (πάνω δεξιά τερματικό). Παρατηρούμε ότι το λαμβάνει αυτόματα και η εφαρμογή και ανταποκρίνεται σε αυτό παρουσιάζοντας σχετικό μήνυμα πληροφοριών.



6. Στη συνέχεια, προσπαθούμε να αποσυνδέσουμε το δίσκο με την εκτέλεση της εντολής sudo ./mountscript OFF (κάτω-αριστερά τερματικό), αλλά πληροφορούμαστε ότι ο δίσκος δεν είναι συνδεδεμένος, όπως είναι αναμενόμενο.



7. Συνδέουμε και πάλι το δίσκο, και αφήνουμε το σύστημα αδρανές. Παρατηρούμε ότι εμφανίζεται στην οθόνη της συσκευής Android σχετικό μήνυμα που μας προτρέπει να επανασυνδέσουμε τον δίσκο. Στο κάτω μέρος της οθόνης παρατηρούμε το μήνυμα Toast που μας ενημερώνει για την αποσύνδεση του δίσκου.



8. Τέλος, παρατηρούμε ότι μόλις πατήσουμε το κουμπί REMOUNT DISK, συνδέεται ο σκληρός δίσκος και μας εμφανίζεται σχετικό μήνυμα Toast.

9 Συμπεράσματα και περαιτέρω προτάσεις

Στις παραπάνω ενότητες παρουσιάσαμε τη διαδικασία της υλοποίησης μιας ιδιωτικής, ασφαλούς υπηρεσίας νέφους. Είδαμε τη διαδικασία που απαιτείται για να εγκαταστήσουμε το απαραίτητο λογισμικό στον εξυπηρετητή, τις απαραίτητες ενέργειες ώστε να είναι προσβάσιμος με ασφάλεια από το διαδίκτυο, και τη προσθήκη δικλειδών ασφαλείας στην εφαρμογή Android. Η υπηρεσία παρέχει τη δυνατότητα χρήσης του νέφους με ασφαλή τρόπο, ανταποκρινόμενη στις ανάγκες ενός σύγχρονου οικιακού χρήστη .
Η ανάπτυξη των παραπάνω δυνατοτήτων δεν ήταν εύκολη. Καταρχήν, μολονότι υπάρχει πολύ σχετικό υλικό για τον Apache Server και το Android, πολλές φορές ήταν δύσκολο να εντοπίσουμε τι ήταν εφαρμόσιμο στη διάταξή μας, ή τι έπρεπε να χρησιμοποιήσουμε. Στη συνέχεια, η ανάπτυξη της εφαρμογής Android είχε την επιπρόσθετη δυσκολία της έλλειψης καθοδήγησης, διότι η εφαρμογή owncloud – αν και επαγγελματικού επιπέδου - δεν συνοδεύεται από κάποιο εγχειρίδιο για προγραμματιστές. Στο σημείο αυτό με βοήθησε πολύ η συνεργασία μου με τον κ. Λυμπερόπουλο, στα πλαίσια της οποίας έγινε δυνατό να υπερπηδήσουμε αυτό το εμπόδιο, και στη συνέχεια να προχωρήσουμε με τη βοήθεια του debugger, με χρήση των οδηγιών του Android προς προγραμματιστές, με αναζητήσεις στο διαδίκτυο και με εκτενείς δοκιμές. Τέλος, υπήρξαν και περιπτώσεις που η ύπηρεσία μας σταμάτησε να λειτουργεί φυσιολογικά, χωρίς να μπορούμε να διορθώσουμε τα σφάλματα. Για την υπέρβαση αυτού του εμποδίου χρησιμοποιήσαμε τα αντίγραφα ασφαλείας που δημιουργήσαμε σε τακτική βάση και τις σημειώσεις που είχαμε κρατήσει κατά τη διάρκεια της υλοποίησης.
Βεβαίως, υπάρχουν αρκετά σημεία της υπηρεσίας τα οποία θα μπορούσαμε να επιμεληθούμε περισσότερο, τα οποία όμως απαιτούν χρόνο αρκετά μεγαλύτερο του ενός εξαμήνου που διατέθηκε για την εκπόνηση της εργασίας. Παρακάτω παραθέτουμε ιδέες για μερικές επεκτάσεις που μπορούν να γίνουν στην υπηρεσία:

- ***Γραφικά***

  Αν και η γραφική διεπαφή δεν είχε πρωτεύοντα ρόλο στην ανάπτυξη της εφαρμογής, είναι σημαντική διότι τα τρέχοντα στανταρ στο τομέα είναι ιδιαίτερα υψηλά. Επομένως, θα μπορούσαμε να παροτρύνουμε το χρήστη να συμπληρώσει τη διεύθυνση του Broker και το Topic, με ένα γραφικό στοιχείο πιο κατατοπιστικό από ένα απλό Toast που δείχνουμε τώρα, π.χ όταν ανοίγει για πρώτη φορά την εφαρμογή. Όπως έχει η εφαρμογή, ο χρήστης θα πρέπει να προσθέτει μόνος του κάθε φορά τη διεύθυνση του Broker και το Topic για να πετύχει τη λειτουργία της εφαρμογής με αυξημένη ασφάλεια.

- ***Δικαιώματα Χρήσης και Εμπορική Εκμετάλλευση***

  Η εφαρμογή όπως έχει δημιουργηθεί μέχρι αυτή τη στιγμή, προορίζεται αποκλειστικά για εκπαιδευτικούς σκοπούς, και για ιδιωτική χρήση του συγγραφέα. Αυτό οφείλεται στην ασυμβατότητα που υπάρχει μεταξύ των αδειών χρήσης της εφαρμογής owncloud-android (GPL version 3) , και της βιβλιοθήκης wmqtt.jar (IBM proprietary). Στη περίπτωση που το ζητούμενο είναι η εμπορική εκμετάλλευση της εφαρμογής, θα πρέπει να αποκτήσουμε πρόσβαση στην εμπορική έκδοση της πλατφόρμας owncloud.

- ***MQTT Broker***
  Στην υλοποίησή μας, ο Broker βρισκόταν στην ίδια φυσική συσκευή με τον Owncloud-

Server, και λειτουργούσε με απλά μηνύματα ON/OFF. Σε μια υλοποίηση που θα απατούσαμε μέγιστη ασφάλεια, αφενός θα έπρεπε να αφαιρέσουμε κάθε χρήσιμη πληροφορία από τα μηνύματα (π.χ μετατροπή σε 0/1), αφετέρου θα έπρεπε να χρησιμοποιήσουμε κρυπτογράφηση με SSL – κάτι που επιβαρύνει όμως αρκετά την επικοινωνία. Επίσης, θα ήταν καλύτερο να χρησιμοποιήσουμε ένα Server υπό τον έλεγχό μας εκτός οικιακού δικτύου για να φιλοξενήσουμε τον Broker, ώστε να είναι δύσκολο να εντοπιστεί η αιτία και ο προορισμός της κίνησης.

- ***Εγκαταστάσεις με πολλούς χρήστες***

  Στην υλοποίηση που περιγράφηκε, υποθέσαμε ότι η υπηρεσία θα υποστηρίζει οικιακή χρήση. Η παρουσία περισσότερων των 4-5 ενεργών χρηστών δημιουργεί προβλήματα στην ανίχνευση αδράνειας από τη πλευρά του server (τα δεδομένα του χρήστη μπορεί να μείνουν προσβάσιμα για πολύ καιρό διότι υπάρχει κοινό αρχείο apache, και κοινό μέσο αποθήκευσης) και επίσης στη χρήση του MQTT, διότι η δραστηριότητα του ενός μπορεί να εμποδίζει ή να διακόπτει τη δραστηριότητα του άλλου. Μία πιθανή λύση είναι να έχει ο κάθε χρήστης ένα προσωπικό (hashed) θέμα – που μπορεί να παράγεται στο χρόνο εκτέλεσης, με επικοινωνία με μια αρμόδια υπηρεσία, ή με βάση κάποιους κανόνες – στο οποίο να στέλνει μηνύματα MQTT. Όταν αποστέλλεται μήνυμα προς αυτό το προσωπικό θέμα, θα συσχετίζεται η διεύθυνση IP με το συγκεκριμένο χρήστη, και έτσι – αν υποθέσουμε ότι ο κάθε χρήστης έχει το δικό του partition για να αποθηκεύονται τα δεδομένα του σε κάποιο server– να διαχειριζόμαστε τον αποθηκευτικό χώρο του κάθε χρήστη ξεχωριστά. Η δημιουργία ξεχωριστού partition για κάθε χρήστη είναι δυνατή, διότι αφενός μπορούμε να χρησιμοποιήσουμε το σύστημα GPT, που επιτρέπει μέχρι και 128 partitions σε κάθε δίσκο, αφετέρου με τη χρήση logical volumes, μπορούμε να αυξήσουμε όσο θέλουμε τον αριθμό αυτό.

- ***Έκδοση του Owncloud-Server***

  Πολλές φορές, η πρόσβαση που αποκτά κάποιος εισβολέας στη συσκευή, οφείλεται στις παλαιές εκδόσεις λογισμικού που είναι εγκατεστημένες στη συσκευή. Στην υλοποίησή μας, αυτό παρατηρείται κατά την εγκατάσταση του Owncloud-Server, καθώς εγκαθίσταται η έκδοση 7 που υπάρχει στις αποθήκες λογισμικού (repositories) του Raspbian, ενώ η πιο πρόσφατη έκδοση της εφαρμογής είναι η 9.1 τη στιγμή που παρουσιάζεται αυτή η εργασία. Η έκδοση 7 αντιμετώπισε αρκετά προβλήματα [12], αλλά τουλάχιστον τα πιο σοβαρά από αυτά διορθώθηκαν. Η εκ βάθρων αντιμετώπιση αυτού του προβλήματος μπορεί να γίνει με τη χρήση της εφαρμογής updater που έχει ο Owncloud-Server, ώστε να εγκατασταθεί η νεότερη έκδοση του προγράμματος από τον ίδιο τον server.

- ***Επέκταση της ιδέας της εφαρμογής σε άλλα Λειτουργικά συστήματα***

  Η ιδέα της εφαρμογής Android μπορεί να υλοποιηθεί και σε άλλα λειτουργικά συστήματα, π.χ για Linux, με τη βοήθεια ενός προγράμματος-περιτυλίγματος (wrapper), ή και με άμεση επέμβαση στον κώδικα της εφαρμογής. Στη περίπτωση ενός wrapper, το τελικό πρόγραμμα μπορεί να στέλνει μήνυμα σύνδεσης στο δίσκο κατά την έναρξη και μήνυμα αποσύνδεσης κατά την λήξη της εκτέλεσης, όπως και να διαχειρίζεται τα μηνύματα MQTT που θα δημοσιεύονται στο θέμα της εφαρμογής στο Broker.

- ***Κρυπτογράφηση από άκρη σε άκρη με χρήση TOR.***

Αν και δεν χρησιμοποιείται στο κώδικα της εφαρμογής, η επικοινωνία client και server, μπορεί να γίνει και μέσω του δικτύου ανωνυμίας TOR, τόσο για τα μηνύματα https, όσο και για τα μηνύματα MQTT. Αυτό δοκιμάστηκε με τις εφαρμογές Orwall και Orbot, οι οποίες κρυπτογράφησαν επιτυχώς το κανάλι επικοινωνίας μεταξύ του Android client και του Owncloud-Server.Τα πλεονεκτήματα στον τομέα της ασφάλειας είναι μεγάλα, αλλά η μεταφορά πληροφοριών από και πρός την εφαρμογή επιβαρύνεται από τη κρυπτογράφηση.

- ***Αντιμετώπιση root privilege escalation***

  Η υπηρεσία που αναπτύξαμε, αποσυνδέει το δίσκο μόλις σταματήσει να χρησιμοποιείται και αυτό σημαίνει ότι κανένας χρήστης του συστήματος δεν μπορεί να δει την ύπαρξή του, ή να τον συνδέσει. Ο μόνος που μπορεί να το κάνει είναι ο διαχειριστής συστήματος (root). Δυστυχώς, μια κοινή στρατηγική όσων αποκτούν πρόσβαση σε ένα εξυπηρετητή είναι να χρησιμοποιούν ευαίσθητα σημεία του πυρήνα linux (kernel vulnerabilities), προκειμένου να αποκτήσουν διαχειριστική πρόσβαση στο σύστημα. Βέβαια ένα ενδεχόμενο επιτυχημένης εκμετάλλευσης μιας τέτοιας αδυναμίας είναι πολύ σπάνιο, και συνήθως απαιτεί ιδιαίτερες ικανότητες. Στη περίπτωση όμως που συμβεί, ο εισβολέας μπορεί να συνδέσει το δίσκο, να σταματήσει όλες τις υπηρεσίες ασφαλείας, και να έχει απεριόριστη πρόσβαση στο σύστημα και τα δεδομένα του. Για την αντιμετώπιση αυτού του ενδεχομένου – εφόσον θέλουμε πάρα πολύ μεγάλη ασφάλεια στο σύστημά μας – θα πρέπει το μήνυμα OFF που στέλνει η συσκευή μας στο θέμα του δίσκου, να κλείνει την *παροχή ρεύματος* προς τον δίσκο. Αυτό προϋποθέτει βέβαια ότι η συσκευή παροχής ρεύματος θα μπορεί να επικοινωνεί με τον MQTT Broker (τον οποίο με τη σειρά του θα πρέπει να προστατέψουμε από τυχόν επιθέσεις) και ότι δεν θα υπάρχει δυνατότητα παρεμπόδισης της λειτουργίας της. Στη περίπτωση αυτή συστήνουμε την τοποθέτηση του MQTT Broker σε άλλο δίκτυο (και εννοείται σε διαφορετικό μηχάνημα) απ' αυτό του Owncloud-Server.

10 Παραπομπές

[1] http://wwlw.telegraph.co.uk/technology/google/10459347/Google-pays-17-million-compensation-over-privacy-breach.htm και επίσης http://bits.blogs.nytimes.com/2015/04/08/f-c-c-fines-att-25-million-for-privacy-breach/?_r=0

[2] https://www.google.com/intl/en/policies/terms/
[3] https://www.dropbox.com/terms
[4] http://articles.latimes.com/2010/sep/16/business/la-fi-google-firing-20100916
[5] http://motherboard.vice.com/read/another-day-another-hack-117-million-linkedin-emails-and-password
[6] Βλέπε http://www.speedguide.net/port.php?port=4431 και http://www.speedguide.net/port.php?port=8000
[7] https://developer.android.com/reference/android/app/Activity.html#ActivityLifecycle
[8] https://developer.android.com/reference/android/app/Service.html#ServiceLifecycle

[9] https://developer.android.com/training/basics/activity-lifecycle/starting.html
[10] https://developer.android.com/studio/index.html
[11] http://dalelane.co.uk/blog/?p=1599
[12] http://www.cvedetails.com/vulnerability-list.php?vendor_id=11929&product_id=22262&version_id=182745&page=1&hasexp=0&opdos=0&opec=0&opov=0&opcsrf=0&opgpriv=0&opsqli=0&opxss=0&opdirt=0&opmemc=0&ophttprs=0&opbyp=0&opfileinc=0&opginf=0&cvssscoremin=0&cvssscor

10.1 Χρήσιμοι Ιστότοποι

http://ubuntuserverguide.com/2013/04/how-to-setup-owncloud-server-5-with-ssl-connection.html

11 Παράρτημα Α – Κώδικας εφαρμογής Android

Παρακάτω παραθέτουμε τον πλήρη κώδικα, και την τοποθεσία αποθήκευσης, για τα αρχεία που μεταβάλλαμε ή δημιουργήσαμε.

11.1 android/res/values/strings.xmlstrings.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="about_android">%1$s Android app</string>
    <string name="about_version">version %1$s</string>
    <string name="actionbar_sync">Refresh account</string>
    <string name="actionbar_upload">Upload</string>
    <string name="actionbar_upload_from_apps">Content from other apps</string>
    <string name="actionbar_upload_files">Files</string>
    <string name="actionbar_open_with">Open with</string>
    <string name="actionbar_mkdir">New folder</string>
    <string name="actionbar_settings">Settings</string>
    <string name="actionbar_see_details">Details</string>
    <string name="actionbar_send_file">Send</string>
    <string name="actionbar_sort">Sort</string>
    <string name="actionbar_sort_title">Sort by</string>
    <string-array name="actionbar_sortby">
       <item>A-Z</item>
       <item>Newest - Oldest</item>
       <!-- TODO re-enable when server-side folder size calculation is available
       <item>Biggest - Smallest</item>  -->
    </string-array>
    <!-- TODO re-enable when "Accounts" is available in Navigation Drawer -->
    <!--<string name="drawer_item_accounts">Accounts</string>-->
    <string name="drawer_item_all_files">All files</string>
    <!-- TODO re-enable when "On Device" is available
    <string name="drawer_item_on_device">On device</string>-->
    <string name="drawer_item_settings">Settings</string>
    <string name="drawer_item_uploads_list">Uploads</string>
   <string name="drawer_close">Close</string>
    <string name="drawer_open">Open</string>
    <string name="prefs_category_general">General</string>
    <string name="prefs_category_more">More</string>
    <string name="prefs_accounts">Accounts</string>
    <string name="prefs_manage_accounts">Manage accounts</string>
    <string name="prefs_passcode">Passcode lock</string>
    <string name="prefs_instant_upload">Instant picture uploads</string>
    <string name="prefs_instant_upload_summary">Instantly upload pictures taken by
camera</string>
    <string name="prefs_instant_video_upload">Instant video uploads</string>
    <string name="prefs_instant_video_upload_summary">Instantly upload videos recorded
by camera</string>
    <string name="prefs_log_title">Enable logging</string>
    <string name="prefs_log_summary">This is used to log problems</string>
    <string name="prefs_log_title_history">Logging history</string>
    <string name="prefs_log_summary_history">This shows the recorded logs</string>
    <string name="prefs_log_delete_history_button">Delete history</string>
```

```
    <string name="prefs_help">Help</string>
    <string name="prefs_recommend">Recommend to a friend</string>
    <string name="prefs_feedback">Feedback</string>
    <string name="prefs_imprint">Imprint</string>
    <string name="prefs_remember_last_share_location">Remember share location</string>
    <string name="prefs_remember_last_upload_location_summary">Remember last share
upload location</string>
   <string name="recommend_subject">"Try %1$s on your smartphone!"</string>
   <string name="recommend_text">"I want to invite you to use %1$s on your
smartphone!\nDownload here: %2$s"</string>
    <string name="auth_check_server">Check server</string>
    <string name="auth_host_url">Server address https://…</string>
    <string name="auth_username">Username</string>
    <string name="auth_password">Password</string>
    <string name="auth_register">New to %1$s?</string>
    <string name="sync_string_files">Files</string>
    <string name="setup_btn_connect">Connect</string>
    <string name="uploader_btn_upload_text">Upload</string>
    <string name="uploader_top_message">Choose upload folder</string>
    <string name="uploader_wrn_no_account_title">No account found</string>
    <string name="uploader_wrn_no_account_text">There are no %1$s accounts on your
device. Please set up an account first.</string>
    <string name="uploader_wrn_no_account_setup_btn_text">Setup</string>
    <string name="uploader_wrn_no_account_quit_btn_text">Quit</string>
    <string name="uploader_error_title_no_file_to_upload">No file to upload</string>
    <string name="uploader_error_message_received_piece_of_text">%1$s cannot upload a
piece of text as a file.</string>
    <string name="uploader_error_message_no_file_to_upload">Received data do not include
any valid file.</string>
    <string name="uploader_error_title_file_cannot_be_uploaded">File cannot be
uploaded</string>
    <string name="uploader_error_message_read_permission_not_granted">%1$s is not
allowed to read a received file</string>
    <string name="uploader_error_message_source_file_not_found">File to upload was not
found in its location. Please check whether the file exists.</string>
    <string name="uploader_error_message_source_file_not_copied">An error occurred while
copying the file to a temporary folder. Please try to send again.</string>
    <string name="file_list_seconds_ago">seconds ago</string>
    <string name="file_list_empty">Nothing in here. Upload something!</string>
    <string name="file_list_loading">Loading&#8230;</string>
    <string name="file_list_no_app_for_file_type">No app found for file type!</string>
    <string name="local_file_list_empty">There are no files in this folder.</string>
    <string name="upload_list_empty">No uploads available.</string>
    <string name="file_list_folder">folder</string>
    <string name="file_list_folders">folders</string>
    <string name="file_list_file">file</string>
    <string name="file_list_files">files</string>
    <string name="filedetails_select_file">Tap on a file to display additional
information.</string>
    <string name="filedetails_size">Size:</string>
    <string name="filedetails_type">Type:</string>
    <string name="filedetails_created">Created:</string>
    <string name="filedetails_modified">Modified:</string>
    <string name="filedetails_download">Download</string>
    <string name="filedetails_sync_file">Synchronize</string>
    <string name="filedetails_renamed_in_upload_msg">File was renamed to %1$s during
upload</string>
    <string name="list_layout">List layout</string>
    <string name="action_share">Share</string>
    <string name="common_yes">Yes</string>
    <string name="common_no">No</string>
    <string name="common_ok">OK</string>
```

```
    <string name="common_remove_upload">Remove upload</string>
    <string name="common_retry_upload">Retry upload</string>
    <string name="common_cancel_sync">Cancel sync</string>
    <string name="common_cancel">Cancel</string>
    <string name="common_back">Back</string>
    <string name="common_save_exit">Save &amp;amp; exit</string>
    <string name="common_error">Error</string>
    <string name="common_loading">Loading &amp;#8230;</string>
    <string name="common_unknown">unknown</string>
    <string name="common_error_unknown">Unknown error</string>
    <string name="about_title">About</string>
    <string name="change_password">Change password</string>
    <string name="delete_account">Remove account</string>
    <string name="create_account">Create account</string>
    <string name="upload_chooser_title">Upload from &amp;#8230;</string>
    <string name="uploader_info_dirname">Folder name</string>
    <string name="uploader_upload_in_progress_ticker">Uploading &amp;#8230;</string>
    <string name="uploader_upload_in_progress_content">%1$d%% Uploading %2$s</string>
    <string name="uploader_upload_succeeded_ticker">Upload succeeded</string>
    <string name="uploader_upload_succeeded_content_single">%1$s uploaded</string>
    <string name="uploader_upload_failed_ticker">Upload failed</string>
    <string name="uploader_upload_failed_content_single">Upload of %1$s could not be
completed</string>
    <string name="uploader_upload_failed_credentials_error">Upload failed, you need to
log in again</string>
    <string name="uploads_view_title">Uploads</string>
    <string name="uploads_view_group_current_uploads">Current</string>
    <string name="uploads_view_group_failed_uploads">Failed (tap to retry)</string>
    <string name="uploads_view_group_finished_uploads">Uploaded</string>
    <string name="uploads_view_upload_status_succeeded">Completed</string>
    <string name="uploads_view_upload_status_cancelled">Cancelled</string>
    <string name="uploads_view_upload_status_paused">Paused</string>
    <string name="uploads_view_upload_status_failed_connection_error">Connection
error</string>
    <string name="uploads_view_upload_status_failed_retry">Upload will be retried
shortly</string>
    <string name="uploads_view_upload_status_failed_credentials_error">Credentials
error</string>
    <string name="uploads_view_upload_status_failed_folder_error">Folder error</string>
    <string name="uploads_view_upload_status_failed_file_error">File error</string>
    <string name="uploads_view_upload_status_failed_localfile_error">Local file not
found</string>
    <string name="uploads_view_upload_status_failed_permission_error">Permission
error</string>
    <string name="uploads_view_upload_status_conflict">Conflict</string>
    <string name="uploads_view_upload_status_service_interrupted">App was
terminated</string>
    <string name="uploads_view_upload_status_unknown_fail">Unknown error</string>
    <string name="uploads_view_upload_status_waiting_for_wifi">Waiting for wifi
connectivity</string>
    <string name="uploads_view_later_waiting_to_upload">Waiting to upload</string>
    <string name="downloader_download_in_progress_ticker">Downloading &amp;#8230;</string>
    <string name="downloader_download_in_progress_content">%1$d%% Downloading
%2$s</string>
    <string name="downloader_download_succeeded_ticker">Download succeeded</string>
    <string name="downloader_download_succeeded_content">%1$s downloaded</string>
    <string name="downloader_download_failed_ticker">Download failed</string>
    <string name="downloader_download_failed_content">Download of %1$s could not be
completed</string>
    <string name="downloader_not_downloaded_yet">Not downloaded yet</string>
    <string name="downloader_download_failed_credentials_error">Download failed, you
need to log in again</string>
```

```
    <string name="common_choose_account">Choose account</string>
    <string name="sync_fail_ticker">Sync failed</string>
    <string name="sync_fail_ticker_unauthorized">Sync failed, you need to log in
again</string>
    <string name="sync_fail_content">Sync of %1$s could not be completed</string>
    <string name="sync_fail_content_unauthorized">Invalid password for %1$s</string>
    <string name="sync_conflicts_in_favourites_ticker">Conflicts found</string>
    <string name="sync_conflicts_in_favourites_content">%1$d kept-in-sync files could
not be sync\'ed</string>
    <string name="sync_fail_in_favourites_ticker">Kept-in-sync files failed</string>
    <string name="sync_fail_in_favourites_content">Contents of %1$d files could not be
sync\'ed (%2$d conflicts)</string>
    <string name="sync_foreign_files_forgotten_ticker">Some local files were
forgotten</string>
    <string name="sync_foreign_files_forgotten_content">%1$d files out of the %2$s
folder could not be copied into</string>
    <string name="sync_foreign_files_forgotten_explanation">As of version 1.3.16, files
uploaded from this device are copied into the local %1$s folder to prevent data loss
when a single file is synced with multiple accounts.\n\nDue to this change, all files
uploaded in previous versions of this app were copied into the %2$s folder. However, an
error prevented the completion of this operation during account synchronization. You may
either leave the file(s) as is and remove the link to %3$s, or move the file(s) into the
%1$s folder and retain the link to %4$s.\n\nListed below are the local file(s), and the
remote file(s) in %5$s they were linked to.</string>
    <string name="sync_current_folder_was_removed">Folder %1$s does not exist
anymore</string>
    <string name="foreign_files_move">"Move all"</string>
    <string name="foreign_files_success">"All files were moved"</string>
    <string name="foreign_files_fail">"Some files could not be moved"</string>
    <string name="foreign_files_local_text">"Local: %1$s"</string>
    <string name="foreign_files_remote_text">"Remote: %1$s"</string>
    <string name="upload_query_move_foreign_files">There is not enough space to copy the
selected files into the %1$s folder. Would you like to move them instead? </string>
    <string name="pass_code_enter_pass_code">Please insert your passcode</string>

    <string name="pass_code_configure_your_pass_code">Enter your passcode</string>
    <string name="pass_code_configure_your_pass_code_explanation">The passcode will be
requested every time the app is started</string>
    <string name="pass_code_reenter_your_pass_code">Please reenter your
passcode</string>
    <string name="pass_code_remove_your_pass_code">Remove your passcode</string>
    <string name="pass_code_mismatch">The passcodes are not the same</string>
    <string name="pass_code_wrong">Incorrect passcode</string>
    <string name="pass_code_removed">Passcode removed</string>
    <string name="pass_code_stored">Passcode stored</string>

    <string name="media_notif_ticker">"%1$s music player"</string>
    <string name="media_state_playing">"%1$s (playing)"</string>
    <string name="media_state_loading">"%1$s (loading)"</string>
    <string name="media_event_done">"%1$s playback finished"</string>
    <string name="media_err_nothing_to_play">No media file found</string>
    <string name="media_err_no_account">No account provided</string>
    <string name="media_err_not_in_owncloud">File not in a valid account</string>
    <string name="media_err_unsupported">Unsupported media codec</string>
    <string name="media_err_io">Media file could not be read</string>
    <string name="media_err_malformed">Media file not correctly encoded</string>
    <string name="media_err_timeout">Timed out while trying to play</string>
    <string name="media_err_invalid_progressive_playback">Media file cannot be
streamed</string>
    <string name="media_err_unknown">Media file cannot be played with the stock media
player</string>
    <string name="media_err_security_ex">Security error trying to play %1$s</string>
    <string name="media_err_io_ex">Input error trying to play %1$s</string>
```

```
    <string name="media_err_unexpected">Unexpected error trying to play %1$s</string>
    <string name="media_rewind_description">Rewind button</string>
    <string name="media_play_pause_description">Play or pause button</string>
    <string name="media_forward_description">Fast forward button</string>
   <string name="auth_getting_authorization">Getting authorization &#8230;</string>
   <string name="auth_trying_to_login">Trying to log in &#8230;</string>
   <string name="auth_no_net_conn_title">No network connection</string>
   <string name="auth_nossl_plain_ok_title">Secure connection unavailable.</string>
   <string name="auth_connection_established">Connection established</string>
   <string name="auth_testing_connection">Testing connection</string>
   <string name="auth_not_configured_title">Malformed server configuration</string>
   <string name="auth_account_not_new">An account for the same user and server already
exists in the device</string>
   <string name="auth_account_not_the_same">The entered user does not match the user of
this account</string>
   <string name="auth_unknown_error_title">Unknown error occurred!</string>
   <string name="auth_unknown_host_title">Couldn\'t find host</string>
   <string name="auth_incorrect_path_title">Server instance not found</string>
   <string name="auth_timeout_title">The server took too long to respond</string>
   <string name="auth_incorrect_address_title">Wrong server address format</string>
   <string name="auth_ssl_general_error_title">SSL initialization failed</string>
   <string name="auth_ssl_unverified_server_title">Couldn\'t verify SSL server\'s
identity</string>
   <string name="auth_bad_oc_version_title">Unrecognized server version</string>
   <string name="auth_wrong_connection_title">Couldn\'t establish connection</string>
   <string name="auth_secure_connection">Secure connection established</string>
   <string name="auth_unauthorized">Wrong username or password</string>
   <string name="auth_oauth_error">Unsuccessful authorization</string>
   <string name="auth_oauth_error_access_denied">Access denied by authorization
server</string>
   <string name="auth_wtf_reenter_URL">Unexpected state; please enter the server address
again</string>
   <string name="auth_expired_oauth_token_toast">Your authorization expired. Please,
authorize again</string>
   <string name="auth_expired_basic_auth_toast">Please enter the current
password</string>
   <string name="auth_expired_saml_sso_token_toast">Your session expired. Please connect
again</string>
   <string name="auth_connecting_auth_server">Connecting to authentication server …
</string>
   <string name="auth_unsupported_auth_method">The server does not support this
authentication method</string>
   <string name="auth_unsupported_multiaccount">%1$s does not support multiple
accounts</string>
   <string name="auth_fail_get_user_name">Your server is not returning a correct user
id, please contact an administrator
   </string>
   <string name="auth_can_not_auth_against_server">Cannot authenticate to this
server</string>
    <string name="auth_account_does_not_exist">Account does not exist in the device
yet</string>

    <string name="favorite">Set as available offline</string>
    <string name="unfavorite">Unset as available offline</string>
    <string name="common_rename">Rename</string>
    <string name="common_remove">Remove</string>
    <string name="confirmation_remove_alert">"Do you really want to remove
%1$s?"</string>
    <string name="confirmation_remove_folder_alert">"Do you really want to remove %1$s
and its contents?"</string>
    <string name="confirmation_remove_local">Local only</string>
    <string name="remove_success_msg">"Removal succeeded"</string>
```

```
    <string name="remove_fail_msg">"Removal failed"</string>
    <string name="rename_dialog_title">Enter a new name</string>
    <string name="rename_local_fail_msg">"Local copy could not be renamed; try a
different name"</string>
    <string name="rename_server_fail_msg">"Rename could not be completed"</string>
    <string name="sync_file_fail_msg">Remote file could not be checked</string>
    <string name="sync_file_nothing_to_do_msg">File contents already
synchronized</string>
    <string name="create_dir_fail_msg">Folder could not be created</string>
    <string name="filename_forbidden_characters">Forbidden characters: / \\ &lt; &gt; :
" | ? *</string>
    <string name="filename_forbidden_charaters_from_server">File name contains at least
one invalid character</string>
    <string name="filename_empty">File name cannot be empty</string>
    <string name="wait_a_moment">Wait a moment</string>
    <string name="wait_checking_credentials">Checking stored credentials</string>
    <string name="filedisplay_unexpected_bad_get_content">"Unexpected problem ; please
select the file from a different app"</string>
    <string name="filedisplay_no_file_selected">No file was selected</string>
    <string name="activity_chooser_title">Send link to &#8230;</string>
    <string name="wait_for_tmp_copy_from_private_storage">Copying file from private
storage</string>

    <string name="oauth_check_onoff">Login with oAuth2</string>
    <string name="oauth_login_connection">Connecting to oAuth2 server…</string>

    <string name="ssl_validator_header">The identity of the site could not be
verified</string>
    <string name="ssl_validator_reason_cert_not_trusted">- The server certificate is not
trusted</string>
    <string name="ssl_validator_reason_cert_expired">- The server certificate
expired</string>
    <string name="ssl_validator_reason_cert_not_yet_valid">- The server certificate
valid dates are in the future</string>
    <string name="ssl_validator_reason_hostname_not_verified">- The URL does not match
the hostname in the certificate</string>
    <string name="ssl_validator_question">Do you want to trust this certificate anyway?
</string>
    <string name="ssl_validator_not_saved">The certificate could not be saved</string>
    <string name="ssl_validator_btn_details_see">Details</string>
    <string name="ssl_validator_btn_details_hide">Hide</string>
    <string name="ssl_validator_label_subject">Issued to:</string>
    <string name="ssl_validator_label_issuer">Issued by:</string>
    <string name="ssl_validator_label_CN">Common name:</string>
    <string name="ssl_validator_label_O">Organization:</string>
    <string name="ssl_validator_label_OU">Organizational unit:</string>
    <string name="ssl_validator_label_C">Country:</string>
    <string name="ssl_validator_label_ST">State:</string>
    <string name="ssl_validator_label_L">Location:</string>
    <string name="ssl_validator_label_validity">Validity:</string>
    <string name="ssl_validator_label_validity_from">From:</string>
    <string name="ssl_validator_label_validity_to">To:</string>
    <string name="ssl_validator_label_signature">Signature:</string>
    <string name="ssl_validator_label_signature_algorithm">Algorithm:</string>
    <string name="digest_algorithm_not_available">This digest algorithm is not available
on your phone.</string>
    <string name="ssl_validator_label_certificate_fingerprint">Fingerprint:</string>
    <string name="certificate_load_problem">There is a problem loading the
certificate.</string>
    <string name="ssl_validator_null_cert">The certificate could not be shown.</string>
    <string name="ssl_validator_no_info_about_error">- No information about the
error</string>
```

```
    <string name="placeholder_sentence">This is a placeholder</string>
    <string name="placeholder_filename">placeholder.txt</string>
    <string name="placeholder_filetype">PNG Image</string>
    <string name="placeholder_filesize">389 KB</string>
    <string name="placeholder_timestamp">2012/05/18 12:23 PM</string>
    <string name="placeholder_media_time">12:23:45</string>
    <string name="instant_upload_on_wifi">Upload pictures via wifi only</string>
    <string name="instant_video_upload_on_wifi">Upload videos via wifi only</string>
    <string name="instant_upload_path">/InstantUpload</string>
    <string name="conflict_title">File conflict</string>
    <string name="conflict_message">Which files do you want to keep? If you select both
versions, the local file will have a number added to its name.</string>
    <string name="conflict_keep_both">Keep both</string>
    <string name="conflict_use_local_version">local version</string>
    <string name="conflict_use_server_version">server version</string>

    <string name="preview_image_description">Image preview</string>
    <string name="preview_image_error_unknown_format">This image cannot be
shown</string>
    <string name="error__upload__local_file_not_copied">%1$s could not be copied to %2$s
local folder</string>
    <string name="prefs_instant_upload_path_title">Upload path</string>
   <string name="share_link_no_support_share_api">Sorry, sharing is not enabled on your
server. Please contact your
       administrator.</string>
   <string name="share_link_file_no_exist">Unable to share. Please check whether the
file exists</string>
   <string name="share_link_file_error">An error occurred while trying to share this
file or folder</string>
   <string name="unshare_link_file_no_exist">Unable to unshare. Please check whether the
file exists</string>
   <string name="unshare_link_file_error">An error occurred while trying to unshare this
file or folder</string>
    <string name="update_link_file_no_exist">Unable to update. Please check whether the
file exists </string>
    <string name="update_link_file_error">An error occurred while trying to update the
share</string>
    <string name="share_link_password_title">Enter a password</string>
    <string name="share_link_empty_password">You must enter a password</string>
    <string name="activity_chooser_send_file_title">Send</string>
    <string name="copy_link">Copy link</string>
    <string name="clipboard_text_copied">Copied to clipboard</string>
    <string name="clipboard_no_text_to_copy">No text received to copy to
clipboard</string>
    <string name="clipboard_uxexpected_error">Unexpected error while copying to
clipboard</string>
    <string name="clipboard_label">Text copied from %1$s</string>
    <string name="error_cant_bind_to_operations_service">Critical error: cannot perform
operations</string>
    <string name="network_error_socket_exception">An error occurred while connecting
with the server.</string>
    <string name="network_error_socket_timeout_exception">An error occurred while
waiting for the server, the operation couldn\'t have been done</string>
    <string name="network_error_connect_timeout_exception">An error occurred while
waiting for the server, the operation couldn\'t have been done</string>
    <string name="network_host_not_available">The operation couldn\'t be completed,
server is unavailable</string>
    <string name="empty" />
    <string name="forbidden_permissions">You do not have permission %s</string>
    <string name="forbidden_permissions_rename">to rename this file</string>
    <string name="forbidden_permissions_delete">to delete this file</string>
    <string name="share_link_forbidden_permissions">to share this file</string>
```

```
    <string name="unshare_link_forbidden_permissions">to unshare this file</string>
    <string name="update_link_forbidden_permissions">to update this share</string>
    <string name="forbidden_permissions_create">to create the file</string>
    <string name="uploader_upload_forbidden_permissions">to upload in this
folder</string>
    <string name="downloader_download_file_not_found">The file is no longer available on
the server</string>
    <string name="prefs_category_accounts">Accounts</string>
    <string name="prefs_add_account">Add account</string>
    <string name="auth_redirect_non_secure_connection_title">Secure connection is
redirected through an unsecured route.</string>
   <string name="actionbar_logger">Logs</string>
   <string name="log_send_history_button">Send history</string>
   <string name="log_send_no_mail_app">No app for sending logs found. Please install a
mail app.</string>
   <string name="log_send_mail_subject">%1$s Android app logs</string>
   <string name="log_progress_dialog_text">Loading data &#8230;</string>
   <string name="saml_authentication_required_text">Authentication required</string>
   <string name="saml_authentication_wrong_pass">Wrong password</string>
   <string name="actionbar_move">Move</string>
   <string name="file_list_empty_moving">Nothing in here. You can add a folder!</string>
   <string name="folder_picker_choose_button_text">Choose</string>
    <string name="move_file_not_found">Unable to move. Please check whether the file
exists</string>
    <string name="move_file_invalid_into_descendent">It is not possible to move a folder
into a descendant</string>
    <string name="move_file_invalid_overwrite">The file exists already in the
destination folder</string>
    <string name="move_file_error">An error occurred while trying to move this file or
folder</string>
    <string name="forbidden_permissions_move">to move this file</string>
    <string name="copy_file_not_found">Unable to copy. Please check whether the file
exists</string>
    <string name="copy_file_invalid_into_descendent">It is not possible to copy a folder
into a descendant</string>
    <string name="copy_file_invalid_overwrite">The file exists already in the
destination folder</string>
    <string name="copy_file_error">An error occurred while trying to copy this file or
folder</string>
    <string name="forbidden_permissions_copy">to copy this file</string>
    <string name="prefs_category_instant_uploading">Instant uploads</string>
    <string name="prefs_category_details">Details</string>
   <string name="prefs_instant_video_upload_path_title">Upload video path</string>
    <string name="sync_folder_failed_content">Synchronization of %1$s folder could not
be completed</string>
   <string name="shared_subject_header">shared</string>
   <string name="with_you_subject_header">with you</string>

   <string name="subject_user_shared_with_you">%1$s shared \"%2$s\" with you</string>
    <string name="subject_shared_with_you">\"%1$s\" has been shared with you</string>
    <string name="auth_refresh_button">Refresh connection</string>
    <string name="auth_host_address">Server address</string>
    <string name="common_error_out_memory">Not enough memory</string>
    <string name="username">Username</string>
    <string name="file_list__footer__folder">1 folder</string>
    <string name="file_list__footer__folders">%1$d folders</string>
    <string name="file_list__footer__file">1 file</string>
    <string name="file_list__footer__file_and_folder">1 file, 1 folder</string>
    <string name="file_list__footer__file_and_folders">1 file, %1$d folders</string>
    <string name="file_list__footer__files">%1$d files</string>
    <string name="file_list__footer__files_and_folder">%1$d files, 1 folder</string>
    <string name="file_list__footer__files_and_folders">%1$d files, %2$d
```

```
folders</string>
    <string name="prefs_instant_behaviour_dialogTitle">Original file will be...</string>
    <string name="prefs_instant_behaviour_title">Original file will be...</string>
    <string name="upload_copy_files">Copy file</string>
    <string name="upload_move_files">Move file</string>
    <string name="pref_behaviour_entries_keep_file">kept in original folder</string>
    <string name="pref_behaviour_entries_move">moved to app folder</string>
    <string name="share_dialog_title">Sharing</string>
    <string name="share_file">Share %1$s</string>
    <string name="share_with_user_section_title">Share with users and groups</string>
    <string name="share_no_users">No data shared with users yet</string>
    <string name="share_add_user_or_group">Add user or group</string>
    <string name="share_via_link_section_title">Share link</string>
    <string name="share_via_link_expiration_date_label">Set expiration date</string>
    <string name="share_via_link_password_label">Password protect</string>
    <string name="share_via_link_password_title">Secured</string>
    <string name="share_via_link_edit_permission_label">Allow editing</string>
    <string name="share_get_public_link_button">Get link</string>
    <string name="share_with_title">Share with &amp;#8230;</string>
    <string name="share_with_edit_title">Share with %1$s</string>
    <string name="share_search">Search</string>
    <string name="search_users_and_groups_hint">Search users and groups</string>
    <string name="share_group_clarification">%1$s (group)</string>
    <string name="share_remote_clarification">%1$s (remote)</string>
    <string name="share_known_remote_clarification">%1$s ( at %2$s )</string>
    <string name="share_sharee_unavailable">Sorry, your server version does not allow
share with users within clients.
        \nPlease contact your administrator</string>
    <string name="share_privilege_can_share">can share</string>
    <string name="share_privilege_can_edit">can edit</string>
    <string name="share_privilege_can_edit_create">create</string>
    <string name="share_privilege_can_edit_change">change</string>
    <string name="share_privilege_can_edit_delete">delete</string>
    <string name="edit_share_unshare">Stop sharing</string>
    <string name="edit_share_done">done</string>
    <string name="action_retry_uploads">Retry failed</string>
    <string name="action_clear_failed_uploads">Clear failed</string>
    <string name="action_clear_successful_uploads">Clear successful</string>
    <string name="action_clear_finished_uploads">Clear all finished</string>
    <string name="action_switch_grid_view">Grid view</string>
    <string name="action_switch_list_view">List view</string>
    <string name="manage_space_title">Manage space</string>
    <string name="manage_space_description">Settings, database and server certificates
from %1$s\'s data will be deleted permanently. \n\nDownloaded files will be kept
untouched.\n\nThis process can take some time.</string>
    <string name="manage_space_clear_data">Clear data</string>
    <string name="manage_space_error">Some files could not be deleted.</string>
    <string name="permission_storage_access">Additional permissions required to upload
&amp;amp; download files.</string>
    <string name="local_file_not_found_toast">The file was not found in the local file
system</string>
    <string name="reconnect_drive">Remount Disk</string>
</resources>
```

11.2 android/owncloud-android-library/src/com/owncloud/android/lib/common/utils/Log_OC.java

```
package com.owncloud.android.lib.common.utils;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.text.SimpleDateFormat;
import java.util.Calendar;
import android.os.Environment;
import android.util.Log;
public class Log_OC {
    private static final String SIMPLE_DATE_FORMAT = "yyyy/MM/dd HH:mm:ss";
    private static final String LOG_FOLDER_NAME = "log";
    private static final long MAX_FILE_SIZE = 1000000; // 1MB
    private static String mOwncloudDataFolderLog = "owncloud_log";
    private static File mLogFile;
    private static File mFolder;
    private static BufferedWriter mBuf;
    private static String[] mLogFileNames = {"currentLog.txt", "olderLog.txt"};
    private static boolean isMaxFileSizeReached = false;
    private static boolean isEnabled = false;
    private static String last_action_timestamp;
    public static void setLogDataFolder(String logFolder){
       mOwncloudDataFolderLog = logFolder;
    }
    public static void i(String TAG, String message){
        Log.i(TAG, message);
        appendLog(TAG+" : "+ message);
    }
    public static void d(String TAG, String message){
        Log.d(TAG, message);
        appendLog(TAG + " : " + message);
    }
    public static void d(String TAG, String message, Exception e) {
        Log.d(TAG, message, e);
        appendLog(TAG + " : " + message + " Exception : "+ e.getStackTrace());
    }
    public static void e(String TAG, String message){
        Log.e(TAG, message);
        appendLog(TAG + " : " + message);
    }

    public static void e(String TAG, String message, Throwable e) {
        Log.e(TAG, message, e);
        appendLog(TAG+" : " + message +" Exception : " + e.getStackTrace());
    }

    public static void v(String TAG, String message){
        Log.v(TAG, message);
        appendLog(TAG+" : "+ message);
    }

    public static void w(String TAG, String message) {
        Log.w(TAG, message);
```

```
        appendLog(TAG+" : "+ message);
    }

    public static void wtf(String TAG, String message) {
        Log.wtf(TAG,message);
        appendLog(TAG+" : "+ message);
    }
    /**
     * Start doing logging
     * @param logPath : path of log file
     */
    public static void startLogging() {
       String logPath = Environment.getExternalStorageDirectory() + File.separator +
                    mOwncloudDataFolderLog + File.separator + LOG_FOLDER_NAME;
        mFolder = new File(logPath);
        mLogFile = new File(mFolder + File.separator + mLogFileNames[0]);
        boolean isFileCreated = false;
        if (!mFolder.exists()) {
            mFolder.mkdirs();
            isFileCreated = true;
            Log.d("LOG_OC", "Log file created");
        }
        try {
            // Create the current log file if does not exist
            mLogFile.createNewFile();
            mBuf = new BufferedWriter(new FileWriter(mLogFile, true));
            isEnabled = true;
            if (isFileCreated) {
                appendPhoneInfo();
            }
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            if(mBuf != null) {
                try {
                    mBuf.close();
                } catch(IOException e) {
                    e.printStackTrace();
                }
            }
        }
    }
    /**
     * Delete history logging
     */
    public static void deleteHistoryLogging() {
        File folderLogs = new File(mFolder + File.separator);
        if(folderLogs.isDirectory()){
            String[] myFiles = folderLogs.list();
            for (int i=0; i<myFiles.length; i++) {
                File myFile = new File(folderLogs, myFiles[i]);
                myFile.delete();
            }
        }
    }

    /**
     * Append the info of the device
     */
    private static void appendPhoneInfo() {
        appendLog("Model : " + android.os.Build.MODEL);
```

```
        appendLog("Brand : " + android.os.Build.BRAND);
        appendLog("Product : " + android.os.Build.PRODUCT);
        appendLog("Device : " + android.os.Build.DEVICE);
        appendLog("Version-Codename : " + android.os.Build.VERSION.CODENAME);
        appendLog("Version-Release : " + android.os.Build.VERSION.RELEASE);
    }

    /**
     * Append to the log file the info passed
     * @param text : text for adding to the log file
     */
    private static void appendLog(String text) {
        if (isEnabled) {
            if (isMaxFileSizeReached) {
                // Move current log file info to another file (old logs)
                File olderFile = new File(mFolder + File.separator + mLogFileNames[1]);
                if (mLogFile.exists()) {
                    mLogFile.renameTo(olderFile);
                }
                // Construct a new file for current log info
                mLogFile = new File(mFolder + File.separator + mLogFileNames[0]);
                isMaxFileSizeReached = false;
            }
            String timeStamp = new
SimpleDateFormat(SIMPLE_DATE_FORMAT).format(Calendar.getInstance().getTime());
            last_action_timestamp = timeStamp;
            try {
                mBuf = new BufferedWriter(new FileWriter(mLogFile, true));
                mBuf.newLine();
                mBuf.write(timeStamp);
                mBuf.newLine();
                mBuf.write(text);
                mBuf.newLine();
            } catch (IOException e) {
                e.printStackTrace();
            } finally {
                try {
                    mBuf.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
            // Check if current log file size is bigger than the max file size defined
            if (mLogFile.length() > MAX_FILE_SIZE) {
                isMaxFileSizeReached = true;
            }
        }
    }
    public static String[] getLogFileNames() {
        return mLogFileNames;
    }
    public static String getLast_action_timestamp(){return last_action_timestamp; }
}
```

11.3 android/src/com/owncloud/android/services/DiskUsageService.java

```
package com.owncloud.android.services;
import android.app.Service;
```

```
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
import android.os.IBinder;
import android.support.annotation.Nullable;
import android.support.v4.content.LocalBroadcastManager;
import android.util.Log;
import com.owncloud.android.lib.common.utils.Log_OC;
import com.owncloud.android.ui.activity.FileDisplayActivity;
import java.text.SimpleDateFormat;
import java.util.Calendar;
import java.util.Timer;
import java.util.TimerTask;
import java.util.concurrent.TimeUnit;
public class DiskUsageService  extends Service {
    private static final int IDLE_SECONDS_LIMIT = 300; // μέγιστος επιτρεπτός χρόνος
αδράνειας της εφαρμογής
    private static final int PRECISION_SECONDS = 10; //μέγιστη επιτρεπτή απόκλιση από το
πάνω όριο
    private static boolean USER_NOTIFIED = false;
    private static boolean FILE_DISPLAY_ACTIVE = true; //είναι η προβολή αρχείων ενεργή;
    private BroadcastReceiver receiver;
    private static final String TAG = DiskUsageService.class.getSimpleName();
    private LocalBroadcastManager broadcaster;
    @Nullable
    @Override
    public IBinder onBind(Intent intent) {
        return null;
    }
    public void onCreate()
    {
        broadcaster = LocalBroadcastManager.getInstance(this);
        final SimpleDateFormat format = new SimpleDateFormat("yyyy/MM/dd HH:mm:ss");
        receiver = new BroadcastReceiver() {
            @Override
            public void onReceive(Context context, Intent intent) {
                if
(intent.getAction().equals("com.filedisplayactivity.user_notified_reset")) {
                    Log.v(TAG,"User responded with remounting");
                    USER_NOTIFIED = false; //αρχικοποίηση της μεταβλητής, ο χρήστης
απάντησε με επανασύνδεση του δίσκου
                }else
if(intent.getAction().equals("com.filedisplayactivity.broadcast_received")){
                    Log.v(TAG, "User was notified on his screen");
                    USER_NOTIFIED = true; //ο χρήστης ενημερώθηκε με κατάλληλη οθόνη
για την αποσύνδεση του δίσκου και τη δυνατότητα επανασύνδεσης
                }else
if(intent.getAction().equals("com.filedisplayactivity.has_focus")){
                    Log.v(TAG,"FileDisplayActivity has just either appeared or
disappeared");
                    FILE_DISPLAY_ACTIVE = intent.getBooleanExtra("focus", false);
                }
            }
        };
// Υπολογισμός του χρόνου μεταξύ της τελευταίας δραστηριότητας (όλες οι δραστηριότητες
λήψης ή ανεβάσματος αρχείων καταγράφονται) και της παρούσης στιγμής σε τακτική βάση.
        Timer timer =new Timer();
        timer.schedule(new TimerTask(){
            @Override
            public void run() {
                long time_idle=0; // πραγματικός χρόνος αδράνειας της εφαρμογής
                try {
```

```
                    if (Log_OC.getLast_action_timestamp() != null) {
                        time_idle = Calendar.getInstance().getTime().getTime() -
format.parse(Log_OC.getLast_action_timestamp()).getTime(); // υπολογισμός πραγματικού
χρόνου αδράνειας
                    } else {
                        Log.d(TAG,"Last Action time cannot be determined");
                    }
                }
                catch (Exception e){
                    Log.e(TAG,"Could not calculate time difference");
                }
                long time_idle_in_seconds = TimeUnit.MILLISECONDS.toSeconds(time_idle);
                if (time_idle_in_seconds>IDLE_SECONDS_LIMIT){
                    if(!USER_NOTIFIED && FILE_DISPLAY_ACTIVE){
                        notifyDiskDisconnected(); //Ενημερώνουμε τον χρήστη μόνο αν
έχουν περάσει περισσότερα δευτερόλεπτα από το όριο, και δεν τον έχουμε ενημερώσει, και
έχουμε ανοικτή την προβολή αρχείων (η FileDisplayActivity είναι ενεργή) .
                    }
                }
            }
        },10,PRECISION_SECONDS*1000);
    }
    public void notifyDiskDisconnected() {
// Πληροφόρηση της FileDisplayActivity για την έλλειψη δραστηριότητας.
        final Intent intent = new
Intent(DiskUsageService.this,FileDisplayActivity.class);
        intent.setAction("com.diskusageservice.remount_action");
        broadcaster.sendBroadcast(intent);
        Log.v(TAG,"notifyDiskDisconnected has sent broadcast");
    }
    @Override
    public int onStartCommand(final Intent intent, int flags, final int startId){
        IntentFilter filter = new
IntentFilter("com.filedisplayactivity.user_notified_reset");
        filter.addAction("com.filedisplayactivity.broadcast_received");
        filter.addAction("com.filedisplayactivity.has_focus"); //Αποδεκτά intents
        LocalBroadcastManager.getInstance(this).registerReceiver((receiver),filter);
        return  START_STICKY; //Επανέναρξη της υπηρεσίας εφόσον τερματιστεί σε
κατάσταση μεγάλης έλλειψης κύριας μνήμης
    }
    @Override
    public void onDestroy(){
        LocalBroadcastManager.getInstance(this).unregisterReceiver(receiver); //
αναίρεση δήλωσης του λήπτη κατά τον τερματισμό.
    }
}
```

11.4 android/src/com/owncloud/android/ui/activity /FileDisplayActivity.java

```
/**
 *  ownCloud Android client application
 *
 *  @author Bartek Przybylski
 *  @author David A. Velasco
 *  Copyright (C) 2011  Bartek Przybylski
```

```
 *  Copyright (C) 2016 ownCloud Inc.
 *
 *  This program is free software: you can redistribute it and/or modify
 *  it under the terms of the GNU General Public License version 2,
 *  as published by the Free Software Foundation.
 *
 *  This program is distributed in the hope that it will be useful,
 *  but WITHOUT ANY WARRANTY; without even the implied warranty of
 *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 *  GNU General Public License for more details.
 *
 *  You should have received a copy of the GNU General Public License
 *  along with this program.  If not, see <http://www.gnu.org/licenses/>.
 */
package com.owncloud.android.ui.activity;
import android.Manifest;
import android.accounts.Account;
import android.accounts.AuthenticatorException;
import android.annotation.TargetApi;
import android.content.BroadcastReceiver;
import android.content.ComponentName;
import android.content.ContentResolver;
import android.content.Context;
import android.content.DialogInterface;
import android.content.Intent;
import android.content.IntentFilter;
import android.content.ServiceConnection;
import android.content.SyncRequest;
import android.content.pm.PackageManager;
import android.content.res.Resources.NotFoundException;
import android.os.Build;
import android.os.Bundle;
import android.os.IBinder;
import android.os.Parcelable;
import android.support.design.widget.Snackbar;
import android.support.v4.app.Fragment;
import android.support.v4.app.FragmentManager;
import android.support.v4.app.FragmentTransaction;
import android.support.v4.content.ContextCompat;
import android.support.v4.content.LocalBroadcastManager;
import android.support.v4.view.GravityCompat;
import android.support.v7.app.AlertDialog;
import android.util.Log;
import android.view.Menu;
import android.view.MenuInflater;
import android.view.MenuItem;
import android.view.View;
import android.widget.Button;
import android.widget.ProgressBar;
import android.widget.Toast;
import com.owncloud.android.MainApp;
import com.owncloud.android.R;
import com.owncloud.android.datamodel.FileDataStorageManager;
import com.owncloud.android.datamodel.OCFile;
import com.owncloud.android.db.PreferenceManager;
import com.owncloud.android.files.services.FileDownloader;
import com.owncloud.android.files.services.FileDownloader.FileDownloaderBinder;
import com.owncloud.android.files.services.FileUploader;
import com.owncloud.android.files.services.FileUploader.FileUploaderBinder;
import com.owncloud.android.lib.common.operations.RemoteOperation;
import com.owncloud.android.lib.common.operations.RemoteOperationResult;
import com.owncloud.android.lib.common.operations.RemoteOperationResult.ResultCode;
import com.owncloud.android.lib.common.utils.Log_OC;
import com.owncloud.android.operations.CopyFileOperation;
import com.owncloud.android.operations.CreateFolderOperation;
```

```
import com.owncloud.android.operations.MoveFileOperation;
import com.owncloud.android.operations.RefreshFolderOperation;
import com.owncloud.android.operations.RemoveFileOperation;
import com.owncloud.android.operations.RenameFileOperation;
import com.owncloud.android.operations.SynchronizeFileOperation;
import com.owncloud.android.operations.UploadFileOperation;
import com.owncloud.android.services.DiskUsageService;
import com.owncloud.android.services.MQTTService;
import com.owncloud.android.services.observer.FileObserverService;
import com.owncloud.android.syncadapter.FileSyncAdapter;
import com.owncloud.android.ui.fragment.FileDetailFragment;
import com.owncloud.android.ui.fragment.FileFragment;
import com.owncloud.android.ui.fragment.OCFileListFragment;
import com.owncloud.android.ui.fragment.TaskRetainerFragment;
import com.owncloud.android.ui.helpers.UriUploader;
import com.owncloud.android.ui.preview.PreviewImageActivity;
import com.owncloud.android.ui.preview.PreviewImageFragment;
import com.owncloud.android.ui.preview.PreviewMediaFragment;
import com.owncloud.android.ui.preview.PreviewTextFragment;
import com.owncloud.android.ui.preview.PreviewVideoActivity;
import com.owncloud.android.utils.DisplayUtils;
import com.owncloud.android.utils.ErrorMessageAdapter;
import com.owncloud.android.utils.PermissionUtil;
import java.io.File;
import java.util.ArrayList;
import java.util.Timer;
import java.util.TimerTask;
import static com.owncloud.android.db.PreferenceManager.*;
/**
 * Displays, what files the user has available in his ownCloud. This is the main view.
 */
public class FileDisplayActivity extends HookActivity
        implements FileFragment.ContainerActivity,
        OnEnforceableRefreshListener {
    private SyncBroadcastReceiver mSyncBroadcastReceiver;
    private UploadFinishReceiver mUploadFinishReceiver;
    private DownloadFinishReceiver mDownloadFinishReceiver;
    private RemoteOperationResult mLastSslUntrustedServerResult = null;
    private boolean mDualPane;
    private View mLeftFragmentContainer;
    private View mRightFragmentContainer;
    private ProgressBar mProgressBar;
    private static final String KEY_WAITING_TO_PREVIEW = "WAITING_TO_PREVIEW";
    private static final String KEY_SYNC_IN_PROGRESS = "SYNC_IN_PROGRESS";
    private static final String KEY_WAITING_TO_SEND = "WAITING_TO_SEND";
    public static final String ACTION_DETAILS =
"com.owncloud.android.ui.activity.action.DETAILS";
    public static final int REQUEST_CODE__SELECT_CONTENT_FROM_APPS =
REQUEST_CODE__LAST_SHARED + 1;
    public static final int REQUEST_CODE__SELECT_FILES_FROM_FILE_SYSTEM =
REQUEST_CODE__LAST_SHARED + 2;
    public static final int REQUEST_CODE__MOVE_FILES = REQUEST_CODE__LAST_SHARED + 3;
    public static final int REQUEST_CODE__COPY_FILES = REQUEST_CODE__LAST_SHARED + 4;
    private static final String TAG = FileDisplayActivity.class.getSimpleName();
    private static final String TAG_LIST_OF_FILES = "LIST_OF_FILES";
    private static final String TAG_SECOND_FRAGMENT = "SECOND_FRAGMENT";
    private OCFile mWaitingToPreview;
    private boolean mSyncInProgress = false;
    private OCFile mWaitingToSend;
    private static final int MAX_MQTT_RETRIES = 120;
    private int mqtt_publish_retries;
    private static boolean mBound;
    private static MQTTService mService;
    BroadcastReceiver receiver;
```

```
    LocalBroadcastManager broadcaster;
    private static Intent mqttService;
    private static Intent diskusageService;
    private boolean mqtt_message_received;
    private static String pending_message;
    public static ServiceConnection mConnection = new ServiceConnection() {
        // Περιγράφουμε τι θα γίνει όταν θα συνδεθεί η FileDisplayActivity με την
mqttService
        @Override
        public void onServiceConnected(ComponentName className,
                                       IBinder service) {
            Log.d(TAG, "Inside onServiceConnected");
            MQTTService.LocalBinder binder = (MQTTService.LocalBinder) service;
            mService = (MQTTService)binder.getService();
            mBound = true;
            // Η αποστολή μηνύματος για τη σύνδεση του δίσκου γίνεται στη μέθοδο
onResume.
        }
        // Περιγράφουμε τι θα γίνει όταν αποσυνδεθεί η δραστηριότητα από την υπηρεσία
        // Δεν αποστέλλεται μήνυμμα OFF γιατί δεν ξέρουμε αν θα εκτελεστεί η μέθοδος
        // διότι η σύνδεση δραστηριότητας – υπηρεσίας γίνεται με την παράμετρο
// Context.BIND_AUTO_CREATE
        @Override
        public void onServiceDisconnected(ComponentName arg0) {
            Log_OC.v(TAG, "MQTT Service disconnected");
            mBound = false;
        }
    };
    public static void publishMessage(String message){
        if (mBound) {
            mService.publishMessageToTopic(message); //Αν η mBound είναι αληθής, τότε η
αναφορά στην mService θα είναι έγκυρη
        }
        else {
            Log.v(TAG, "Tried to publish but mBound is false");
        }
        pending_message = message;
    }
    private Boolean getmqttmessagestatus(){return mqtt_message_received;}
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        Log_OC.v(TAG, "onCreate() start");
        broadcaster = LocalBroadcastManager.getInstance(this);
        receiver = new BroadcastReceiver() {
            @Override
            public void onReceive(Context context, Intent intent) {
                if (intent.getAction().equals("com.mqttservice.message_received")) {
                    if
(pending_message.equals(intent.getStringExtra("message_received"))) {
                        mqtt_message_received = true;
                        Log.v(TAG, "The pending message has been received");
                    } else {
                        Log.v(TAG, "A message different from the pending has been
received");
                    }
                } else if
(intent.getAction().equals("com.diskusageservice.remount_action")) {
                    publishMessage("OFF");//λάβαμε το μήνυμα για να αποσυνδέσουμε το
δίσκο οπότε τον αποσυνδέουμε και δείχνουμε την οθόνη με πληροφορίες για την επανασύνδεση
                    setContentView(R.layout.hard_disk_reconnect);
                    Log.d(TAG, "message from notifyDiskDisconnected received - disk
unmounted");
```

```
                    final Intent reply_intent = new Intent(FileDisplayActivity.this,
DiskUsageService.class);

reply_intent.setAction("com.filedisplayactivity.broadcast_received");
                    broadcaster.sendBroadcast(reply_intent); //αποστολή μηνύματος στη
DiskUsageService, ότι ο χρήστης είναι ενήμερος για τη παρέλευση μεγάλου χρονικού
διαστήματος αδρανείας του δίσκου
                    Button RemountButton = (Button) findViewById(R.id.remount_disk);
                    RemountButton.setOnClickListener(new View.OnClickListener() {
                        @Override
                        public void onClick(View v) {
                            // TODO Auto-generated method stub
                            Log.d(TAG, "Remount Button Clicked");
                            stopService(diskusageService);
                            stopService(mqttService);
                            unbindService(mConnection);
                            mBound = false; //η onServiceDisconnected καλείται μόνο
όταν η mBound τερματίζει απρόοπτα, επομένως δεν μπορούμε να μεταφέρουμε εκεί αυτή την
εντολή.

reply_intent.setAction("com.filedisplayactivity.user_notified_reset");
                            broadcaster.sendBroadcast(reply_intent);
                            FileDisplayActivity.this.recreate(); // Επανέναρξη της
δραστηριότητας και αποστολή του μηνύματος ON για επανασύνδεση του δίσκου κατά τη κλήση
της onResume.
                        }
                    });
                }
            }
        };
        //Δημιουργία της υπηρεσίας mqttService, τύπου MQTTService
        mqttService = new Intent(this, MQTTService.class);
        startService(mqttService);
        //Δημιουργία της υπηρεσίας diskusageService, τύπου MQTTService
        Log.v(TAG, "Disk Service Starting");
        diskusageService = new Intent(this, DiskUsageService.class);
        startService(diskusageService);
        Log.v(TAG, "Disk Service Started");
        super.onCreate(savedInstanceState); // this calls onAccountChanged() when
ownCloud Account
        // is valid
        /// grant that FileObserverService is watching favorite files
        if (savedInstanceState == null) {
            Intent initObserversIntent = FileObserverService.makeInitIntent(this);
            startService(initObserversIntent);
        }
        /// Load of saved instance state
        if (savedInstanceState != null) {
            mWaitingToPreview = (OCFile) savedInstanceState.getParcelable(
                    FileDisplayActivity.KEY_WAITING_TO_PREVIEW);
            mSyncInProgress = savedInstanceState.getBoolean(KEY_SYNC_IN_PROGRESS);
            mWaitingToSend = (OCFile) savedInstanceState.getParcelable(
                    FileDisplayActivity.KEY_WAITING_TO_SEND);
        } else {
            mWaitingToPreview = null;
            mSyncInProgress = false;
            mWaitingToSend = null;
        }
        /// USER INTERFACE
        // Inflate and set the layout view
        setContentView(R.layout.files);
        // Navigation Drawer
```

```
        initDrawer();
        mProgressBar = (ProgressBar) findViewById(R.id.progressBar);
        mProgressBar.setIndeterminateDrawable(
                ContextCompat.getDrawable(this,
                        R.drawable.actionbar_progress_indeterminate_horizontal));
        mDualPane = getResources().getBoolean(R.bool.large_land_layout);
        mLeftFragmentContainer = findViewById(R.id.left_fragment_container);
        mRightFragmentContainer = findViewById(R.id.right_fragment_container);
        // Action bar setup
        getSupportActionBar().setHomeButtonEnabled(true);       // mandatory since
Android ICS,
        // according to the official
        // documentation
        // enable ActionBar app icon to behave as action to toggle nav drawer
        //getSupportActionBar().setDisplayHomeAsUpEnabled(true);
        getSupportActionBar().setHomeButtonEnabled(true);
        // Init Fragment without UI to retain AsyncTask across configuration changes
        FragmentManager fm = getSupportFragmentManager();
        TaskRetainerFragment taskRetainerFragment =
                (TaskRetainerFragment)
fm.findFragmentByTag(TaskRetainerFragment.FTAG_TASK_RETAINER_FRAGMENT);
        if (taskRetainerFragment == null) {
            taskRetainerFragment = new TaskRetainerFragment();
            fm.beginTransaction()
                    .add(taskRetainerFragment,
TaskRetainerFragment.FTAG_TASK_RETAINER_FRAGMENT).commit();
        }   // else, Fragment already created and retained across configuration change
        Log_OC.v(TAG, "onCreate() end");
    }
    @Override
    protected void onPostCreate(Bundle savedInstanceState) {
        super.onPostCreate(savedInstanceState);
        if (PermissionUtil.checkSelfPermission(this,
Manifest.permission.WRITE_EXTERNAL_STORAGE)) {
            // Check if we should show an explanation
            if (PermissionUtil.shouldShowRequestPermissionRationale(this,
                    Manifest.permission.WRITE_EXTERNAL_STORAGE)) {
                // Show explanation to the user and then request permission
                Snackbar snackbar = Snackbar.make(findViewById(R.id.ListLayout),
R.string.permission_storage_access,
                        Snackbar.LENGTH_INDEFINITE)
                        .setAction(R.string.common_ok, new View.OnClickListener() {
                            @Override
                            public void onClick(View v) {

PermissionUtil.requestWriteExternalStoreagePermission(FileDisplayActivity.this);
                            }
                        });
                DisplayUtils.colorSnackbar(this, snackbar);
                snackbar.show();
            } else {
                // No explanation needed, request the permission.
                PermissionUtil.requestWriteExternalStoreagePermission(this);
            }
        }
        if (savedInstanceState == null) {
            createMinFragments();
        }
        mProgressBar.setIndeterminate(mSyncInProgress);
        // always AFTER setContentView(...) in onCreate(); to work around bug in its
implementation
        setBackgroundText();
```

```
    }
    @Override
    public void onRequestPermissionsResult(int requestCode,
                                           String permissions[], int[] grantResults)
{
        switch (requestCode) {
            case PermissionUtil.PERMISSIONS_WRITE_EXTERNAL_STORAGE: {
                // If request is cancelled, result arrays are empty.
                if (grantResults.length > 0
                        && grantResults[0] == PackageManager.PERMISSION_GRANTED) {
                    // permission was granted
                    startSynchronization();
                    // toggle on is save since this is the only scenario this code
gets accessed
                } else {
                    // permission denied --> do nothing
                }
                return;
            }
        }
    }
    @Override
    protected void onStart() {
        Log_OC.v(TAG, "onStart() start");
        IntentFilter actions = new IntentFilter("com.diskusageservice.remount_action");
        actions.addAction("com.mqttservice.message_received");
        LocalBroadcastManager.getInstance(this).registerReceiver((receiver),actions);
        super.onStart();
        Log_OC.v(TAG, "onStart() end");
    }
    @Override
    protected void onStop() {
        Log_OC.v(TAG, "onStop() start");
        LocalBroadcastManager.getInstance(this).unregisterReceiver(receiver);
        super.onStop();
        Log_OC.v(TAG, "onStop() end");
    }
    @Override
    protected void onDestroy() {
        Log_OC.v(TAG, "onDestroy() start");
        super.onDestroy();
        Log_OC.v(TAG, "onDestroy() end");
    }
    /**
     * Called when the ownCloud {@link Account} associated to the Activity was just
updated.
     */
    @Override
    protected void onAccountSet(boolean stateWasRecovered) {
        super.onAccountSet(stateWasRecovered);
        if (getAccount() != null) {
            /// Check whether the 'main' OCFile handled by the Activity is contained in
the
            // current Account
            OCFile file = getFile();
            // get parent from path
            String parentPath = "";
            if (file != null) {
                if (file.isDown() && file.getLastSyncDateForProperties() == 0) {
                    // upload in progress - right now, files are not inserted in the
local
                    // cache until the upload is successful get parent from path
```

```
                    parentPath = file.getRemotePath().substring(0,
                            file.getRemotePath().lastIndexOf(file.getFileName()));
                    if (getStorageManager().getFileByPath(parentPath) == null)
                        file = null; // not able to know the directory where the file
is uploading
                } else {
                    file = getStorageManager().getFileByPath(file.getRemotePath());
                    // currentDir = null if not in the current Account
                }
            }
            if (file == null) {
                // fall back to root folder
                file = getStorageManager().getFileByPath(OCFile.ROOT_PATH);  // never
returns null
            }
            setFile(file);
            if (mAccountWasSet) {
                setUsernameInDrawer(findViewById(R.id.left_drawer), getAccount());
            }
            if (!stateWasRecovered) {
                Log_OC.d(TAG, "Initializing Fragments in onAccountChanged..");
                initFragmentsWithFile();
                if (file.isFolder()) {
                    startSyncFolderOperation(file, false);
                }
            } else {
                updateFragmentsVisibility(!file.isFolder());
                updateActionBarTitleAndHomeButton(file.isFolder() ? null : file);
            }
        }
    }
    private void createMinFragments() {
        OCFileListFragment listOfFiles = new OCFileListFragment();
        FragmentTransaction transaction =
getSupportFragmentManager().beginTransaction();
        transaction.add(R.id.left_fragment_container, listOfFiles, TAG_LIST_OF_FILES);
        transaction.commit();
    }
    private void initFragmentsWithFile() {
        if (getAccount() != null && getFile() != null) {
            /// First fragment
            OCFileListFragment listOfFiles = getListOfFilesFragment();
            if (listOfFiles != null) {
                listOfFiles.listDirectory(getCurrentDir());
                // TODO Enable when "On Device" is recovered
                // listOfFiles.listDirectory(getCurrentDir(),
MainApp.getOnlyOnDevice());
            } else {
                Log_OC.e(TAG, "Still have a chance to lose the initializacion of list
fragment >(");
            }
            /// Second fragment
            OCFile file = getFile();
            Fragment secondFragment = chooseInitialSecondFragment(file);
            if (secondFragment != null) {
                setSecondFragment(secondFragment);
                updateFragmentsVisibility(true);
                updateActionBarTitleAndHomeButton(file);
            } else {
                cleanSecondFragment();
                if (file.isDown() && PreviewTextFragment.canBePreviewed(file))
                    startTextPreview(file);
```

```
            }
        } else {
            Log_OC.wtf(TAG, "initFragments() called with invalid NULLs!");
            if (getAccount() == null) {
                Log_OC.wtf(TAG, "\t account is NULL");
            }
            if (getFile() == null) {
                Log_OC.wtf(TAG, "\t file is NULL");
            }
        }
    }
    private Fragment chooseInitialSecondFragment(OCFile file) {
        Fragment secondFragment = null;
        if (file != null && !file.isFolder()) {
            if (file.isDown() && PreviewMediaFragment.canBePreviewed(file)
                    && file.getLastSyncDateForProperties() > 0  // temporal fix
                    ) {
                int startPlaybackPosition =

getIntent().getIntExtra(PreviewVideoActivity.EXTRA_START_POSITION, 0);
                boolean autoplay =

getIntent().getBooleanExtra(PreviewVideoActivity.EXTRA_AUTOPLAY, true);
                secondFragment = new PreviewMediaFragment(file, getAccount(),
                        startPlaybackPosition, autoplay);
            } else if (file.isDown() && PreviewTextFragment.canBePreviewed(file)) {
                secondFragment = null;
            } else {
                secondFragment = FileDetailFragment.newInstance(file, getAccount());
            }
        }
        return secondFragment;
    }
    /**
     * Replaces the second fragment managed by the activity with the received as
     * a parameter.
     * <p/>
     * Assumes never will be more than two fragments managed at the same time.
     *
     * @param fragment New second Fragment to set.
     */
    private void setSecondFragment(Fragment fragment) {
        FragmentTransaction transaction =
getSupportFragmentManager().beginTransaction();
        transaction.replace(R.id.right_fragment_container, fragment,
TAG_SECOND_FRAGMENT);
        transaction.commit();
    }
    private void updateFragmentsVisibility(boolean existsSecondFragment) {
        if (mDualPane) {
            if (mLeftFragmentContainer.getVisibility() != View.VISIBLE) {
                mLeftFragmentContainer.setVisibility(View.VISIBLE);
            }
            if (mRightFragmentContainer.getVisibility() != View.VISIBLE) {
                mRightFragmentContainer.setVisibility(View.VISIBLE);
            }
        } else if (existsSecondFragment) {
            if (mLeftFragmentContainer.getVisibility() != View.GONE) {
                mLeftFragmentContainer.setVisibility(View.GONE);
            }
            if (mRightFragmentContainer.getVisibility() != View.VISIBLE) {
```

```
                    mRightFragmentContainer.setVisibility(View.VISIBLE);
                }
            } else {
                if (mLeftFragmentContainer.getVisibility() != View.VISIBLE) {
                    mLeftFragmentContainer.setVisibility(View.VISIBLE);
                }
                if (mRightFragmentContainer.getVisibility() != View.GONE) {
                    mRightFragmentContainer.setVisibility(View.GONE);
                }
            }
        }
        private OCFileListFragment getListOfFilesFragment() {
            Fragment listOfFiles = getSupportFragmentManager().findFragmentByTag(
                    FileDisplayActivity.TAG_LIST_OF_FILES);
            if (listOfFiles != null) {
                return (OCFileListFragment) listOfFiles;
            }
            Log_OC.wtf(TAG, "Access to unexisting list of files fragment!!");
            return null;
        }
        public FileFragment getSecondFragment() {
            Fragment second = getSupportFragmentManager().findFragmentByTag(
                    FileDisplayActivity.TAG_SECOND_FRAGMENT);
            if (second != null) {
                return (FileFragment) second;
            }
            return null;
        }
        protected void cleanSecondFragment() {
            Fragment second = getSecondFragment();
            if (second != null) {
                FragmentTransaction tr = getSupportFragmentManager().beginTransaction();
                tr.remove(second);
                tr.commit();
            }
            updateFragmentsVisibility(false);
            updateActionBarTitleAndHomeButton(null);
        }
        protected void refreshListOfFilesFragment() {
            OCFileListFragment fileListFragment = getListOfFilesFragment();
            if (fileListFragment != null) {
                fileListFragment.listDirectory();
                // TODO Enable when "On Device" is recovered ?
                // fileListFragment.listDirectory(MainApp.getOnlyOnDevice());
            }
        }
        protected void refreshSecondFragment(String downloadEvent, String
downloadedRemotePath,
                                             boolean success) {
            FileFragment secondFragment = getSecondFragment();
            boolean waitedPreview = (mWaitingToPreview != null &&
                    mWaitingToPreview.getRemotePath().equals(downloadedRemotePath));
            if (secondFragment != null && secondFragment instanceof FileDetailFragment) {
                FileDetailFragment detailsFragment = (FileDetailFragment) secondFragment;
                OCFile fileInFragment = detailsFragment.getFile();
                if (fileInFragment != null &&
                        !downloadedRemotePath.equals(fileInFragment.getRemotePath())) {
                    // the user browsed to other file ; forget the automatic preview
                    mWaitingToPreview = null;
                } else if (downloadEvent.equals(FileDownloader.getDownloadAddedMessage()))
{
```

```
                    // grant that the right panel updates the progress bar
                    detailsFragment.listenForTransferProgress();
                    detailsFragment.updateFileDetails(true, false);
               } else if (downloadEvent.equals(FileDownloader.getDownloadFinishMessage()))
{
                    //  update the right panel
                    boolean detailsFragmentChanged = false;
                    if (waitedPreview) {
                         if (success) {
                              mWaitingToPreview = getStorageManager().getFileById(
                                        mWaitingToPreview.getFileId());   // update the file
from database,
                              // for the local storage path
                              if (PreviewMediaFragment.canBePreviewed(mWaitingToPreview)) {
                                   startMediaPreview(mWaitingToPreview, 0, true);
                                   detailsFragmentChanged = true;
                              } else if
(PreviewTextFragment.canBePreviewed(mWaitingToPreview)) {
                                   startTextPreview(mWaitingToPreview);
                                   detailsFragmentChanged = true;
                              } else {
                                   getFileOperationsHelper().openFile(mWaitingToPreview);
                              }
                         }
                         mWaitingToPreview = null;
                    }
                    if (!detailsFragmentChanged) {
                         detailsFragment.updateFileDetails(false, (success));
                    }
               }
          }
     }
     @Override
     public boolean onPrepareOptionsMenu(Menu menu) {
          boolean drawerOpen = mDrawerLayout.isDrawerOpen(GravityCompat.START);
          menu.findItem(R.id.action_sort).setVisible(!drawerOpen);
          menu.findItem(R.id.action_sync_account).setVisible(!drawerOpen);
          menu.findItem(R.id.action_switch_view).setVisible(!drawerOpen);
          return super.onPrepareOptionsMenu(menu);
     }
     @Override
     public boolean onCreateOptionsMenu(Menu menu) {
          MenuInflater inflater = getMenuInflater();
          inflater.inflate(R.menu.main_menu, menu);
          menu.findItem(R.id.action_create_dir).setVisible(false);
          return true;
     }
     @Override
     public boolean onOptionsItemSelected(MenuItem item) {
          boolean retval = true;
          switch (item.getItemId()) {
               case R.id.action_sync_account: {
                    startSynchronization();
                    break;
               }
               case android.R.id.home: {
                    FileFragment second = getSecondFragment();
                    OCFile currentDir = getCurrentDir();
                    if (mDrawerLayout.isDrawerOpen(GravityCompat.START)) {
                         mDrawerLayout.closeDrawer(GravityCompat.START);
                    } else if ((currentDir != null && currentDir.getParentId() != 0) ||
```

```
                        (second != null && second.getFile() != null)) {
                    onBackPressed();
                } else {
                    mDrawerLayout.openDrawer(GravityCompat.START);
                }
                break;
            }
            case R.id.action_sort: {
                Integer sortOrder = getSortOrder(this);
                AlertDialog.Builder builder = new AlertDialog.Builder(this);
                builder.setTitle(R.string.actionbar_sort_title)
                        .setSingleChoiceItems(R.array.actionbar_sortby, sortOrder,
                                new DialogInterface.OnClickListener() {
                                    public void onClick(DialogInterface dialog, int
which) {
                                        switch (which) {
                                            case 0:
                                                sortByName(true);
                                                break;
                                            case 1:
                                                sortByDate(false);
                                                break;
                                        }
                                        dialog.dismiss();
                                    }
                                });
                builder.create().show();
                break;
            }
            case R.id.action_switch_view: {
                if (isGridView()) {
                    item.setTitle(getString(R.string.action_switch_grid_view));
                    item.setIcon(ContextCompat.getDrawable(getApplicationContext(),
                            R.drawable.ic_view_module));
                    getListOfFilesFragment().setListAsPreferred();
                } else {

item.setTitle(getApplicationContext().getString(R.string.action_switch_list_view));
                    item.setIcon(ContextCompat.getDrawable(getApplicationContext(),
                            R.drawable.ic_view_list));
                    getListOfFilesFragment().setGridAsPreferred();
                }
                return true;
            }
            default:
                retval = super.onOptionsItemSelected(item);
        }
        return retval;
    }
    private void startSynchronization() {
        Log_OC.d(TAG, "Got to start sync");
        if (android.os.Build.VERSION.SDK_INT < android.os.Build.VERSION_CODES.KITKAT) {
            Log_OC.d(TAG, "Canceling all syncs for " + MainApp.getAuthority());
            ContentResolver.cancelSync(null, MainApp.getAuthority());
            // cancel the current synchronizations of any ownCloud account
            Bundle bundle = new Bundle();
            bundle.putBoolean(ContentResolver.SYNC_EXTRAS_MANUAL, true);
            bundle.putBoolean(ContentResolver.SYNC_EXTRAS_EXPEDITED, true);
            Log_OC.d(TAG, "Requesting sync for " + getAccount().name + " at " +
                    MainApp.getAuthority());
            ContentResolver.requestSync(
```

```
                    getAccount(),
                    MainApp.getAuthority(), bundle);
        } else {
            Log_OC.d(TAG, "Requesting sync for " + getAccount().name + " at " +
                    MainApp.getAuthority() + " with new API");
            SyncRequest.Builder builder = new SyncRequest.Builder();
            builder.setSyncAdapter(getAccount(), MainApp.getAuthority());
            builder.setExpedited(true);
            builder.setManual(true);
            builder.syncOnce();
            // Fix bug in Android Lollipop when you click on refresh the whole account
            Bundle extras = new Bundle();
            builder.setExtras(extras);
            SyncRequest request = builder.build();
            ContentResolver.requestSync(request);
        }
    }
    /**
     * Called, when the user selected something for uploading
     */
    @TargetApi(Build.VERSION_CODES.JELLY_BEAN)
    @Override
    protected void onActivityResult(int requestCode, int resultCode, Intent data) {
        if (requestCode == REQUEST_CODE__SELECT_CONTENT_FROM_APPS &&
            (resultCode == RESULT_OK || resultCode ==
UploadFilesActivity.RESULT_OK_AND_MOVE)) {
            requestUploadOfContentFromApps(data, resultCode);
        } else if (requestCode == REQUEST_CODE__SELECT_FILES_FROM_FILE_SYSTEM &&
            (resultCode == RESULT_OK || resultCode ==
UploadFilesActivity.RESULT_OK_AND_MOVE)) {
            requestUploadOfFilesFromFileSystem(data, resultCode);
        } else if (requestCode == REQUEST_CODE__MOVE_FILES && resultCode == RESULT_OK) {
            final Intent fData = data;
            final int fResultCode = resultCode;
            getHandler().postDelayed(
                    new Runnable() {
                        @Override
                        public void run() {
                            requestMoveOperation(fData, fResultCode);
                        }
                    },
                    DELAY_TO_REQUEST_OPERATIONS_LATER
            );
        } else if (requestCode == REQUEST_CODE__COPY_FILES && resultCode == RESULT_OK) {
            final Intent fData = data;
            final int fResultCode = resultCode;
            getHandler().postDelayed(
                    new Runnable() {
                        @Override
                        public void run() {
                            requestCopyOperation(fData, fResultCode);
                        }
                    },
                    DELAY_TO_REQUEST_OPERATIONS_LATER
            );
        } else {
            super.onActivityResult(requestCode, resultCode, data);
        }
    }
    private void requestUploadOfFilesFromFileSystem(Intent data, int resultCode) {
        String[] filePaths =
```

```
data.getStringArrayExtra(UploadFilesActivity.EXTRA_CHOSEN_FILES);
        if (filePaths != null) {
            String[] remotePaths = new String[filePaths.length];
            String remotePathBase = getCurrentDir().getRemotePath();
            for (int j = 0; j < remotePaths.length; j++) {
                remotePaths[j] = remotePathBase + (new File(filePaths[j])).getName();
            }
            int behaviour = (resultCode == UploadFilesActivity.RESULT_OK_AND_MOVE) ?
FileUploader
                    .LOCAL_BEHAVIOUR_MOVE : FileUploader.LOCAL_BEHAVIOUR_COPY;
            FileUploader.UploadRequester requester = new
FileUploader.UploadRequester();
            requester.uploadNewFile(
                    this,
                    getAccount(),
                    filePaths,
                    remotePaths,
                    null,           // MIME type will be detected from file name
                    behaviour,
                    false,          // do not create parent folder if not existent
                    UploadFileOperation.CREATED_BY_USER
            );
        } else {
            Log_OC.d(TAG, "User clicked on 'Update' with no selection");
            Toast t = Toast.makeText(this,
getString(R.string.filedisplay_no_file_selected),
                    Toast.LENGTH_LONG);
            t.show();
            return;
        }
    }
    private void requestUploadOfContentFromApps(Intent contentIntent, int resultCode) {
        ArrayList<Parcelable> streamsToUpload = new ArrayList<>();
        //getClipData is only supported on api level 16+, Jelly Bean
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.JELLY_BEAN &&
            contentIntent.getClipData() != null &&
            contentIntent.getClipData().getItemCount() > 0) {
            for (int i = 0; i < contentIntent.getClipData().getItemCount(); i++) {
                streamsToUpload.add(contentIntent.getClipData().getItemAt(i).getUri());
            }
        } else {
            streamsToUpload.add(contentIntent.getData());
        }
        int behaviour = (resultCode == UploadFilesActivity.RESULT_OK_AND_MOVE) ?
FileUploader.LOCAL_BEHAVIOUR_MOVE :
                FileUploader.LOCAL_BEHAVIOUR_COPY;
        OCFile currentDir = getCurrentDir();
        String remotePath = (currentDir != null) ? currentDir.getRemotePath() :
OCFile.ROOT_PATH;
        UriUploader uploader = new UriUploader(
                this,
                streamsToUpload,
                remotePath,
                getAccount(),
                behaviour,
                false, // Not show waiting dialog while file is being copied from
private storage
                null  // Not needed copy temp task listener
        );
        uploader.uploadUris();
    }
    /**
```

```
     * Request the operation for moving the file/folder from one path to another
     *
     * @param data       Intent received
     * @param resultCode Result code received
     */
    private void requestMoveOperation(Intent data, int resultCode) {
        OCFile folderToMoveAt = (OCFile)
data.getParcelableExtra(FolderPickerActivity.EXTRA_FOLDER);
        OCFile targetFile = (OCFile)
data.getParcelableExtra(FolderPickerActivity.EXTRA_FILE);
        getFileOperationsHelper().moveFile(folderToMoveAt, targetFile);
    }
    /**
     * Request the operation for copying the file/folder from one path to another
     *
     * @param data       Intent received
     * @param resultCode Result code received
     */
    private void requestCopyOperation(Intent data, int resultCode) {
        OCFile folderToMoveAt =
data.getParcelableExtra(FolderPickerActivity.EXTRA_FOLDER);
        OCFile targetFile = data.getParcelableExtra(FolderPickerActivity.EXTRA_FILE);
        getFileOperationsHelper().copyFile(folderToMoveAt, targetFile);
    }
    @Override
    public void onBackPressed() {
        boolean isFabOpen = isFabOpen();
        boolean isDrawerOpen = isDrawerOpen();
        /*
         * BackPressed priority/hierarchy:
         *    1. close drawer if opened
         *    2. close FAB if open (only if drawer isn't open)
         *    3. navigate up (only if drawer and FAB aren't open)
         */
        if(isDrawerOpen && isFabOpen) {
            // close drawer first
            super.onBackPressed();
        } else if(isDrawerOpen && !isFabOpen) {
            // close drawer
            super.onBackPressed();
        } else if (!isDrawerOpen && isFabOpen) {
            // close fab
            getListOfFilesFragment().getFabMain().collapse();
        } else {
            // all closed
            OCFileListFragment listOfFiles = getListOfFilesFragment();
            if (mDualPane || getSecondFragment() == null) {
                OCFile currentDir = getCurrentDir();
                if (currentDir == null || currentDir.getParentId() ==
FileDataStorageManager.ROOT_PARENT_ID) {
                    finish();
                    return;
                }
                if (listOfFiles != null) {  // should never be null, indeed
                    listOfFiles.onBrowseUp();
                }
            }
            if (listOfFiles != null) {  // should never be null, indeed
                setFile(listOfFiles.getCurrentFile());
            }
            cleanSecondFragment();
```

```
        }
    }
    @Override
    protected void onSaveInstanceState(Bundle outState) {
        // responsibility of restore is preferred in onCreate() before than in
        // onRestoreInstanceState when there are Fragments involved
        Log_OC.v(TAG, "onSaveInstanceState() start");
        super.onSaveInstanceState(outState);
        outState.putParcelable(FileDisplayActivity.KEY_WAITING_TO_PREVIEW,
mWaitingToPreview);
        outState.putBoolean(FileDisplayActivity.KEY_SYNC_IN_PROGRESS, mSyncInProgress);
        //outState.putBoolean(FileDisplayActivity.KEY_REFRESH_SHARES_IN_PROGRESS,
        // mRefreshSharesInProgress);
        outState.putParcelable(FileDisplayActivity.KEY_WAITING_TO_SEND, mWaitingToSend);
        Log_OC.v(TAG, "onSaveInstanceState() end");
    }
    @Override
    protected void onResume() {
        Log_OC.v(TAG, "onResume() start");
        Intent focus_change = new Intent(this, DiskUsageService.class);
        focus_change.setAction("com.filedisplayactivity.has_focus");
        focus_change.putExtra("focus", true);
        if (!mBound) {
            bindService(mqttService, mConnection, Context.BIND_AUTO_CREATE);
        }
        publishMessage("ON"); //αποστολή μηνύματος στο σκληρό δίσκο
        Log_OC.v(TAG, "OnResume published ON message, but did it arrive?");
        mqtt_message_received = false;
        mqtt_publish_retries = 0;
        final Timer timer = new Timer();
//Αποστολή μηνυμάτων με αρχική καθυστέρηση 1000ms, κάθε 500ms, μέχρις ότου συνδεθεί
//ο δίσκος.
        timer.schedule(new TimerTask() {
            @Override
            public void run() {
                if (!getmqttmessagestatus()) {
                    if (mqtt_publish_retries < MAX_MQTT_RETRIES) {
                        publishMessage("ON");
                        mqtt_publish_retries++;
                    }
                } else {
                    Log.v(TAG, "Succeeded by persisting " + mqtt_publish_retries + "
times");
                    timer.cancel();
                }
            }
        }, 1000, 500);
        super.onResume();
        // refresh Navigation Drawer account list
        mNavigationDrawerAdapter.updateAccountList();
        // refresh list of files
        refreshListOfFilesFragment();
        // Listen for sync messages
        IntentFilter syncIntentFilter = new
IntentFilter(FileSyncAdapter.EVENT_FULL_SYNC_START);
        syncIntentFilter.addAction(FileSyncAdapter.EVENT_FULL_SYNC_END);

syncIntentFilter.addAction(FileSyncAdapter.EVENT_FULL_SYNC_FOLDER_CONTENTS_SYNCED);

syncIntentFilter.addAction(RefreshFolderOperation.EVENT_SINGLE_FOLDER_CONTENTS_SYNCED);

syncIntentFilter.addAction(RefreshFolderOperation.EVENT_SINGLE_FOLDER_SHARES_SYNCED);
```

```
        mSyncBroadcastReceiver = new SyncBroadcastReceiver();
        registerReceiver(mSyncBroadcastReceiver, syncIntentFilter);

//LocalBroadcastManager.getInstance(this).registerReceiver(mSyncBroadcastReceiver,
        // syncIntentFilter);
        // Listen for upload messages
        IntentFilter uploadIntentFilter = new
IntentFilter(FileUploader.getUploadFinishMessage());
        mUploadFinishReceiver = new UploadFinishReceiver();
        registerReceiver(mUploadFinishReceiver, uploadIntentFilter);
        // Listen for download messages
        IntentFilter downloadIntentFilter = new IntentFilter(
                FileDownloader.getDownloadAddedMessage());
        downloadIntentFilter.addAction(FileDownloader.getDownloadFinishMessage());
        mDownloadFinishReceiver = new DownloadFinishReceiver();
        registerReceiver(mDownloadFinishReceiver, downloadIntentFilter);
        Log_OC.v(TAG, "onResume() end");
    }
    @Override
    protected void onPause() {
        Log_OC.v(TAG, "onPause() start");
        publishMessage("OFF");    //αποσύνδεση του δίσκου
        if (mBound) {
            unbindService(mConnection); //αποσύνδεση της υπηρεσίας
            mBound = false;
        }
        Log_OC.v(TAG, "Off message from onPause - Notifying of change of visibility
status");
        Intent focus_change = new Intent(this,DiskUsageService.class);
        focus_change.setAction("com.filedisplayactivity.has_focus");
        focus_change.putExtra("focus",false);
        broadcaster.sendBroadcast(focus_change); //πληροφόρηση ότι η
FileDisplayActivity είναι στο παρασκήνιο
        if (mSyncBroadcastReceiver != null) {
            unregisterReceiver(mSyncBroadcastReceiver);

//LocalBroadcastManager.getInstance(this).unregisterReceiver(mSyncBroadcastReceiver);
            mSyncBroadcastReceiver = null;
        }
        if (mUploadFinishReceiver != null) {
            unregisterReceiver(mUploadFinishReceiver);
            mUploadFinishReceiver = null;
        }
        if (mDownloadFinishReceiver != null) {
            unregisterReceiver(mDownloadFinishReceiver);
            mDownloadFinishReceiver = null;
        }
        super.onPause();
        Log_OC.v(TAG, "onPause() end");
    }
    public boolean isFabOpen() {
        if(getListOfFilesFragment() != null
                && getListOfFilesFragment().getFabMain() != null
                && getListOfFilesFragment().getFabMain().isExpanded()) {
            return true;
        } else {
            return false;
        }
    }
    private class SyncBroadcastReceiver extends BroadcastReceiver {
        /**
         * {@link BroadcastReceiver} to enable syncing feedback in UI
```

```
         */
        @Override
        public void onReceive(Context context, Intent intent) {
            try {
                String event = intent.getAction();
                Log_OC.d(TAG, "Received broadcast " + event);
                String accountName =
intent.getStringExtra(FileSyncAdapter.EXTRA_ACCOUNT_NAME);
                String synchFolderRemotePath =
                        intent.getStringExtra(FileSyncAdapter.EXTRA_FOLDER_PATH);
                RemoteOperationResult synchResult =
                        (RemoteOperationResult) intent.getSerializableExtra(
                                FileSyncAdapter.EXTRA_RESULT);
                boolean sameAccount = (getAccount() != null &&
                        accountName.equals(getAccount().name) && getStorageManager() !
= null);
                if (sameAccount) {
                    if (FileSyncAdapter.EVENT_FULL_SYNC_START.equals(event)) {
                        mSyncInProgress = true;
                    } else {
                        OCFile currentFile = (getFile() == null) ? null :

getStorageManager().getFileByPath(getFile().getRemotePath());
                        OCFile currentDir = (getCurrentDir() == null) ? null :

getStorageManager().getFileByPath(getCurrentDir().getRemotePath());
                        if (currentDir == null) {
                            // current folder was removed from the server
                            Toast.makeText(FileDisplayActivity.this,
                                    String.format(
                                            getString(R.string.

sync_current_folder_was_removed),
                                            synchFolderRemotePath),
                                    Toast.LENGTH_LONG)
                                    .show();
                            browseToRoot();
                        } else {
                            if (currentFile == null && !getFile().isFolder()) {
                                // currently selected file was removed in the server,
and now we
                                // know it
                                cleanSecondFragment();
                                currentFile = currentDir;
                            }
                            if (synchFolderRemotePath != null &&

currentDir.getRemotePath().equals(synchFolderRemotePath)) {
                                OCFileListFragment fileListFragment =
getListOfFilesFragment();
                                if (fileListFragment != null) {
                                    fileListFragment.listDirectory();
                                    // TODO Enable when "On Device" is recovered ?
                                    // fileListFragment.listDirectory(currentDir,
                                    // MainApp.getOnlyOnDevice());
                                }
                            }
                            setFile(currentFile);
                        }
                        mSyncInProgress = (!
FileSyncAdapter.EVENT_FULL_SYNC_END.equals(event) &&
                                !
```

```
RefreshFolderOperation.EVENT_SINGLE_FOLDER_SHARES_SYNCED
                                                .equals(event));
                            if
(RefreshFolderOperation.EVENT_SINGLE_FOLDER_CONTENTS_SYNCED.
                                    equals(event)) {
                                    if (synchResult != null && !synchResult.isSuccess()) {
                                        /// TODO refactor and make common
                                        if
(ResultCode.UNAUTHORIZED.equals(synchResult.getCode()) ||
                                                (synchResult.isException() &&
synchResult.getException()
                                                        instanceof AuthenticatorException)) {
                                            requestCredentialsUpdate(context);
                                        } else if
(RemoteOperationResult.ResultCode.SSL_RECOVERABLE_PEER_UNVERIFIED.equals(
                                                synchResult.getCode())) {
                                            showUntrustedCertDialog(synchResult);
                                        }
                                    }
                                    if (synchFolderRemotePath.equals(OCFile.ROOT_PATH)) {
                                        setUsernameInDrawer(mDrawerLayout, getAccount());
                                    }
                                }
                            }
                            removeStickyBroadcast(intent);
                            Log_OC.d(TAG, "Setting progress visibility to " +
mSyncInProgress);
                            mProgressBar.setIndeterminate(mSyncInProgress);
                            setBackgroundText();
                        }
                        if (synchResult != null) {
                            if (synchResult.getCode().equals(

RemoteOperationResult.ResultCode.SSL_RECOVERABLE_PEER_UNVERIFIED)) {
                                mLastSslUntrustedServerResult = synchResult;
                            }
                        }
                    } catch (RuntimeException e) {
                        // avoid app crashes after changing the serial id of
RemoteOperationResult
                        // in owncloud library with broadcast notifications pending to process
                        removeStickyBroadcast(intent);
                    }
                }
            }
            /**
             * Show a text message on screen view for notifying user if content is
             * loading or folder is empty
             */
            private void setBackgroundText() {
                OCFileListFragment ocFileListFragment = getListOfFilesFragment();
                if (ocFileListFragment != null) {
                    int message = R.string.file_list_loading;
                    if (!mSyncInProgress) {
                        // In case file list is empty
                        message = R.string.file_list_empty;
                    }
                    ocFileListFragment.setMessageForEmptyList(getString(message));
                } else {
                    Log_OC.e(TAG, "OCFileListFragment is null");
                }
            }
```

```
    /**
     * Once the file upload has finished -> update view
     */
    private class UploadFinishReceiver extends BroadcastReceiver {
        /**
         * Once the file upload has finished -> update view
         *
         * @author David A. Velasco
         * {@link BroadcastReceiver} to enable upload feedback in UI
         */
        @Override
        public void onReceive(Context context, Intent intent) {
            try {
                String uploadedRemotePath =
intent.getStringExtra(FileUploader.EXTRA_REMOTE_PATH);
                String accountName = intent.getStringExtra(FileUploader.ACCOUNT_NAME);
                boolean sameAccount = getAccount() != null &&
accountName.equals(getAccount().name);
                OCFile currentDir = getCurrentDir();
                boolean isDescendant = (currentDir != null) && (uploadedRemotePath !=
null) &&
                        (uploadedRemotePath.startsWith(currentDir.getRemotePath()));
                if (sameAccount && isDescendant) {
                    String linkedToRemotePath =
                            intent.getStringExtra(FileUploader.EXTRA_LINKED_TO_PATH);
                    if (linkedToRemotePath == null || isAscendant(linkedToRemotePath))
{
                        refreshListOfFilesFragment();
                    }
                }
                boolean uploadWasFine = intent.getBooleanExtra(
                        FileUploader.EXTRA_UPLOAD_RESULT,
                        false);
                boolean renamedInUpload = getFile().getRemotePath().

equals(intent.getStringExtra(FileUploader.EXTRA_OLD_REMOTE_PATH));
                boolean sameFile = getFile().getRemotePath().equals(uploadedRemotePath)
||
                        renamedInUpload;
                FileFragment details = getSecondFragment();
                boolean detailFragmentIsShown = (details != null &&
                        details instanceof FileDetailFragment);
                if (sameAccount && sameFile && detailFragmentIsShown) {
                    if (uploadWasFine) {

setFile(getStorageManager().getFileByPath(uploadedRemotePath));
                    } else {
                        //TODO remove upload progress bar after upload failed.
                    }
                    if (renamedInUpload) {
                        String newName = (new File(uploadedRemotePath)).getName();
                        Toast msg = Toast.makeText(
                                context,
                                String.format(

getString(R.string.filedetails_renamed_in_upload_msg),
                                        newName),
                                Toast.LENGTH_LONG);
                        msg.show();
                    }
                    if (uploadWasFine || getFile().fileExists()) {
                        ((FileDetailFragment) details).updateFileDetails(false, true);
```

```
                    } else {
                        cleanSecondFragment();
                    }
                    // Force the preview if the file is an image or text file
                    if (uploadWasFine) {
                        OCFile ocFile = getFile();
                        if (PreviewImageFragment.canBePreviewed(ocFile))
                            startImagePreview(getFile());
                        else if (PreviewTextFragment.canBePreviewed(ocFile))
                            startTextPreview(ocFile);
                        // TODO what about other kind of previews?
                    }
                }
                mProgressBar.setIndeterminate(false);
            } finally {
                if (intent != null) {
                    removeStickyBroadcast(intent);
                }
            }
        }
        // TODO refactor this receiver, and maybe DownloadFinishReceiver; this method
is duplicated :S
        private boolean isAscendant(String linkedToRemotePath) {
            OCFile currentDir = getCurrentDir();
            return (
                    currentDir != null &&
                            currentDir.getRemotePath().startsWith(linkedToRemotePath)
            );
        }
    }
    /**
     * Class waiting for broadcast events from the {@link FileDownloader} service.
     * <p/>
     * Updates the UI when a download is started or finished, provided that it is
relevant for the
     * current folder.
     */
    private class DownloadFinishReceiver extends BroadcastReceiver {
        @Override
        public void onReceive(Context context, Intent intent) {
            try {
                boolean sameAccount = isSameAccount(intent);
                String downloadedRemotePath =
                        intent.getStringExtra(FileDownloader.EXTRA_REMOTE_PATH);
                boolean isDescendant = isDescendant(downloadedRemotePath);
                if (sameAccount && isDescendant) {
                    String linkedToRemotePath =

intent.getStringExtra(FileDownloader.EXTRA_LINKED_TO_PATH);
                    if (linkedToRemotePath == null || isAscendant(linkedToRemotePath))
{
                        refreshListOfFilesFragment();
                    }
                    refreshSecondFragment(
                            intent.getAction(),
                            downloadedRemotePath,

intent.getBooleanExtra(FileDownloader.EXTRA_DOWNLOAD_RESULT, false)
                    );
                }
                if (mWaitingToSend != null) {
                    mWaitingToSend =
```

```
getStorageManager().getFileByPath(mWaitingToSend.getRemotePath());
                        if (mWaitingToSend.isDown()) {
                            sendDownloadedFile();
                        }
                    }
                } finally {
                    if (intent != null) {
                        removeStickyBroadcast(intent);
                    }
                }
            }
            private boolean isDescendant(String downloadedRemotePath) {
                OCFile currentDir = getCurrentDir();
                return (
                        currentDir != null &&
                                downloadedRemotePath != null &&

downloadedRemotePath.startsWith(currentDir.getRemotePath())
                );
            }
            private boolean isAscendant(String linkedToRemotePath) {
                OCFile currentDir = getCurrentDir();
                return (
                        currentDir != null &&
                                currentDir.getRemotePath().startsWith(linkedToRemotePath)
                );
            }
            private boolean isSameAccount(Intent intent) {
                String accountName = intent.getStringExtra(FileDownloader.ACCOUNT_NAME);
                return (accountName != null && getAccount() != null &&
                        accountName.equals(getAccount().name));
            }
        }
        public void browseToRoot() {
            OCFileListFragment listOfFiles = getListOfFilesFragment();
            if (listOfFiles != null) {  // should never be null, indeed
                OCFile root = getStorageManager().getFileByPath(OCFile.ROOT_PATH);
                listOfFiles.listDirectory(root);
                // TODO Enable when "On Device" is recovered ?
                // listOfFiles.listDirectory(root, MainApp.getOnlyOnDevice());
                setFile(listOfFiles.getCurrentFile());
                startSyncFolderOperation(root, false);
            }
            cleanSecondFragment();
        }
        /**
         * {@inheritDoc}
         * Updates action bar and second fragment, if in dual pane mode.
         */
        @Override
        public void onBrowsedDownTo(OCFile directory) {
            setFile(directory);
            cleanSecondFragment();
            // Sync Folder
            startSyncFolderOperation(directory, false);
        }
        /**
         * Shows the information of the {@link OCFile} received as a
         * parameter in the second fragment.
         *
```

```
     * @param file {@link OCFile} whose details will be shown
     */
    @Override
    public void showDetails(OCFile file) {
        Fragment detailFragment = FileDetailFragment.newInstance(file, getAccount());
        setSecondFragment(detailFragment);
        updateFragmentsVisibility(true);
        updateActionBarTitleAndHomeButton(file);
        setFile(file);
    }
    @Override
    protected void updateActionBarTitleAndHomeButton(OCFile chosenFile) {
        if (mDualPane) {
            // in dual pane mode, keep the focus of title an action bar in the current
folder
            super.updateActionBarTitleAndHomeButton(getCurrentDir());
        } else {
            super.updateActionBarTitleAndHomeButton(chosenFile);
        }
    }
    @Override
    protected ServiceConnection newTransferenceServiceConnection() {
        return new ListServiceConnection();
    }
    /**
     * Defines callbacks for service binding, passed to bindService()
     */
    private class ListServiceConnection implements ServiceConnection {
        @Override
        public void onServiceConnected(ComponentName component, IBinder service) {
            if (component.equals(new ComponentName(
                    FileDisplayActivity.this, FileDownloader.class))) {
                Log_OC.d(TAG, "Download service connected");
                mDownloaderBinder = (FileDownloaderBinder) service;
                if (mWaitingToPreview != null)
                    if (getStorageManager() != null) {
                        // update the file
                        mWaitingToPreview =

getStorageManager().getFileById(mWaitingToPreview.getFileId());
                        if (!mWaitingToPreview.isDown()) {
                            requestForDownload();
                        }
                    }
            } else if (component.equals(new ComponentName(FileDisplayActivity.this,
                    FileUploader.class))) {
                Log_OC.d(TAG, "Upload service connected");
                mUploaderBinder = (FileUploaderBinder) service;
            } else {
                return;
            }
            // a new chance to get the mDownloadBinder through
            // getFileDownloadBinder() - THIS IS A MESS
            OCFileListFragment listOfFiles = getListOfFilesFragment();
            if (listOfFiles != null) {
                listOfFiles.listDirectory();
                // TODO Enable when "On Device" is recovered ?
                // listOfFiles.listDirectory(MainApp.getOnlyOnDevice());
            }
            FileFragment secondFragment = getSecondFragment();
            if (secondFragment != null && secondFragment instanceof FileDetailFragment)
```

```
{
                    FileDetailFragment detailFragment = (FileDetailFragment)
secondFragment;
                    detailFragment.listenForTransferProgress();
                    detailFragment.updateFileDetails(false, false);
               }
          }
          @Override
          public void onServiceDisconnected(ComponentName component) {
               if (component.equals(new ComponentName(FileDisplayActivity.this,
                         FileDownloader.class))) {
                    Log_OC.d(TAG, "Download service disconnected");
                    mDownloaderBinder = null;
               } else if (component.equals(new ComponentName(FileDisplayActivity.this,
                         FileUploader.class))) {
                    Log_OC.d(TAG, "Upload service disconnected");
                    mUploaderBinder = null;
               }
          }
     }
     /**
      * Updates the view associated to the activity after the finish of some operation
over files
      * in the current account.
      *
      * @param operation Removal operation performed.
      * @param result    Result of the removal.
      */
     @Override
     public void onRemoteOperationFinish(RemoteOperation operation, RemoteOperationResult
result) {
          super.onRemoteOperationFinish(operation, result);
          if (operation instanceof RemoveFileOperation) {
               onRemoveFileOperationFinish((RemoveFileOperation) operation, result);
          } else if (operation instanceof RenameFileOperation) {
               onRenameFileOperationFinish((RenameFileOperation) operation, result);
          } else if (operation instanceof SynchronizeFileOperation) {
               onSynchronizeFileOperationFinish((SynchronizeFileOperation) operation,
result);
          } else if (operation instanceof CreateFolderOperation) {
               onCreateFolderOperationFinish((CreateFolderOperation) operation, result);
          } else if (operation instanceof MoveFileOperation) {
               onMoveFileOperationFinish((MoveFileOperation) operation, result);
          } else if (operation instanceof CopyFileOperation) {
               onCopyFileOperationFinish((CopyFileOperation) operation, result);
          }
     }
     private void refreshShowDetails() {
          FileFragment details = getSecondFragment();
          if (details != null) {
               OCFile file = details.getFile();
               if (file != null) {
                    file = getStorageManager().getFileByPath(file.getRemotePath());
                    if (details instanceof PreviewMediaFragment) {
                         // Refresh  OCFile of the fragment
                         ((PreviewMediaFragment) details).updateFile(file);
                    } else if (details instanceof PreviewTextFragment) {
                         // Refresh  OCFile of the fragment
                         ((PreviewTextFragment) details).updateFile(file);
                    } else {
                         showDetails(file);
                    }
```

```
            }
            invalidateOptionsMenu();
        }
    }
    /**
     * Updates the view associated to the activity after the finish of an operation
trying to
     * remove a file.
     *
     * @param operation Removal operation performed.
     * @param result    Result of the removal.
     */
    private void onRemoveFileOperationFinish(RemoveFileOperation operation,
                                             RemoteOperationResult result) {
        Toast msg = Toast.makeText(this,
            ErrorMessageAdapter.getErrorCauseMessage(result, operation,
getResources()),
            Toast.LENGTH_LONG);
        msg.show();
        if (result.isSuccess()) {
            OCFile removedFile = operation.getFile();
            FileFragment second = getSecondFragment();
            if (second != null && removedFile.equals(second.getFile())) {
                if (second instanceof PreviewMediaFragment) {
                    ((PreviewMediaFragment) second).stopPreview(true);
                }
                setFile(getStorageManager().getFileById(removedFile.getParentId()));
                cleanSecondFragment();
            }
            if
(getStorageManager().getFileById(removedFile.getParentId()).equals(getCurrentDir())) {
                refreshListOfFilesFragment();
            }
            invalidateOptionsMenu();
        } else {
            if (result.isSslRecoverableException()) {
                mLastSslUntrustedServerResult = result;
                showUntrustedCertDialog(mLastSslUntrustedServerResult);
            }
        }
    }
    /**
     * Updates the view associated to the activity after the finish of an operation
trying to move a
     * file.
     *
     * @param operation Move operation performed.
     * @param result    Result of the move operation.
     */
    private void onMoveFileOperationFinish(MoveFileOperation operation,
                                           RemoteOperationResult result) {
        if (result.isSuccess()) {
            refreshListOfFilesFragment();
        } else {
            try {
                Toast msg = Toast.makeText(FileDisplayActivity.this,
                        ErrorMessageAdapter.getErrorCauseMessage(result, operation,
getResources()),
                        Toast.LENGTH_LONG);
                msg.show();
            } catch (NotFoundException e) {
                Log_OC.e(TAG, "Error while trying to show fail message ", e);
```

```
            }
        }
    }
    /**
     * Updates the view associated to the activity after the finish of an operation
trying to copy a
     * file.
     *
     * @param operation Copy operation performed.
     * @param result    Result of the copy operation.
     */
    private void onCopyFileOperationFinish(CopyFileOperation operation,
RemoteOperationResult result) {
        if (result.isSuccess()) {
            refreshListOfFilesFragment();
        } else {
            try {
                Toast msg = Toast.makeText(FileDisplayActivity.this,
                        ErrorMessageAdapter.getErrorCauseMessage(result, operation,
getResources()),
                        Toast.LENGTH_LONG);
                msg.show();
            } catch (NotFoundException e) {
                Log_OC.e(TAG, "Error while trying to show fail message ", e);
            }
        }
    }
    /**
     * Updates the view associated to the activity after the finish of an operation
trying to rename
     * a file.
     *
     * @param operation Renaming operation performed.
     * @param result    Result of the renaming.
     */
    private void onRenameFileOperationFinish(RenameFileOperation operation,
                                             RemoteOperationResult result) {
        OCFile renamedFile = operation.getFile();
        if (result.isSuccess()) {
            FileFragment details = getSecondFragment();
            if (details != null) {
                if (details instanceof FileDetailFragment &&
                        renamedFile.equals(details.getFile())) {
                    ((FileDetailFragment) details).updateFileDetails(renamedFile,
getAccount());
                    showDetails(renamedFile);
                } else if (details instanceof PreviewMediaFragment &&
                        renamedFile.equals(details.getFile())) {
                    ((PreviewMediaFragment) details).updateFile(renamedFile);
                    if (PreviewMediaFragment.canBePreviewed(renamedFile)) {
                        int position = ((PreviewMediaFragment) details).getPosition();
                        startMediaPreview(renamedFile, position, true);
                    } else {
                        getFileOperationsHelper().openFile(renamedFile);
                    }
                } else if (details instanceof PreviewTextFragment &&
                        renamedFile.equals(details.getFile())) {
                    ((PreviewTextFragment) details).updateFile(renamedFile);
                    if (PreviewTextFragment.canBePreviewed(renamedFile)) {
                        startTextPreview(renamedFile);
                    } else {
                        getFileOperationsHelper().openFile(renamedFile);
```

```
                    }
                }
            }
            if
(getStorageManager().getFileById(renamedFile.getParentId()).equals(getCurrentDir())) {
                refreshListOfFilesFragment();
            }
        } else {
            Toast msg = Toast.makeText(this,
                    ErrorMessageAdapter.getErrorCauseMessage(result, operation,
getResources()),
                    Toast.LENGTH_LONG);
            msg.show();
            if (result.isSslRecoverableException()) {
                mLastSslUntrustedServerResult = result;
                showUntrustedCertDialog(mLastSslUntrustedServerResult);
            }
        }
    }
    private void onSynchronizeFileOperationFinish(SynchronizeFileOperation operation,
                                                  RemoteOperationResult result) {
        if (result.isSuccess()) {
            if (operation.transferWasRequested()) {
                OCFile syncedFile = operation.getLocalFile();
                onTransferStateChanged(syncedFile, true, true);
                invalidateOptionsMenu();
                refreshShowDetails();
            }
        }
    }
    /**
     * Updates the view associated to the activity after the finish of an operation
trying create a
     * new folder
     *
     * @param operation Creation operation performed.
     * @param result    Result of the creation.
     */
    private void onCreateFolderOperationFinish(CreateFolderOperation operation,
                                               RemoteOperationResult result) {
        if (result.isSuccess()) {
            refreshListOfFilesFragment();
        } else {
            try {
                Toast msg = Toast.makeText(FileDisplayActivity.this,
                        ErrorMessageAdapter.getErrorCauseMessage(result, operation,
getResources()),
                        Toast.LENGTH_LONG);
                msg.show();
            } catch (NotFoundException e) {
                Log_OC.e(TAG, "Error while trying to show fail message ", e);
            }
        }
    }
    /**
     * {@inheritDoc}
     */
    @Override
    public void onTransferStateChanged(OCFile file, boolean downloading, boolean
uploading) {
        refreshListOfFilesFragment();
        FileFragment details = getSecondFragment();
```

```
        if (details != null && details instanceof FileDetailFragment &&
                file.equals(details.getFile())) {
            if (downloading || uploading) {
                ((FileDetailFragment) details).updateFileDetails(file, getAccount());
            } else {
                if (!file.fileExists()) {
                    cleanSecondFragment();
                } else {
                    ((FileDetailFragment) details).updateFileDetails(false, true);
                }
            }
        }
    }
    private void requestForDownload() {
        Account account = getAccount();
        //if (!mWaitingToPreview.isDownloading()) {
        if (!mDownloaderBinder.isDownloading(account, mWaitingToPreview)) {
            Intent i = new Intent(this, FileDownloader.class);
            i.putExtra(FileDownloader.EXTRA_ACCOUNT, account);
            i.putExtra(FileDownloader.EXTRA_FILE, mWaitingToPreview);
            startService(i);
        }
    }
    @Override
    public void onSavedCertificate() {
        startSyncFolderOperation(getCurrentDir(), false);
    }
    /**
     * Starts an operation to refresh the requested folder.
     * <p/>
     * The operation is run in a new background thread created on the fly.
     * <p/>
     * The refresh updates is a "light sync": properties of regular files in folder are
updated (including
     * associated shares), but not their contents. Only the contents of files marked to
be kept-in-sync are
     * synchronized too.
     *
     * @param folder     Folder to refresh.
     * @param ignoreETag If 'true', the data from the server will be fetched and sync'ed
even if the eTag
     *                   didn't change.
     */
    public void startSyncFolderOperation(final OCFile folder, final boolean ignoreETag)
{
        // the execution is slightly delayed to allow the activity get the window focus
if it's being started
        // or if the method is called from a dialog that is being dismissed
        getHandler().postDelayed(
                new Runnable() {
                    @Override
                    public void run() {
                        if (hasWindowFocus()) {
                            long currentSyncTime = System.currentTimeMillis();
                            mSyncInProgress = true;
                            // perform folder synchronization
                            RemoteOperation synchFolderOp = new
RefreshFolderOperation(folder,
                                    currentSyncTime,
                                    false,
                                    getFileOperationsHelper().isSharedSupported(),
                                    ignoreETag,
```

```
                                        getStorageManager(),
                                        getAccount(),
                                        getApplicationContext()
                                );
                                synchFolderOp.execute(
                                        getAccount(),
                                        MainApp.getAppContext(),
                                        FileDisplayActivity.this,
                                        null,
                                        null
                                );
                                mProgressBar.setIndeterminate(true);
                                setBackgroundText();
                            }   // else: NOTHING ; lets' not refresh when the user rotates
the device but there is
                            // another window floating over
                        }
                    },
                    DELAY_TO_REQUEST_OPERATIONS_LATER
            );
        }
        private void requestForDownload(OCFile file) {
            Account account = getAccount();
            if (!mDownloaderBinder.isDownloading(account, mWaitingToPreview)) {
                Intent i = new Intent(this, FileDownloader.class);
                i.putExtra(FileDownloader.EXTRA_ACCOUNT, account);
                i.putExtra(FileDownloader.EXTRA_FILE, file);
                startService(i);
            }
        }
        private void sendDownloadedFile() {
            getFileOperationsHelper().sendDownloadedFile(mWaitingToSend);
            mWaitingToSend = null;
        }
        /**
         * Requests the download of the received {@link OCFile} , updates the UI
         * to monitor the download progress and prepares the activity to send the file
         * when the download finishes.
         *
         * @param file {@link OCFile} to download and preview.
         */
        public void startDownloadForSending(OCFile file) {
            mWaitingToSend = file;
            requestForDownload(mWaitingToSend);
            boolean hasSecondFragment = (getSecondFragment() != null);
            updateFragmentsVisibility(hasSecondFragment);
        }
        /**
         * Opens the image gallery showing the image {@link OCFile} received as parameter.
         *
         * @param file Image {@link OCFile} to show.
         */
        public void startImagePreview(OCFile file) {
            Intent showDetailsIntent = new Intent(this, PreviewImageActivity.class);
            showDetailsIntent.putExtra(EXTRA_FILE, file);
            showDetailsIntent.putExtra(EXTRA_ACCOUNT, getAccount());
            startActivity(showDetailsIntent);
        }
        /**
         * Stars the preview of an already down media {@link OCFile}.
         *
```

```
     * @param file                  Media {@link OCFile} to preview.
     * @param startPlaybackPosition Media position where the playback will be started,
     *                              in milliseconds.
     * @param autoplay              When 'true', the playback will start without user
     *                              interactions.
     */
    public void startMediaPreview(OCFile file, int startPlaybackPosition, boolean
autoplay) {
        Fragment mediaFragment = new PreviewMediaFragment(file, getAccount(),
startPlaybackPosition,
                autoplay);
        setSecondFragment(mediaFragment);
        updateFragmentsVisibility(true);
        updateActionBarTitleAndHomeButton(file);
        setFile(file);
    }
    /**
     * Stars the preview of a text file {@link OCFile}.
     *
     * @param file Text {@link OCFile} to preview.
     */
    public void startTextPreview(OCFile file) {
        Bundle args = new Bundle();
        args.putParcelable(EXTRA_FILE, file);
        args.putParcelable(EXTRA_ACCOUNT, getAccount());
        Fragment textPreviewFragment = Fragment.instantiate(getApplicationContext(),
                PreviewTextFragment.class.getName(), args);
        setSecondFragment(textPreviewFragment);
        updateFragmentsVisibility(true);
        //updateNavigationElementsInActionBar(file);
        setFile(file);
    }
    /**
     * Requests the download of the received {@link OCFile} , updates the UI
     * to monitor the download progress and prepares the activity to preview
     * or open the file when the download finishes.
     *
     * @param file {@link OCFile} to download and preview.
     */
    public void startDownloadForPreview(OCFile file) {
        Fragment detailFragment = FileDetailFragment.newInstance(file, getAccount());
        setSecondFragment(detailFragment);
        mWaitingToPreview = file;
        requestForDownload();
        updateFragmentsVisibility(true);
        updateActionBarTitleAndHomeButton(file);
        setFile(file);
    }
    public void cancelTransference(OCFile file) {
        getFileOperationsHelper().cancelTransference(file);
        if (mWaitingToPreview != null &&
                mWaitingToPreview.getRemotePath().equals(file.getRemotePath())) {
            mWaitingToPreview = null;
        }
        if (mWaitingToSend != null &&
                mWaitingToSend.getRemotePath().equals(file.getRemotePath())) {
            mWaitingToSend = null;
        }
        onTransferStateChanged(file, false, false);
    }
    @Override
```

```
    public void onRefresh(boolean ignoreETag) {
        refreshList(ignoreETag);
    }
    @Override
    public void onRefresh() {
        refreshList(true);
    }
    private void refreshList(boolean ignoreETag) {
        OCFileListFragment listOfFiles = getListOfFilesFragment();
        if (listOfFiles != null) {
            OCFile folder = listOfFiles.getCurrentFile();
            if (folder != null) {
                /*mFile =
mContainerActivity.getStorageManager().getFileById(mFile.getFileId());
                listDirectory(mFile);*/
                startSyncFolderOperation(folder, ignoreETag);
            }
        }
    }
    private void sortByDate(boolean ascending) {
        getListOfFilesFragment().sortByDate(ascending);
    }
    private void sortBySize(boolean ascending) {
        getListOfFilesFragment().sortBySize(ascending);
    }
    private void sortByName(boolean ascending) {
        getListOfFilesFragment().sortByName(ascending);
    }
    private boolean isGridView() {
        return getListOfFilesFragment().isGridView();
    }
    public void allFilesOption() {
        browseToRoot();
    }
}
```

11.5 media/andreas/SSD_Data/OwncloudAndroidBuilding/android/src/com/owncloud/android/files/services/ConnectivityActionReceiver.java

```
/**
 *   ownCloud Android client application
 *
 *   @author LukeOwncloud
 *   Copyright (C) 2016 ownCloud Inc.
 *
 *   This program is free software: you can redistribute it and/or modify
 *   it under the terms of the GNU General Public License version 2,
 *   as published by the Free Software Foundation.
 *
 *   This program is distributed in the hope that it will be useful,
 *   but WITHOUT ANY WARRANTY; without even the implied warranty of
 *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 *   GNU General Public License for more details.
 *
 *   You should have received a copy of the GNU General Public License
 *   along with this program.  If not, see <http://www.gnu.org/licenses/>.
```

```
 *
 */
package com.owncloud.android.files.services;
import android.content.BroadcastReceiver;
import android.content.ComponentName;
import android.content.Context;
import android.content.Intent;
import android.content.pm.PackageManager;
import android.net.NetworkInfo;
import android.net.wifi.WifiInfo;
import android.net.wifi.WifiManager;
import android.os.Bundle;
import android.support.v4.content.LocalBroadcastManager;
import android.util.Log;
import com.owncloud.android.db.PreferenceManager;
import com.owncloud.android.db.UploadResult;
import com.owncloud.android.lib.common.utils.Log_OC;
import com.owncloud.android.services.MQTTService;
/**
 * Receives all connectivity action from Android OS at all times and performs
 * required OC actions. For now that are: - Signal connectivity to
 * {@link FileUploader}.
 *
 * Later can be added: - Signal connectivity to download service, deletion
 * service, ... - Handle offline mode (cf.
 * https://github.com/owncloud/android/issues/162)
 *
 * Have fun with the comments :S
 */
public class ConnectivityActionReceiver extends BroadcastReceiver {
    private static final String TAG = ConnectivityActionReceiver.class.getSimpleName();
    /**
     * Magic keyword, by Google.
     *
     * {@See
http://developer.android.com/intl/es/reference/android/net/wifi/WifiInfo.html#getSSID()}
     */
    private static final String UNKNOWN_SSID = "<unknown ssid>";
    LocalBroadcastManager broadcaster;
    Intent mqttservice;
    @Override
    public void onReceive(final Context context, Intent intent) {
        // LOG ALL EVENTS:
        Log_OC.v(TAG, "action: " + intent.getAction());
        Log_OC.v(TAG, "component: " + intent.getComponent());
        Bundle extras = intent.getExtras();
        if (extras != null) {
            for (String key : extras.keySet()) {
                Log_OC.v(TAG, "key [" + key + "]: " + extras.get(key));
            }
        } else {
            Log_OC.v(TAG, "no extras");
        }
        /**
         * There is an interesting mess to process
WifiManager.NETWORK_STATE_CHANGED_ACTION and
         * ConnectivityManager.CONNECTIVITY_ACTION in a simple and reliable way.
         *
         * The former triggers much more events than what we really need to know about
Wifi connection.
         *
         * But there are annoying uncertainties about
ConnectivityManager.CONNECTIVITY_ACTION due
```

```
         * to the deprecation of ConnectivityManager.EXTRA_NETWORK_INFO in API level
14, and the absence
         * of ConnectivityManager.EXTRA_NETWORK_TYPE until API level 17. Dear Google,
how should we
         * handle API levels 14 to 16?
         *
         * In the end maybe we need to keep in memory the current knowledge about
connectivity
         * and update it taking into account several Intents received in a row
         *
         * But first let's try something "simple" to keep a basic retry of instant
uploads in
         * version 1.9.2, similar to the existent until 1.9.1. To be improved.
         */
        if(intent.getAction().equals(WifiManager.NETWORK_STATE_CHANGED_ACTION)) {
            NetworkInfo networkInfo =
                intent.getParcelableExtra(WifiManager.EXTRA_NETWORK_INFO);
            WifiInfo wifiInfo =
                intent.getParcelableExtra(WifiManager.EXTRA_WIFI_INFO);
            String bssid =
                intent.getStringExtra(WifiManager.EXTRA_BSSID);
            if(networkInfo.isConnected()   &&      // not enough; see (*) right below
                wifiInfo != null    &&
                !UNKNOWN_SSID.equals(wifiInfo.getSSID().toLowerCase()) &&
                bssid != null
                    ) {
                Log_OC.d(TAG, "WiFi connected");
                wifiConnected(context);
            } else {
                // TODO tons of things to check to conclude disconnection;
                // TODO maybe alternative commented below, based on CONNECTIVITY_ACTION
is better
                Log_OC.d(TAG, "WiFi disconnected ... but don't know if right now");
            }
        }
        // (*) When WiFi is lost, an Intent with network state CONNECTED and SSID
"<unknown ssid>" is
        //      received right before another Intent with network state DISCONNECTED;
needs to
        //      be differentiated of a new Wifi connection.
        //
        //  Besides, with a new connection two Intents are received, having only the
second the extra
        //  WifiManager.EXTRA_BSSID, with the BSSID of the access point accessed.
        //
        //  Not sure if this protocol is exact, since it's not documented. Only found
mild references in
        //   -
http://developer.android.com/intl/es/reference/android/net/wifi/WifiInfo.html#getSSID()
        //   -
http://developer.android.com/intl/es/reference/android/net/wifi/WifiManager.html#EXTRA_B
SSID
        //  and reproduced in Nexus 5 with Android 6.
        /**
         * Possible alternative attending ConnectivityManager.CONNECTIVITY_ACTION.
         *
         * Let's see what QA has to say
         *
        if(intent.getAction().equals(ConnectivityManager.CONNECTIVITY_ACTION)) {
            NetworkInfo networkInfo = intent.getParcelableExtra(
                    ConnectivityManager.EXTRA_NETWORK_INFO      // deprecated in API
14
```

```
            );
            int networkType = intent.getIntExtra(
                    ConnectivityManager.EXTRA_NETWORK_TYPE,     // only from API level
17
                    -1
            );
            boolean couldBeWifiAction =
                    (networkInfo == null && networkType < 0)    ||      // cases of
lack of info
                    networkInfo.getType() == ConnectivityManager.TYPE_WIFI  ||
                    networkType == ConnectivityManager.TYPE_WIFI;
            if (couldBeWifiAction) {
                if (ConnectivityUtils.isAppConnectedViaWiFi(context)) {
                    Log_OC.d(TAG, "WiFi connected");
                    wifiConnected(context);
                } else {
                    Log_OC.d(TAG, "WiFi disconnected");
                    wifiDisconnected(context);
                }
            } /* else, CONNECTIVIY_ACTION is (probably) about other network interface
(mobile, bluetooth, ...)
        }
        */
    }
    private void wifiConnected(Context context) {
        Log.v(TAG,"wifi connected, so mqtt is restarted");
        mqttservice = new Intent(context.getApplicationContext(),MQTTService.class);
        mqttservice.setAction("com.mqttservice.restart");
        broadcaster = LocalBroadcastManager.getInstance(context);
        broadcaster.sendBroadcast(mqttservice);
        // for the moment, only recovery of instant uploads, similar to behaviour in
release 1.9.1
        if (
                (PreferenceManager.instantPictureUploadEnabled(context) &&
                        PreferenceManager.instantPictureUploadViaWiFiOnly(context)) ||
                (PreferenceManager.instantVideoUploadEnabled(context) &&
                        PreferenceManager.instantVideoUploadViaWiFiOnly(context))
                ) {
            Log_OC.d(TAG, "Requesting retry of instant uploads (& friends)");
            FileUploader.UploadRequester requester = new
FileUploader.UploadRequester();
            requester.retryFailedUploads(
                context,
                null,
                UploadResult.NETWORK_CONNECTION     // for the interrupted when Wifi
fell, if any
                // (side effect: any upload failed due to network error will be retried
too, instant or not)
            );
            requester.retryFailedUploads(
                context,
                null,
                UploadResult.DELAYED_FOR_WIFI       // for the rest of enqueued when
Wifi fell
            );
        }
    }
    private void wifiDisconnected(Context context) {
        // TODO something smart
        // NOTE: explicit cancellation of only-wifi instant uploads is not needed
anymore, since currently:
        //  - any upload in progress will be interrupted due to the lack of
```

```
connectivity while the device
        //      reconnects through other network interface;
        //  - FileUploader checks instant upload settings and connection state before
executing each
        //    upload operation, so other pending instant uploads after the current one
will not be run
        //    (currently are silently moved to FAILED state)
    }
    static public void enableActionReceiver(Context context) {
        PackageManager pm = context.getPackageManager();
        ComponentName compName = new ComponentName(context.getApplicationContext(),
ConnectivityActionReceiver.class);
        pm.setComponentEnabledSetting(compName,
PackageManager.COMPONENT_ENABLED_STATE_ENABLED,
                PackageManager.DONT_KILL_APP);
    }
    static public void disableActionReceiver(Context context) {
        PackageManager pm = context.getPackageManager();
        ComponentName compName = new ComponentName(context.getApplicationContext(),
ConnectivityActionReceiver.class);
        pm.setComponentEnabledSetting(compName,
PackageManager.COMPONENT_ENABLED_STATE_DISABLED,
                PackageManager.DONT_KILL_APP);
    }
}
```

11.6 android/src/com/owncloud/android/services/MQTTService.java

```
package com.owncloud.android.services;
import android.app.AlarmManager;
import android.app.Notification;
import android.app.NotificationManager;
import android.app.PendingIntent;
import android.app.Service;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
import android.content.SharedPreferences;
import android.graphics.Color;
import android.net.ConnectivityManager;
import android.os.Binder;
import android.os.Handler;
import android.os.IBinder;
import android.os.PowerManager;
import android.os.PowerManager.WakeLock;
import android.preference.PreferenceManager;
import android.provider.Settings;
import android.provider.Settings.Secure;
import android.support.v4.content.LocalBroadcastManager;
import android.util.Log;
import android.widget.Toast;
import com.ibm.mqtt.IMqttClient;
import com.ibm.mqtt.MqttClient;
import com.ibm.mqtt.MqttException;
import com.ibm.mqtt.MqttNotConnectedException;
```

```
import com.ibm.mqtt.MqttPersistence;
import com.ibm.mqtt.MqttPersistenceException;
import com.ibm.mqtt.MqttSimpleCallback;
import com.owncloud.android.ui.activity.FileDisplayActivity;
import java.lang.ref.WeakReference;
import java.util.Calendar;
import java.util.Date;
import java.util.Enumeration;
import java.util.Hashtable;
import java.util.Timer;
import java.util.TimerTask;
/*
 * An example of how to implement an MQTT client in Android, able to receive
 *  push notifications from an MQTT message broker server.
 *
 *  Dale Lane (dale.lane@gmail.com)
 *    28 Jan 2011
 */
public class MQTTService extends Service implements MqttSimpleCallback
{
    /************************************************************************/
    /*    CONSTANTS                                                         */
    /************************************************************************/

   private static final String TAG ="MQTT";
   private final static String RESP_TOPIC = "Test";

    // something unique to identify your app - used for stuff like accessing
    //   application preferences
    public static final String APP_ID = "com.imsight.androidmqtt";

    // constants used to notify the Activity UI of received messages
    public static final String MQTT_MSG_RECEIVED_INTENT =
"com.imsight.androidmqtt.MSGRECVD";
    public static final String MQTT_MSG_RECEIVED_TOPIC  =
"com.imsight.androidmqtt.MSGRECVD_TOPIC";
    public static final String MQTT_MSG_RECEIVED_MSG    =
"com.imsight.androidmqtt.MSGRECVD_MSGBODY";

    // constants used to tell the Activity UI the connection status
    public static final String MQTT_STATUS_INTENT = "com.imsight.androidmqtt.STATUS";
    public static final String MQTT_STATUS_MSG    =
"com.imsight.androidmqtt.STATUS_MSG";
    // constant used internally to schedule the next ping event
    public static final String MQTT_PING_ACTION = "com.imsight.androidmqtt.PING";

    // constants used by status bar notifications
    public static final int MQTT_NOTIFICATION_ONGOING = 1;
    public static final int MQTT_NOTIFICATION_UPDATE  = 2;

    // constants used to define MQTT connection status
    public enum MQTTConnectionStatus
    {
        INITIAL,                            // initial status
        CONNECTING,                         // attempting to connect
        CONNECTED,                          // connected
        NOTCONNECTED_WAITINGFORINTERNET,    // can't connect because the phone
                                            //     does not have Internet access
        NOTCONNECTED_USERDISCONNECT,        // user has explicitly requested
                                            //     disconnection
        NOTCONNECTED_DATADISABLED,          // can't connect because the user
                                            //     has disabled data access
        NOTCONNECTED_UNKNOWNREASON          // failed to connect for some reason
```

```
}
// MQTT constants
public static final int MAX_MQTT_CLIENTID_LENGTH = 22;

/************************************************************************/
/*    VARIABLES used to maintain state                                  */
/************************************************************************/

// status of MQTT client connection
private MQTTConnectionStatus connectionStatus = MQTTConnectionStatus.INITIAL;

/************************************************************************/
/*    VARIABLES used to configure MQTT connection                       */
/************************************************************************/

// taken from preferences
//    host name of the server we're receiving push notifications from
private String brokerHostName       = "";
// taken from preferences
//    topic we want to receive messages about
//    can include wildcards - e.g.  '#' matches anything
private String topicName            = "";

// defaults - this sample uses very basic defaults for it's interactions
//   with message brokers
private int             brokerPortNumber     = 1883;
private MqttPersistence usePersistence       = null;
private boolean         cleanStart           = false;
private int[]           qualitiesOfService   = { 0 } ;
//  how often should the app ping the server to keep the connection alive?
//
//   too frequently - and you waste battery life
//   too infrequently - and you wont notice if you lose your connection
//                       until the next unsuccessfull attempt to ping
//
//   it's a trade-off between how time-sensitive the data is that your
//      app is handling, vs the acceptable impact on battery life
//
//   it is perhaps also worth bearing in mind the network's support for
//     long running, idle connections. Ideally, to keep a connection open
//     you want to use a keep alive value that is less than the period of
//     time after which a network operator will kill an idle connection
private short           keepAliveSeconds     = 20 * 60;

// This is how the Android client app will identify itself to the
//  message broker.
// It has to be unique to the broker - two clients are not permitted to
//  connect to the same broker using the same client ID.
private String mqttClientId = null;


/************************************************************************/
/*    VARIABLES  - other local variables                                */
/************************************************************************/
// connection to the message broker
private IMqttClient mqttClient = null;

// receiver that notifies the Service when the phone gets data connection
private NetworkConnectionIntentReceiver netConnReceiver;

// receiver that notifies the Service when the user changes data use preferences
```

```
    private BackgroundDataChangeIntentReceiver dataEnabledReceiver;

    // receiver that wakes the Service up when it's time to ping the server
    private PingSender pingSender;
    /**********************************************************************/
    /*   VARIABLES ADDED FOR OWNCLOUD CLIENT                              */
    /**********************************************************************/
    private Handler handler = new Handler();
    private BroadcastReceiver receiver;
    private String pending_message = "none";
    private Boolean RESTART = false;
    /**********************************************************************/
    /*    METHODS - core Service lifecycle methods                        */
    /**********************************************************************/
    // see http://developer.android.com/guide/topics/fundamentals.html#lcycles
    private void send_pending_messages(){
        if (!pending_message.equals("none")) {
            Log.v(TAG,"Just Sent a pending message, "+pending_message);
            publishMessageToTopic(pending_message);
            pending_message="none";
        }
    }
    @Override
    public void onCreate()
    {
        Log.v(TAG, "Current broker address is: " + brokerHostName);
        Log.v(TAG, "Current Mqtt topic is: " + topicName);
//Παρακάτω παίρνουμε τη διεύθυνση του Broker και το θέμα από τις ρυθμίσεις.
        PreferenceManager preferenceManager ;
        SharedPreferences prefs =
PreferenceManager.getDefaultSharedPreferences(getApplicationContext());
        String updatedBrokerHostName=prefs.getString("brokerHostName",brokerHostName);
        String updatedTopicName = prefs.getString("topicName",topicName);
        if (updatedBrokerHostName!=null && !
updatedBrokerHostName.equals(brokerHostName)) {
            Log.v(TAG, "The updated broker address is: " + updatedBrokerHostName);
            brokerHostName = updatedBrokerHostName;
        }
        if (updatedTopicName!=null && !updatedTopicName.equals(topicName)) {
            Log.v(TAG, "The updated Mqtt topic is: " + updatedTopicName);
            topicName = updatedTopicName;
        }
        receiver = new BroadcastReceiver() {
            @Override
            public void onReceive(Context context, Intent intent) {
                if (intent.getAction().equals("com.owncloud.preferences.update_mqtt"))
{ //Ανανεώθηκε η διεύθυνση του Broker ή το θέμα.
                    RESTART = true;
                    stopSelf();
                } else if (intent.getAction().equals("com.mqttservice.restart")) {
//Συνδεθήκαμε στο δίκτυο, και πρέπει να δοκιμάσουμε να συνδεθούμε ξανά.
                    Log.d(TAG, "Received Alert to subscribe once more");
                    Log.v(TAG, "these are the current values for broker: " +
brokerHostName + " and topic: " + topicName);
                    Timer timer = new Timer();
                    timer.schedule(new TimerTask() {
                        @Override
                        public void run() {
//Προσπαθούμε να δημιουργήσουμε σύνδεση με τις καινούργιες παραμέτρους.
                            Log.v(TAG, "Trying to connect with the help of Timer to
topic " + topicName + " and broker " + brokerHostName);
                            if (mqttClient == null) {
```

```
                                    defineConnectionToBroker(brokerHostName);
                                }
                                if (connectToBroker()) {
                                    subscribeToTopic(topicName);
                                    send_pending_messages();
                                    Log.v(TAG, "Connected with the help of Timer method
and sent pending messages");
                                    cancel(); //Ακυρώνουμε τη χρονοπρογραμματισμένη
εργασία διότι στάλθηκαν τα εκκρεμή μηνύματα.
                                }
                            }
                        }, 10, 200);
                    }
                }
            };
            super.onCreate();

            // reset status variable to initial state
            connectionStatus = MQTTConnectionStatus.INITIAL;

            // create a binder that will let the Activity UI send
            //   commands to the Service
            mBinder = new LocalBinder<MQTTService>(this);

            // register to be notified whenever the user changes their preferences
            //  relating to background data use - so that we can respect the current
            //  preference
            dataEnabledReceiver = new BackgroundDataChangeIntentReceiver();
            registerReceiver(dataEnabledReceiver,
                             new
IntentFilter(ConnectivityManager.ACTION_BACKGROUND_DATA_SETTING_CHANGED));

            // define the connection to the broker
            defineConnectionToBroker(brokerHostName);
        }


        @Override
        public void onStart(final Intent intent, final int startId)
        {
            // This is the old onStart method that will be called on the pre-2.0
            // platform.  On 2.0 or later we override onStartCommand() so this
            // method will not be called.

            new Thread(new Runnable() {
                @Override
                public void run() {
                    handleStart(intent, startId);
                }
            }, "MQTTservice").start();
        }
        @Override
        public int onStartCommand(final Intent intent, int flags, final int startId)
        {
            IntentFilter allowed_intents = new
IntentFilter("com.owncloud.preferences.update_mqtt");
            allowed_intents.addAction("com.mqttservice.restart");
            LocalBroadcastManager.getInstance(this).registerReceiver((receiver),
                    new IntentFilter(allowed_intents));
            new Thread(new Runnable() {
                @Override
```

```
            public void run() {
                handleStart(intent, startId);
            }
        }, "MQTTservice").start();

        // return START_NOT_STICKY - we want this Service to be left running
        //  unless explicitly stopped, and it's process is killed, we want it to
        //  be restarted
        return START_STICKY;
    }
    synchronized void handleStart(Intent intent, int startId)
    {
        // before we start - check for a couple of reasons why we should stop

        if (mqttClient == null)
        {
            // we were unable to define the MQTT client connection, so we stop
            //  immediately - there is nothing that we can do
            stopSelf();
            return;
        }

        ConnectivityManager cm =
(ConnectivityManager)getSystemService(CONNECTIVITY_SERVICE);
        if (cm.getBackgroundDataSetting() == false) // respect the user's request not
to use data!
        {
            // user has disabled background data
            connectionStatus = MQTTConnectionStatus.NOTCONNECTED_DATADISABLED;

            // update the app to show that the connection has been disabled
            broadcastServiceStatus("Not connected - background data disabled");

            // we have a listener running that will notify us when this
            //   preference changes, and will call handleStart again when it
            //   is - letting us pick up where we leave off now
            return;
        }

        // the Activity UI has started the MQTT service - this may be starting
        //  the Service new for the first time, or after the Service has been
        //  running for some time (multiple calls to startService don't start
        //  multiple Services, but it does call this method multiple times)
        // if we have been running already, we re-send any stored data
        rebroadcastStatus();
        rebroadcastReceivedMessages();

        // if the Service was already running and we're already connected - we
        //   don't need to do anything
        if (isAlreadyConnected() == false)
        {
            // set the status to show we're trying to connect
            connectionStatus = MQTTConnectionStatus.CONNECTING;


            // before we attempt to connect - we check if the phone has a
            //  working data connection
            if (isOnline())
            {
                // we think we have an Internet connection, so try to connect
                //  to the message broker
```

```
                    if (connectToBroker())
                    {
                        // we subscribe to a topic - registering to receive push
                        //  notifications with a particular key
                        // in a 'real' app, you might want to subscribe to multiple
                        //  topics - I'm just subscribing to one as an example
                        // note that this topicName could include a wildcard, so
                        //  even just with one subscription, we could receive
                        //  messages for multiple topics
                        subscribeToTopic(topicName);
                    }
                }
                else
                {
                    // we can't do anything now because we don't have a working
                    //  data connection
                    connectionStatus =
MQTTConnectionStatus.NOTCONNECTED_WAITINGFORINTERNET;

                    // inform the app that we are not connected
                    broadcastServiceStatus("Waiting for network connection");
                }
            }

            // changes to the phone's network - such as bouncing between WiFi
            //  and mobile data networks - can break the MQTT connection
            // the MQTT connectionLost can be a bit slow to notice, so we use
            //  Android's inbuilt notification system to be informed of
            //  network changes - so we can reconnect immediately, without
            //  haing to wait for the MQTT timeout
            if (netConnReceiver == null)
            {
                netConnReceiver = new NetworkConnectionIntentReceiver();
                registerReceiver(netConnReceiver,
                                 new
IntentFilter(ConnectivityManager.CONNECTIVITY_ACTION));

            }

            // creates the intents that are used to wake up the phone when it is
            //  time to ping the server
            if (pingSender == null)
            {
                pingSender = new PingSender();
                registerReceiver(pingSender, new IntentFilter(MQTT_PING_ACTION));
            }
        }
        @Override
        public void onDestroy()
        {
            LocalBroadcastManager.getInstance(this).unregisterReceiver(receiver);
            super.onDestroy();
            // disconnect immediately
            disconnectFromBroker();
            // inform the app that the app has successfully disconnected
            broadcastServiceStatus("Disconnected");

            // try not to leak the listener
            if (dataEnabledReceiver != null)
            {
                unregisterReceiver(dataEnabledReceiver);
```

```
            dataEnabledReceiver = null;
        }

        if (mBinder != null) {
            mBinder.close();
            mBinder = null;
        }
        if(RESTART){
            startService(new Intent(this,MQTTService.class));
            Log.v(TAG,"Mqtt service should restart");
        }
    }


    /**********************************************************************/
    /*    METHODS - broadcasts and notifications                          */
    /**********************************************************************/

    // methods used to notify the Activity UI of something that has happened
    //  so that it can be updated to reflect status and the data received
    //  from the server

    private void broadcastServiceStatus(String statusDescription)
    {
        // inform the app (for times when the Activity UI is running /
        //   active) of the current MQTT connection status so that it
        //   can update the UI accordingly
        Intent broadcastIntent = new Intent();
        broadcastIntent.setAction(MQTT_STATUS_INTENT);
        broadcastIntent.putExtra(MQTT_STATUS_MSG, statusDescription);
        sendBroadcast(broadcastIntent);
    }

    private void broadcastReceivedMessage(String topic, String message)
    {
        // pass a message received from the MQTT server on to the Activity UI
        //   (for times when it is running / active) so that it can be displayed
        //   in the app GUI
        Intent broadcastIntent = new Intent();
        broadcastIntent.setAction(MQTT_MSG_RECEIVED_INTENT);
        broadcastIntent.putExtra(MQTT_MSG_RECEIVED_TOPIC, topic);
        broadcastIntent.putExtra(MQTT_MSG_RECEIVED_MSG,   message);
        sendBroadcast(broadcastIntent);
    }

    // methods used to notify the user of what has happened for times when
    //  the app Activity UI isn't running
    private void notifyUser(String alert, String title, String body)
    {
        if (body.equals("ON")) body = "Hard disk has been mounted";
        else if (body.equals("OFF")) body = "Hard disk has been unmounted";
        final String parent_alert = body;
        handler.post(new Runnable() {
            public void run() {

Toast.makeText(getApplicationContext(),parent_alert,Toast.LENGTH_SHORT).show();
            }
        });
    }
```

```
    /**********************************************************************/
    /*    METHODS - binding that allows access from the Actitivy          */
    /**********************************************************************/

    // trying to do local binding while minimizing leaks - code thanks to
    //   Geoff Bruckner - which I found at
    //   http://groups.google.com/group/cw-
android/browse_thread/thread/d026cfa71e48039b/c3b41c728fedd0e7?
show_docid=c3b41c728fedd0e7

    private LocalBinder<MQTTService> mBinder;

    @Override
    public IBinder onBind(Intent intent)
    {
       return mBinder;
    }
    public class LocalBinder<S> extends Binder
    {
        private WeakReference<S> mService;

        public LocalBinder(S service)
        {
            mService = new WeakReference<S>(service);
        }
        public S getService()
        {
            return mService.get();
        }
        public void close()
        {
            mService = null;
        }
    }

    //
    // public methods that can be used by Activities that bind to the Service
    //

    public MQTTConnectionStatus getConnectionStatus()
    {
        return connectionStatus;
    }

    public void rebroadcastStatus()
    {
        String status = "";

        switch (connectionStatus)
        {
            case INITIAL:
                status = "Please wait";
                break;
            case CONNECTING:
                status = "Connecting...";
                break;
            case CONNECTED:
                status = "Connected";
                break;
            case NOTCONNECTED_UNKNOWNREASON:
                status = "Not connected - waiting for network connection";
```

```
                break;
            case NOTCONNECTED_USERDISCONNECT:
                status = "Disconnected";
                break;
            case NOTCONNECTED_DATADISABLED:
                status = "Not connected - background data disabled";
                break;
            case NOTCONNECTED_WAITINGFORINTERNET:
                status = "Unable to connect";
                break;
        }

        //
        // inform the app that the Service has successfully connected
        broadcastServiceStatus(status);
    }

    public void disconnect()
    {
        disconnectFromBroker();
        // set status
        connectionStatus = MQTTConnectionStatus.NOTCONNECTED_USERDISCONNECT;

        // inform the app that the app has successfully disconnected
        broadcastServiceStatus("Disconnected");
    }


    /************************************************************************/
    /*    METHODS - MQTT methods inherited from MQTT classes                */
    /************************************************************************/

    /*
     * callback - method called when we no longer have a connection to the
     *  message broker server
     */
    public void connectionLost() throws Exception
    {
        // we protect against the phone switching off while we're doing this
        //  by requesting a wake lock - we request the minimum possible wake
        //  lock - just enough to keep the CPU running until we've finished
        PowerManager pm = (PowerManager) getSystemService(POWER_SERVICE);
        WakeLock wl = pm.newWakeLock(PowerManager.PARTIAL_WAKE_LOCK, "MQTT");
        wl.acquire();


        //
        // have we lost our data connection?
        //

        if (isOnline() == false)
        {
            connectionStatus = MQTTConnectionStatus.NOTCONNECTED_WAITINGFORINTERNET;

            // inform the app that we are not connected any more
            broadcastServiceStatus("Connection lost - no network connection");

            //
            // inform the user (for times when the Activity UI isn't running)
            //   that we are no longer able to receive messages
            notifyUser("Connection lost - no network connection",
```

```
                        "MQTT", "Connection lost - no network connection");
                //
                // wait until the phone has a network connection again, when we
                //  the network connection receiver will fire, and attempt another
                //  connection to the broker
            }
            else
            {
                //
                // we are still online
                //   the most likely reason for this connectionLost is that we've
                //   switched from wifi to cell, or vice versa
                //   so we try to reconnect immediately
                //

                connectionStatus = MQTTConnectionStatus.NOTCONNECTED_UNKNOWNREASON;

                // inform the app that we are not connected any more, and are
                //   attempting to reconnect
                broadcastServiceStatus("Connection lost - reconnecting...");

                // try to reconnect
                if (connectToBroker()) {
                    subscribeToTopic(topicName);
                }
            }

            // we're finished - if the phone is switched off, it's okay for the CPU
            //  to sleep now
            wl.release();
        }

        /*
         *   callback - called when we receive a message from the server
         */
        public void publishArrived(String topic, byte[] payloadbytes, int qos, boolean
retained)
        {
            // we protect against the phone switching off while we're doing this
            //  by requesting a wake lock - we request the minimum possible wake
            //  lock - just enough to keep the CPU running until we've finished
            PowerManager pm = (PowerManager) getSystemService(POWER_SERVICE);
            WakeLock wl = pm.newWakeLock(PowerManager.PARTIAL_WAKE_LOCK, "MQTT");
            wl.acquire();

            //
            //  I'm assuming that all messages I receive are being sent as strings
            //   this is not an MQTT thing - just me making as assumption about what
            //   data I will be receiving - your app doesn't have to send/receive
            //   strings - anything that can be sent as bytes is valid
            String messageBody = new String(payloadbytes);
            //
            //  for times when the app's Activity UI is not running, the Service
            //   will need to safely store the data that it receives
            if (addReceivedMessageToStore(topic, messageBody))
            {
                // this is a new message - a value we haven't seen before

                //
                // inform the app (for times when the Activity UI is running) of the
                //   received message so the app UI can be updated with the new data
```

```
            broadcastReceivedMessage(topic, messageBody);

            //
            // inform the user (for times when the Activity UI isn't running)
            //   that there is new data available
            notifyUser("New data received", topic, messageBody);
        }

        // receiving this message will have kept the connection alive for us, so
        //  we take advantage of this to postpone the next scheduled ping
        scheduleNextPing();
        // we're finished - if the phone is switched off, it's okay for the CPU
        //  to sleep now
        wl.release();
        Intent reception_of_message = new Intent(MQTTService.this,
FileDisplayActivity.class);
        reception_of_message.setAction("com.mqttservice.message_received");
        reception_of_message.putExtra("message_received",messageBody);
        LocalBroadcastManager.getInstance(this).sendBroadcast(reception_of_message);

    }

    /*********************************************************************/
    /*    METHODS - wrappers for some of the MQTT methods that we use       */
    /*********************************************************************/

    /*
     * Create a client connection object that defines our connection to a
     *   message broker server
     */
    private void defineConnectionToBroker(String brokerHostName)
    {
        String mqttConnSpec = "tcp://" + brokerHostName + "@" + brokerPortNumber;

        try
        {
            // define the connection to the broker
            mqttClient = MqttClient.createMqttClient(mqttConnSpec, usePersistence);
            // register this client app has being able to receive messages
            mqttClient.registerSimpleHandler(this);
        }
        catch (MqttException e)
        {
            // something went wrong!
            mqttClient = null;
            connectionStatus = MQTTConnectionStatus.NOTCONNECTED_UNKNOWNREASON;

            //
            // inform the app that we failed to connect so that it can update
            //  the UI accordingly
            broadcastServiceStatus("Invalid connection parameters");
            //
            // inform the user (for times when the Activity UI isn't running)
            //   that we failed to connect
            notifyUser("Unable to connect", "MQTT", "Unable to connect");
        }
    }

    /*
     * (Re-)connect to the message broker
     */
```

```
    private boolean connectToBroker()
    {
        try
        {
            // try to connect
            mqttClient.connect(generateClientId(), cleanStart, keepAliveSeconds);
            //
            // inform the app that the app has successfully connected
            broadcastServiceStatus("Connected");

            // we are connected
            connectionStatus = MQTTConnectionStatus.CONNECTED;
            // we need to wake up the phone's CPU frequently enough so that the
            //  keep alive messages can be sent
            // we schedule the first one of these now
            scheduleNextPing();
            return true;
        }
        catch (MqttException e)
        {
            // something went wrong!

            connectionStatus = MQTTConnectionStatus.NOTCONNECTED_UNKNOWNREASON;

            //
            // inform the app that we failed to connect so that it can update
            //  the UI accordingly
            broadcastServiceStatus("Unable to connect");
            //
            // inform the user (for times when the Activity UI isn't running)
            //   that we failed to connect
            notifyUser("Unable to connect", "MQTT", "Unable to connect - will retry
later. Please ensure that you have inserted a valid MQTT broker IP address and a
topic");

            // if something has failed, we wait for one keep-alive period before
            //   trying again
            // in a real implementation, you would probably want to keep count
            //  of how many times you attempt this, and stop trying after a
            //  certain number, or length of time - rather than keep trying
            //  forever.
            // a failure is often an intermittent network issue, however, so
            //  some limited retry is a good idea
            scheduleNextPing();

            return false;
        }
    }

    /*
     * Ashu Publish Message to a Topic
     *
     */

    public  void publishMessageToTopic(String message)
    {
       Boolean retained = false;
        if (isAlreadyConnected() == false)
        {
            pending_message = message;
            Log.d(TAG, "mqtt, Unable to publish as we are not connected");
```

```
    }
   else
   {
      try
      {

         Log.d(TAG, "MQTT Publish Message Rcvd: " + message);


         // String[] tps = { topicName };
         // MqttPayload msg = new MqttPayload(message.getBytes());
         byte[] msg = message.getBytes();


         // mqttClient.publish(RESP_TOPIC, msg, 0, false);
         mqttClient.publish(topicName, msg, 0, false);
         //subscribed = true;

      }
      catch (MqttNotConnectedException e)
        {
            Log.e("mqtt", "subscribe failed - MQTT not connected", e);
        }
        catch (IllegalArgumentException e)
        {
            Log.e("mqtt", "subscribe failed - illegal argument", e);
        }
        catch (MqttException e)
        {
            Log.e("mqtt", "subscribe failed - MQTT exception", e);
        }
   }
}

/*
 * Send a request to the message broker to be sent messages published with
 *  the specified topic name. Wildcards are allowed.
 */
private void subscribeToTopic(String topicName)
{
    boolean subscribed = false;

    if (isAlreadyConnected() == false)
    {
        // quick sanity check - don't try and subscribe if we
        //  don't have a connection

        Log.e("mqtt", "Unable to subscribe as we are not connected");
    }
    else
    {
        try
        {
            String[] topics = { topicName };
            mqttClient.subscribe(topics, qualitiesOfService);

            subscribed = true;
        }
        catch (MqttNotConnectedException e)
        {
            Log.e("mqtt", "subscribe failed - MQTT not connected", e);
```

```
            }
            catch (IllegalArgumentException e)
            {
                Log.e("mqtt", "subscribe failed - illegal argument", e);
            }
            catch (MqttException e)
            {
                Log.e("mqtt", "subscribe failed - MQTT exception", e);
            }
        }

        if (subscribed == false)
        {
            //
            // inform the app of the failure to subscribe so that the UI can
            //  display an error
            broadcastServiceStatus("Unable to subscribe");
            //
            // inform the user (for times when the Activity UI isn't running)
            notifyUser("Unable to subscribe", "MQTT", "Unable to subscribe");
        }
    }

    /*
     * Terminates a connection to the message broker.
     */
    private void disconnectFromBroker()
    {
        // if we've been waiting for an Internet connection, this can be
        //  cancelled - we don't need to be told when we're connected now
        try
        {
            if (netConnReceiver != null)
            {
                unregisterReceiver(netConnReceiver);
                netConnReceiver = null;
            }

            if (pingSender != null)
            {
                unregisterReceiver(pingSender);
                pingSender = null;
            }
        }
        catch (Exception eee)
        {
            // probably because we hadn't registered it
            Log.e("mqtt", "unregister failed", eee);
        }
        try
        {
            if (mqttClient != null)
            {
                mqttClient.disconnect();
            }
        }
        catch (MqttPersistenceException e)
        {
            Log.e("mqtt", "disconnect failed - persistence exception", e);
        }
        finally
```

```
        {
            mqttClient = null;
        }

        // we can now remove the ongoing notification that warns users that
        //  there was a long-running ongoing service running
        NotificationManager nm = (NotificationManager)
getSystemService(NOTIFICATION_SERVICE);
        nm.cancelAll();
    }

    /*
     * Checks if the MQTT client thinks it has an active connection
     */
    private boolean isAlreadyConnected()
    {
        return ((mqttClient != null) && (mqttClient.isConnected() == true));
    }

    private class BackgroundDataChangeIntentReceiver extends BroadcastReceiver
    {
        @Override
        public void onReceive(Context ctx, Intent intent)
        {
            // we protect against the phone switching off while we're doing this
            //  by requesting a wake lock - we request the minimum possible wake
            //  lock - just enough to keep the CPU running until we've finished
            PowerManager pm = (PowerManager) getSystemService(POWER_SERVICE);
            WakeLock wl = pm.newWakeLock(PowerManager.PARTIAL_WAKE_LOCK, "MQTT");
            wl.acquire();

            ConnectivityManager cm =
(ConnectivityManager)getSystemService(CONNECTIVITY_SERVICE);
            if (cm.getBackgroundDataSetting())
            {
                // user has allowed background data - we start again - picking
                //  up where we left off in handleStart before
                defineConnectionToBroker(brokerHostName);
                handleStart(intent, 0);
            }
            else
            {
                // user has disabled background data
                connectionStatus = MQTTConnectionStatus.NOTCONNECTED_DATADISABLED;

                // update the app to show that the connection has been disabled
                broadcastServiceStatus("Not connected - background data disabled");

                // disconnect from the broker
                disconnectFromBroker();
            }

            // we're finished - if the phone is switched off, it's okay for the CPU
            //  to sleep now
            wl.release();
        }
    }


    /*
     * Called in response to a change in network connection - after losing a
```

```
     *  connection to the server, this allows us to wait until we have a usable
     *  data connection again
     */
    private class NetworkConnectionIntentReceiver extends BroadcastReceiver
    {
        @Override
        public void onReceive(Context ctx, Intent intent)
        {
            // we protect against the phone switching off while we're doing this
            //  by requesting a wake lock - we request the minimum possible wake
            //  lock - just enough to keep the CPU running until we've finished
            PowerManager pm = (PowerManager) getSystemService(POWER_SERVICE);
            WakeLock wl = pm.newWakeLock(PowerManager.PARTIAL_WAKE_LOCK, "MQTT");
            wl.acquire();

            if (isOnline())
            {
                // we have an internet connection - have another try at connecting
                if (connectToBroker())
                {
                    // we subscribe to a topic - registering to receive push
                    //  notifications with a particular key
                    subscribeToTopic(topicName);
                }
            }

            // we're finished - if the phone is switched off, it's okay for the CPU
            //  to sleep now
            wl.release();
        }
    }

    /*
     * Schedule the next time that you want the phone to wake up and ping the
     *  message broker server
     */
    private void scheduleNextPing()
    {
        // When the phone is off, the CPU may be stopped. This means that our
        //   code may stop running.
        // When connecting to the message broker, we specify a 'keep alive'
        //   period - a period after which, if the client has not contacted
        //   the server, even if just with a ping, the connection is considered
        //   broken.
        // To make sure the CPU is woken at least once during each keep alive
        //   period, we schedule a wake up to manually ping the server
        //   thereby keeping the long-running connection open
        // Normally when using this Java MQTT client library, this ping would be
        //   handled for us.
        // Note that this may be called multiple times before the next scheduled
        //   ping has fired. This is good - the previously scheduled one will be
        //   cancelled in favour of this one.
        // This means if something else happens during the keep alive period,
        //   (e.g. we receive an MQTT message), then we start a new keep alive
        //   period, postponing the next ping.

        PendingIntent pendingIntent = PendingIntent.getBroadcast(this, 0,
                                                                 new
Intent(MQTT_PING_ACTION),

PendingIntent.FLAG_UPDATE_CURRENT);
```

```
        // in case it takes us a little while to do this, we try and do it
        //  shortly before the keep alive period expires
        // it means we're pinging slightly more frequently than necessary
        Calendar wakeUpTime = Calendar.getInstance();
        wakeUpTime.add(Calendar.SECOND, keepAliveSeconds);

        AlarmManager aMgr = (AlarmManager) getSystemService(ALARM_SERVICE);
        aMgr.set(AlarmManager.RTC_WAKEUP,
                 wakeUpTime.getTimeInMillis(),
                 pendingIntent);
    }


    /*
     * Used to implement a keep-alive protocol at this Service level - it sends
     *  a PING message to the server, then schedules another ping after an
     *  interval defined by keepAliveSeconds
     */
    public class PingSender extends BroadcastReceiver
    {
        @Override
        public void onReceive(Context context, Intent intent)
        {
            // Note that we don't need a wake lock for this method (even though
            //  it's important that the phone doesn't switch off while we're
            //  doing this).
            // According to the docs, "Alarm Manager holds a CPU wake lock as
            //  long as the alarm receiver's onReceive() method is executing.
            //  This guarantees that the phone will not sleep until you have
            //  finished handling the broadcast."
            // This is good enough for our needs.

            try
            {
                mqttClient.ping();
            }
            catch (MqttException e)
            {
                // if something goes wrong, it should result in connectionLost
                //  being called, so we will handle it there
                Log.e("mqtt", "ping failed - MQTT exception", e);

                // assume the client connection is broken - trash it
                try {
                    mqttClient.disconnect();
                }
                catch (MqttPersistenceException e1) {
                    Log.e("mqtt", "disconnect failed - persistence exception", e1);
                }

                // reconnect
                if (connectToBroker()) {
                    subscribeToTopic(topicName);
                }
            }
            // start the next keep alive period
            scheduleNextPing();
        }
    }
```

```
/*********************************************************************/
/*   APP SPECIFIC - stuff that would vary for different uses of MQTT    */
/*********************************************************************/

//  apps that handle very small amounts of data - e.g. updates and
//   notifications that don't need to be persisted if the app / phone
//   is restarted etc. may find it acceptable to store this data in a
//   variable in the Service
//  that's what I'm doing in this sample: storing it in a local hashtable
//  if you are handling larger amounts of data, and/or need the data to
//   be persisted even if the app and/or phone is restarted, then
//   you need to store the data somewhere safely
//  see http://developer.android.com/guide/topics/data/data-storage.html
//   for your storage options - the best choice depends on your needs

// stored internally

private Hashtable<String, String> dataCache = new Hashtable<String, String>();
private boolean addReceivedMessageToStore(String key, String value)
{
    String previousValue = null;

    if (value.length() == 0)
    {
        previousValue = dataCache.remove(key);
    }
    else
    {
        previousValue = dataCache.put(key, value);
    }

    // is this a new value? or am I receiving something I already knew?
    //  we return true if this is something new
    return ((previousValue == null) ||
            (previousValue.equals(value) == false));
}
// provide a public interface, so Activities that bind to the Service can
//  request access to previously received messages

public void rebroadcastReceivedMessages()
{
    Enumeration<String> e = dataCache.keys();
    while(e.hasMoreElements())
    {
        String nextKey = e.nextElement();
        String nextValue = dataCache.get(nextKey);

        broadcastReceivedMessage(nextKey, nextValue);
    }
}

/*********************************************************************/
/*    METHODS - internal utility methods                             */
/*********************************************************************/

private String generateClientId()
{
    // generate a unique client id if we haven't done so before, otherwise
    //   re-use the one we already have
    if (mqttClientId == null)
```

```
        {
            // generate a unique client ID - I'm basing this on a combination of
            //  the phone device id and the current timestamp
            String timestamp = "" + (new Date()).getTime();
            String android_id = Settings.System.getString(getContentResolver(),
                                                          Secure.ANDROID_ID);
            mqttClientId = timestamp + android_id;

            // truncate - MQTT spec doesn't allow client ids longer than 23 chars
            if (mqttClientId.length() > MAX_MQTT_CLIENTID_LENGTH) {
                mqttClientId = mqttClientId.substring(0, MAX_MQTT_CLIENTID_LENGTH);
            }
        }

        return mqttClientId;
    }

    private boolean isOnline()
    {
        ConnectivityManager cm =
(ConnectivityManager)getSystemService(CONNECTIVITY_SERVICE);
        if(cm.getActiveNetworkInfo() != null &&
           cm.getActiveNetworkInfo().isAvailable() &&
           cm.getActiveNetworkInfo().isConnected())
        {
            return true;
        }

        return false;
    }
}
```

11.7 android/src/com/owncloud/android/ui/activity/Preferences.java

```
/**
 *   ownCloud Android client application
 *
 *   @author Bartek Przybylski
 *   @author David A. Velasco
 *   Copyright (C) 2011  Bartek Przybylski
 *   Copyright (C) 2016 ownCloud Inc.
 *
 *   This program is free software: you can redistribute it and/or modify
 *   it under the terms of the GNU General Public License version 2,
 *   as published by the Free Software Foundation.
 *
 *   This program is distributed in the hope that it will be useful,
 *   but WITHOUT ANY WARRANTY; without even the implied warranty of
 *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 *   GNU General Public License for more details.
 *
 *   You should have received a copy of the GNU General Public License
 *   along with this program.  If not, see <http://www.gnu.org/licenses/>.
 *
 */
package com.owncloud.android.ui.activity;
import android.accounts.Account;
import android.accounts.AccountManager;
import android.accounts.AccountManagerCallback;
```

```
import android.accounts.AccountManagerFuture;
import android.app.Activity;
import android.content.ComponentName;
import android.content.Context;
import android.content.Intent;
import android.content.ServiceConnection;
import android.content.SharedPreferences;
import android.content.pm.PackageInfo;
import android.content.pm.PackageManager.NameNotFoundException;
import android.content.res.Configuration;
import android.net.Uri;
import android.os.Bundle;
import android.os.Handler;
import android.os.IBinder;
import android.preference.CheckBoxPreference;
import android.preference.EditTextPreference;
import android.preference.Preference;
import android.preference.Preference.OnPreferenceChangeListener;
import android.preference.Preference.OnPreferenceClickListener;
import android.preference.PreferenceActivity;
import android.preference.PreferenceCategory;
import android.preference.PreferenceManager;
import android.support.annotation.LayoutRes;
import android.support.annotation.Nullable;
import android.support.v4.content.LocalBroadcastManager;
import android.support.v7.app.ActionBar;
import android.support.v7.app.AlertDialog;
import android.support.v7.app.AppCompatDelegate;
import android.support.v7.widget.Toolbar;
import android.util.Log;
import android.view.ContextMenu;
import android.view.ContextMenu.ContextMenuInfo;
import android.view.Menu;
import android.view.MenuInflater;
import android.view.MenuItem;
import android.view.View;
import android.view.ViewGroup;
import android.widget.AdapterView;
import android.widget.AdapterView.OnItemLongClickListener;
import android.widget.ArrayAdapter;
import android.widget.ListAdapter;
import android.widget.ListView;
import android.widget.Toast;
import com.owncloud.android.BuildConfig;
import com.owncloud.android.MainApp;
import com.owncloud.android.R;
import com.owncloud.android.authentication.AccountUtils;
import com.owncloud.android.authentication.AuthenticatorActivity;
import com.owncloud.android.datamodel.FileDataStorageManager;
import com.owncloud.android.datamodel.OCFile;
import com.owncloud.android.files.FileOperationsHelper;
import com.owncloud.android.files.services.FileDownloader;
import com.owncloud.android.files.services.FileUploader;
import com.owncloud.android.lib.common.OwnCloudAccount;
import com.owncloud.android.lib.common.utils.Log_OC;
import com.owncloud.android.services.MQTTService;
import com.owncloud.android.services.OperationsService;
import com.owncloud.android.ui.RadioButtonPreference;
import com.owncloud.android.utils.DisplayUtils;
import com.owncloud.android.utils.IPAddressPreference;
/**
 * An Activity that allows the user to change the application's settings.
 *
 * It proxies the necessary calls via {@link android.support.v7.app.AppCompatDelegate}
to be used
 * with AppCompat.
```

```
 */
public class Preferences extends PreferenceActivity
        implements AccountManagerCallback<Boolean>, ComponentsGetter {

    private static final String TAG = Preferences.class.getSimpleName();
    private static final int ACTION_SELECT_UPLOAD_PATH = 1;
    private static final int ACTION_SELECT_UPLOAD_VIDEO_PATH = 2;
    private static final int ACTION_REQUEST_PASSCODE = 5;
    private static final int ACTION_CONFIRM_PASSCODE = 6;
    private CheckBoxPreference pCode;
    private Preference pAboutApp;
    private AppCompatDelegate mDelegate;
    private PreferenceCategory mAccountsPrefCategory = null;
    private final Handler mHandler = new Handler();
    private String mAccountName;
    private boolean mShowContextMenu = false;
    private String mUploadPath;
    private PreferenceCategory mPrefInstantUploadCategory;
    private Preference mPrefInstantUpload;
    private Preference mPrefInstantUploadBehaviour;
    private Preference mPrefInstantUploadPath;
    private Preference mPrefInstantUploadPathWiFi;
    private Preference mPrefInstantVideoUpload;
    private Preference mPrefInstantVideoUploadPath;
    private Preference mPrefInstantVideoUploadPathWiFi;
    private String mUploadVideoPath;
    protected FileDownloader.FileDownloaderBinder mDownloaderBinder = null;
    protected FileUploader.FileUploaderBinder mUploaderBinder = null;
    private ServiceConnection mDownloadServiceConnection, mUploadServiceConnection =
null;
    LocalBroadcastManager broadcaster;
    private IPAddressPreference mPrefBroker;
    private EditTextPreference mPrefTopic;
    private String mBrokerAddress;
    private String mTopic;
    @SuppressWarnings("deprecation")
    @Override
    public void onCreate(Bundle savedInstanceState) {
        getDelegate().installViewFactory();
        getDelegate().onCreate(savedInstanceState);
        super.onCreate(savedInstanceState);
        addPreferencesFromResource(R.xml.preferences);
        ActionBar actionBar = getSupportActionBar();
        actionBar.setDisplayHomeAsUpEnabled(true);
        actionBar.setTitle(R.string.actionbar_settings);
        // For adding content description tag to a title field in the action bar
        int actionBarTitleId = getResources().getIdentifier("action_bar_title", "id",
"android");
        View actionBarTitleView =
getWindow().getDecorView().findViewById(actionBarTitleId);
        if (actionBarTitleView != null) {    // it's null in Android 2.x
            getWindow().getDecorView().findViewById(actionBarTitleId).
                    setContentDescription(getString(R.string.actionbar_settings));
        }
        final Activity current_preference_activity = this;
        mPrefBroker = (IPAddressPreference) findPreference("broker");
        mPrefBroker.setOnPreferenceChangeListener(new OnPreferenceChangeListener() {
            @Override
            public boolean onPreferenceChange(Preference preference,Object object) {
                mBrokerAddress = mPrefBroker.getEditText().getText().toString();
                mPrefBroker.setText(mBrokerAddress);
```

```
                SharedPreferences preferences =
PreferenceManager.getDefaultSharedPreferences(getApplicationContext());
                SharedPreferences.Editor editor = preferences.edit();
                editor.putString("brokerHostName", mBrokerAddress); // αποθήκευση της
αλλαγμένης διεύθυνσης του Broker.
                editor.commit();
                Intent intent = new Intent (Preferences.this, MQTTService.class);
                intent.setAction("com.owncloud.preferences.update_mqtt");
                intent.putExtra("brokerHostName",mBrokerAddress);
                intent.putExtra("topicName",mTopic);

LocalBroadcastManager.getInstance(current_preference_activity).sendBroadcast(intent);
                Log.v(TAG,"Just sent the new broker address "+mBrokerAddress);
                return false;
            }
        });
        mPrefTopic = (EditTextPreference) findPreference("topic");
        mPrefTopic.setOnPreferenceChangeListener(new OnPreferenceChangeListener() {
            @Override
            public boolean onPreferenceChange(Preference preference,Object object) {
                //mPrefBroker.getSharedPreferences().getString(mBrokerAddress,null);
                mTopic = mPrefTopic.getEditText().getText().toString();
                mPrefTopic.setText(mTopic);
                SharedPreferences preferences =
PreferenceManager.getDefaultSharedPreferences(getApplicationContext());
                SharedPreferences.Editor editor = preferences.edit();
                editor.putString("topicName", mTopic); // Αποθήκευση του ανανεωμένου
θέματος MQTT
                editor.commit();
                Intent intent = new Intent (Preferences.this, MQTTService.class);
                intent.setAction("com.owncloud.preferences.update_mqtt");
                intent.putExtra("brokerHostName",mBrokerAddress);
                intent.putExtra("topicName",mTopic);

LocalBroadcastManager.getInstance(current_preference_activity).sendBroadcast(intent);
                Log.v(TAG,"Just sent the topic, which is "+mTopic);
                return false;
            }
        });
        // Load the accounts category for adding the list of accounts
        mAccountsPrefCategory = (PreferenceCategory)
findPreference("accounts_category");
        ListView listView = getListView();
        listView.setOnItemLongClickListener(new OnItemLongClickListener() {
            @Override
            public boolean onItemLongClick(AdapterView<?> parent, View view, int
position, long id) {
                ListView listView = (ListView) parent;
                ListAdapter listAdapter = listView.getAdapter();
                Object obj = listAdapter.getItem(position);
                if (obj != null && obj instanceof RadioButtonPreference) {
                    mShowContextMenu = true;
                    mAccountName = ((RadioButtonPreference) obj).getKey();
                    String[] items = {
                            getResources().getString(R.string.change_password),
                            getResources().getString(R.string.delete_account)
                    };
                    final AlertDialog.Builder alertDialogBuilder = new
AlertDialog.Builder(Preferences.this);
                    View convertView =
getLayoutInflater().inflate(R.layout.alert_dialog_list_view, null);
                    alertDialogBuilder.setView(convertView);
```

```
                        ListView lv = (ListView) convertView.findViewById(R.id.list);
                        ArrayAdapter<String> adapter = new ArrayAdapter<String>(
                                Preferences.this,R.layout.simple_dialog_list_item,items);
                        lv.setAdapter(adapter);
                        //Setup proper inline listener
                        final AlertDialog alertDialog = alertDialogBuilder.create();
                        lv.setOnItemClickListener(new AdapterView.OnItemClickListener() {
                            @Override
                            public void onItemClick(AdapterView<?> parent, View view, int
position, long id) {
                                AccountManager am = (AccountManager)
getSystemService(ACCOUNT_SERVICE);
                                Account accounts[] =
am.getAccountsByType(MainApp.getAccountType());
                                for (Account a : accounts) {
                                    if (a.name.equals(mAccountName)) {
                                        if (position==0) {
                                            // Change account password
                                            Intent updateAccountCredentials = new
Intent(Preferences.this, AuthenticatorActivity.class);

updateAccountCredentials.putExtra(AuthenticatorActivity.EXTRA_ACCOUNT, a);

updateAccountCredentials.putExtra(AuthenticatorActivity.EXTRA_ACTION,

AuthenticatorActivity.ACTION_UPDATE_TOKEN);
                                            startActivity(updateAccountCredentials);
                                            alertDialog.cancel();

                                        } else if (position==1) {
                                            // Remove account
                                            am.removeAccount(a, Preferences.this,
mHandler);
                                            Log_OC.d(TAG, "Remove an account " +
a.name);
                                            alertDialog.cancel();
                                        }
                                    }
                                }
                            }
                        });
                        alertDialog.show();
                        View.OnLongClickListener longListener = (View.OnLongClickListener)
obj;
                        return longListener.onLongClick(view);
                    }
                    return false;
                }
            });

            // Load package info
            String temp;
            try {
                PackageInfo pkg = getPackageManager().getPackageInfo(getPackageName(), 0);
                temp = pkg.versionName;
            } catch (NameNotFoundException e) {
                temp = "";
                Log_OC.e(TAG, "Error while showing about dialog", e);
            }
            final String appVersion = temp;

            // Register context menu for list of preferences.
```

```
        registerForContextMenu(getListView());
        pCode = (CheckBoxPreference)
findPreference(PassCodeActivity.PREFERENCE_SET_PASSCODE);
        if (pCode != null){
            pCode.setOnPreferenceChangeListener(new OnPreferenceChangeListener() {
                @Override
                public boolean onPreferenceChange(Preference preference, Object
newValue) {
                    Intent i = new Intent(getApplicationContext(),
PassCodeActivity.class);
                    Boolean incoming = (Boolean) newValue;
                    i.setAction(
                            incoming.booleanValue() ?
PassCodeActivity.ACTION_REQUEST_WITH_RESULT :
                                    PassCodeActivity.ACTION_CHECK_WITH_RESULT
                    );
                    startActivityForResult(i, incoming.booleanValue() ?
ACTION_REQUEST_PASSCODE :
                            ACTION_CONFIRM_PASSCODE);
                    // Don't update just yet, we will decide on it in onActivityResult
                    return false;
                }
            });

        }
        PreferenceCategory preferenceCategory = (PreferenceCategory)
findPreference("more");

        boolean helpEnabled = getResources().getBoolean(R.bool.help_enabled);
        Preference pHelp =  findPreference("help");
        if (pHelp != null ){
            if (helpEnabled) {
                pHelp.setOnPreferenceClickListener(new OnPreferenceClickListener() {
                    @Override
                    public boolean onPreferenceClick(Preference preference) {
                        String helpWeb   =(String) getText(R.string.url_help);
                        if (helpWeb != null && helpWeb.length() > 0) {
                            Uri uriUrl = Uri.parse(helpWeb);
                            Intent intent = new Intent(Intent.ACTION_VIEW, uriUrl);
                            startActivity(intent);
                        }
                        return true;
                    }
                });
            } else {
                preferenceCategory.removePreference(pHelp);
            }

        }

       boolean recommendEnabled = getResources().getBoolean(R.bool.recommend_enabled);
       Preference pRecommend =  findPreference("recommend");
        if (pRecommend != null){
            if (recommendEnabled) {
                pRecommend.setOnPreferenceClickListener(new OnPreferenceClickListener()
{
                    @Override
                    public boolean onPreferenceClick(Preference preference) {
                        Intent intent = new Intent(Intent.ACTION_SENDTO);
                        intent.setType("text/plain");
                        intent.setData(Uri.parse(getString(R.string.mail_recommend)));
                        intent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
```

```
                        String appName = getString(R.string.app_name);
                        String downloadUrl = getString(R.string.url_app_download);
                        String recommendSubject =
                                String.format(getString(R.string.recommend_subject),
                                appName);
                        String recommendText =
String.format(getString(R.string.recommend_text),
                                appName, downloadUrl);

                        intent.putExtra(Intent.EXTRA_SUBJECT, recommendSubject);
                        intent.putExtra(Intent.EXTRA_TEXT, recommendText);
                        startActivity(intent);
                        return(true);
                    }
                });
            } else {
                preferenceCategory.removePreference(pRecommend);
            }

        }

        boolean feedbackEnabled = getResources().getBoolean(R.bool.feedback_enabled);
        Preference pFeedback =  findPreference("feedback");
        if (pFeedback != null){
            if (feedbackEnabled) {
                pFeedback.setOnPreferenceClickListener(new OnPreferenceClickListener()
{
                    @Override
                    public boolean onPreferenceClick(Preference preference) {
                        String feedbackMail   =(String)
getText(R.string.mail_feedback);
                        String feedback   =(String) getText(R.string.prefs_feedback) +
                                " - android v" + appVersion;
                        Intent intent = new Intent(Intent.ACTION_SENDTO);
                        intent.setType("text/plain");
                        intent.putExtra(Intent.EXTRA_SUBJECT, feedback);

                        intent.setData(Uri.parse(feedbackMail));
                        intent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
                        startActivity(intent);

                        return true;
                    }
                });
            } else {
                preferenceCategory.removePreference(pFeedback);
            }
        }
        boolean loggerEnabled = getResources().getBoolean(R.bool.logger_enabled) ||
BuildConfig.DEBUG;
        Preference pLogger =  findPreference("logger");
        if (pLogger != null){
            if (loggerEnabled) {
                pLogger.setOnPreferenceClickListener(new OnPreferenceClickListener() {
                    @Override
                    public boolean onPreferenceClick(Preference preference) {
                        Intent loggerIntent = new Intent(getApplicationContext(),
LogHistoryActivity.class);
                        startActivity(loggerIntent);
                        return true;
                    }
```

```
                });
            } else {
                preferenceCategory.removePreference(pLogger);
            }
        }
        boolean imprintEnabled = getResources().getBoolean(R.bool.imprint_enabled);
        Preference pImprint =  findPreference("imprint");
        if (pImprint != null) {
            if (imprintEnabled) {
                pImprint.setOnPreferenceClickListener(new OnPreferenceClickListener() {
                    @Override
                    public boolean onPreferenceClick(Preference preference) {
                        String imprintWeb = (String) getText(R.string.url_imprint);
                        if (imprintWeb != null && imprintWeb.length() > 0) {
                            Uri uriUrl = Uri.parse(imprintWeb);
                            Intent intent = new Intent(Intent.ACTION_VIEW, uriUrl);
                            startActivity(intent);
                        }
                        //ImprintDialog.newInstance(true).show(preference.get,
"IMPRINT_DIALOG");
                        return true;
                    }
                });
            } else {
                preferenceCategory.removePreference(pImprint);
            }
        }
        mPrefInstantUploadPath =  findPreference("instant_upload_path");
        if (mPrefInstantUploadPath != null){
            mPrefInstantUploadPath.setOnPreferenceClickListener(new
OnPreferenceClickListener() {
                    @Override
                    public boolean onPreferenceClick(Preference preference) {
                        if (!mUploadPath.endsWith(OCFile.PATH_SEPARATOR)) {
                            mUploadPath += OCFile.PATH_SEPARATOR;
                        }
                        Intent intent = new Intent(Preferences.this,
UploadPathActivity.class);
                        intent.putExtra(UploadPathActivity.KEY_INSTANT_UPLOAD_PATH,
mUploadPath);
                        startActivityForResult(intent, ACTION_SELECT_UPLOAD_PATH);
                        return true;
                    }
                });
        }

        mPrefInstantUploadCategory =
                (PreferenceCategory) findPreference("instant_uploading_category");

        mPrefInstantUploadPathWiFi =  findPreference("instant_upload_on_wifi");
        mPrefInstantUpload = findPreference("instant_uploading");

        toggleInstantPictureOptions(((CheckBoxPreference)
mPrefInstantUpload).isChecked());

        mPrefInstantUpload.setOnPreferenceChangeListener(new
OnPreferenceChangeListener() {

            @Override
            public boolean onPreferenceChange(Preference preference, Object newValue) {
                toggleInstantPictureOptions((Boolean) newValue);
                toggleInstantUploadBehaviour(
```

```
                        ((CheckBoxPreference)mPrefInstantVideoUpload).isChecked(),
                        (Boolean) newValue);
                return true;
            }
        });

        mPrefInstantVideoUploadPath =  findPreference("instant_video_upload_path");
        if (mPrefInstantVideoUploadPath != null){
            mPrefInstantVideoUploadPath.setOnPreferenceClickListener(new
OnPreferenceClickListener() {
                    @Override
                    public boolean onPreferenceClick(Preference preference) {
                        if (!mUploadVideoPath.endsWith(OCFile.PATH_SEPARATOR)) {
                            mUploadVideoPath += OCFile.PATH_SEPARATOR;
                        }
                        Intent intent = new Intent(Preferences.this,
UploadPathActivity.class);
                        intent.putExtra(UploadPathActivity.KEY_INSTANT_UPLOAD_PATH,
                                mUploadVideoPath);
                        startActivityForResult(intent,
ACTION_SELECT_UPLOAD_VIDEO_PATH);
                        return true;
                    }
                });
        }

        mPrefInstantVideoUploadPathWiFi =
findPreference("instant_video_upload_on_wifi");
        mPrefInstantVideoUpload = findPreference("instant_video_uploading");
        toggleInstantVideoOptions(((CheckBoxPreference)
mPrefInstantVideoUpload).isChecked());

        mPrefInstantVideoUpload.setOnPreferenceChangeListener(new
OnPreferenceChangeListener() {
            @Override
            public boolean onPreferenceChange(Preference preference, Object newValue) {
                toggleInstantVideoOptions((Boolean) newValue);
                toggleInstantUploadBehaviour(
                        (Boolean) newValue,
                        ((CheckBoxPreference) mPrefInstantUpload).isChecked());
                return true;
            }
        });
        mPrefInstantUploadBehaviour = findPreference("prefs_instant_behaviour");
        toggleInstantUploadBehaviour(
                ((CheckBoxPreference)mPrefInstantVideoUpload).isChecked(),
                ((CheckBoxPreference)mPrefInstantUpload).isChecked());
        /* About App */
       pAboutApp = (Preference) findPreference("about_app");
       if (pAboutApp != null) {
               pAboutApp.setTitle(String.format(getString(R.string.about_android),
                       getString(R.string.app_name)));
               pAboutApp.setSummary(String.format(getString(R.string.about_version),
appVersion));
       }
       loadInstantUploadPath();
       loadInstantUploadVideoPath();
        /* ComponentsGetter */
        mDownloadServiceConnection = newTransferenceServiceConnection();
        if (mDownloadServiceConnection != null) {
            bindService(new Intent(this, FileDownloader.class),
mDownloadServiceConnection,
```

```
                    Context.BIND_AUTO_CREATE);
        }
        mUploadServiceConnection = newTransferenceServiceConnection();
        if (mUploadServiceConnection != null) {
            bindService(new Intent(this, FileUploader.class), mUploadServiceConnection,
                    Context.BIND_AUTO_CREATE);
        }
    }

    private void toggleInstantPictureOptions(Boolean value){
        if (value){
            mPrefInstantUploadCategory.addPreference(mPrefInstantUploadPathWiFi);
            mPrefInstantUploadCategory.addPreference(mPrefInstantUploadPath);
        } else {
            mPrefInstantUploadCategory.removePreference(mPrefInstantUploadPathWiFi);
            mPrefInstantUploadCategory.removePreference(mPrefInstantUploadPath);
        }
    }

    private void toggleInstantVideoOptions(Boolean value){
        if (value){
            mPrefInstantUploadCategory.addPreference(mPrefInstantVideoUploadPathWiFi);
            mPrefInstantUploadCategory.addPreference(mPrefInstantVideoUploadPath);
        } else {

mPrefInstantUploadCategory.removePreference(mPrefInstantVideoUploadPathWiFi);
            mPrefInstantUploadCategory.removePreference(mPrefInstantVideoUploadPath);
        }
    }
    private void toggleInstantUploadBehaviour(Boolean video, Boolean picture){
        if (picture || video){
            mPrefInstantUploadCategory.addPreference(mPrefInstantUploadBehaviour);
        } else {
            mPrefInstantUploadCategory.removePreference(mPrefInstantUploadBehaviour);
        }
    }
    @Override
    public void onCreateContextMenu(ContextMenu menu, View v, ContextMenuInfo menuInfo)
{
        // Filter for only showing contextual menu when long press on the
        // accounts
        if (mShowContextMenu) {
            getMenuInflater().inflate(R.menu.account_picker_long_click, menu);
            mShowContextMenu = false;
        }
        super.onCreateContextMenu(menu, v, menuInfo);
    }
    @Override
    public void run(AccountManagerFuture<Boolean> future) {
        if (future.isDone()) {
            // after remove account
            Account account = new Account(mAccountName, MainApp.getAccountType());
            if (!AccountUtils.exists(account, MainApp.getAppContext())) {
                // Cancel tranfers
                if (mUploaderBinder != null) {
                    mUploaderBinder.cancel(account);
                }
                if (mDownloaderBinder != null) {
                    mDownloaderBinder.cancel(account);
                }
            }
```

```
            Account a = AccountUtils.getCurrentOwnCloudAccount(this);
            String accountName = "";
            if (a == null) {
                Account[] accounts = AccountManager.get(this)
                        .getAccountsByType(MainApp.getAccountType());
                if (accounts.length != 0)
                    accountName = accounts[0].name;
                AccountUtils.setCurrentOwnCloudAccount(this, accountName);
            }
            addAccountsCheckboxPreferences();
        }
    }
    @Override
    protected void onResume() {
        super.onResume();
        SharedPreferences appPrefs =
                PreferenceManager.getDefaultSharedPreferences(getApplicationContext());
        boolean state = appPrefs.getBoolean(PassCodeActivity.PREFERENCE_SET_PASSCODE,
false);
        pCode.setChecked(state);
        // Populate the accounts category with the list of accounts
        addAccountsCheckboxPreferences();
    }
    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        super.onCreateOptionsMenu(menu);
        return true;
    }
    @Override
    public boolean onMenuItemSelected(int featureId, MenuItem item) {
        super.onMenuItemSelected(featureId, item);
        Intent intent;
        switch (item.getItemId()) {
        case android.R.id.home:
            intent = new Intent(getBaseContext(), FileDisplayActivity.class);
            intent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
            startActivity(intent);
            break;
        default:
            Log_OC.w(TAG, "Unknown menu item triggered");
            return false;
        }
        return true;
    }
    @Override
    protected void onActivityResult(int requestCode, int resultCode, Intent data) {
        super.onActivityResult(requestCode, resultCode, data);
        if (requestCode == ACTION_SELECT_UPLOAD_PATH && resultCode == RESULT_OK){
            OCFile folderToUpload =
                    (OCFile) data.getParcelableExtra(UploadPathActivity.EXTRA_FOLDER);
            mUploadPath = folderToUpload.getRemotePath();
            mUploadPath = DisplayUtils.getPathWithoutLastSlash(mUploadPath);
            // Show the path on summary preference
            mPrefInstantUploadPath.setSummary(mUploadPath);
            saveInstantUploadPathOnPreferences();
        } else if (requestCode == ACTION_SELECT_UPLOAD_VIDEO_PATH && resultCode ==
RESULT_OK){
            OCFile folderToUploadVideo =
                    (OCFile) data.getParcelableExtra(UploadPathActivity.EXTRA_FOLDER);
            mUploadVideoPath = folderToUploadVideo.getRemotePath();
            mUploadVideoPath = DisplayUtils.getPathWithoutLastSlash(mUploadVideoPath);
```

```
                // Show the video path on summary preference
                mPrefInstantVideoUploadPath.setSummary(mUploadVideoPath);
                saveInstantUploadVideoPathOnPreferences();
            } else if (requestCode == ACTION_REQUEST_PASSCODE && resultCode == RESULT_OK) {
                String passcode = data.getStringExtra(PassCodeActivity.KEY_PASSCODE);
                if (passcode != null && passcode.length() == 4) {
                    SharedPreferences.Editor appPrefs = PreferenceManager
                            .getDefaultSharedPreferences(getApplicationContext()).edit();
                    for (int i = 1; i <= 4; ++i) {
                        appPrefs.putString(PassCodeActivity.PREFERENCE_PASSCODE_D + i,
passcode.substring(i-1, i));
                    }
                    appPrefs.putBoolean(PassCodeActivity.PREFERENCE_SET_PASSCODE, true);
                    appPrefs.commit();
                    Toast.makeText(this, R.string.pass_code_stored,
Toast.LENGTH_LONG).show();
                }
            } else if (requestCode == ACTION_CONFIRM_PASSCODE && resultCode == RESULT_OK) {
                if (data.getBooleanExtra(PassCodeActivity.KEY_CHECK_RESULT, false)) {
                    SharedPreferences.Editor appPrefs = PreferenceManager
                            .getDefaultSharedPreferences(getApplicationContext()).edit();
                    appPrefs.putBoolean(PassCodeActivity.PREFERENCE_SET_PASSCODE, false);
                    appPrefs.commit();
                    Toast.makeText(this, R.string.pass_code_removed,
Toast.LENGTH_LONG).show();
                }
            }
        }
        public ActionBar getSupportActionBar() {
            return getDelegate().getSupportActionBar();
        }
        public void setSupportActionBar(@Nullable Toolbar toolbar) {
            getDelegate().setSupportActionBar(toolbar);
        }
        @Override
        public MenuInflater getMenuInflater() {
            return getDelegate().getMenuInflater();
        }
        @Override
        public void setContentView(@LayoutRes int layoutResID) {
            getDelegate().setContentView(layoutResID);
        }
        @Override
        public void setContentView(View view) {
            getDelegate().setContentView(view);
        }
        @Override
        public void setContentView(View view, ViewGroup.LayoutParams params) {
            getDelegate().setContentView(view, params);
        }
        @Override
        public void addContentView(View view, ViewGroup.LayoutParams params) {
            getDelegate().addContentView(view, params);
        }
        @Override
        protected void onPostResume() {
            super.onPostResume();
            getDelegate().onPostResume();
        }
        @Override
        protected void onTitleChanged(CharSequence title, int color) {
```

```
        super.onTitleChanged(title, color);
        getDelegate().setTitle(title);
    }
    @Override
    public void onConfigurationChanged(Configuration newConfig) {
        super.onConfigurationChanged(newConfig);
        getDelegate().onConfigurationChanged(newConfig);
    }
    @Override
    protected void onPostCreate(Bundle savedInstanceState) {
        super.onPostCreate(savedInstanceState);
        getDelegate().onPostCreate(savedInstanceState);
    }
    @Override
    protected void onDestroy() {
        if (mDownloadServiceConnection != null) {
            unbindService(mDownloadServiceConnection);
            mDownloadServiceConnection = null;
        }
        if (mUploadServiceConnection != null) {
            unbindService(mUploadServiceConnection);
            mUploadServiceConnection = null;
        }
        super.onDestroy();
        getDelegate().onDestroy();
    }
    @Override
    protected void onStop() {
        super.onStop();
        getDelegate().onStop();
    }
    public void invalidateOptionsMenu() {
        getDelegate().invalidateOptionsMenu();
    }
    private AppCompatDelegate getDelegate() {
        if (mDelegate == null) {
            mDelegate = AppCompatDelegate.create(this, null);
        }
        return mDelegate;
    }
    /**
     * Create the list of accounts that has been added into the app
     */
    @SuppressWarnings("deprecation")
    private void addAccountsCheckboxPreferences() {
        // Remove accounts in case list is refreshing for avoiding to have
        // duplicate items
        if (mAccountsPrefCategory.getPreferenceCount() > 0) {
            mAccountsPrefCategory.removeAll();
        }
        AccountManager am = (AccountManager) getSystemService(ACCOUNT_SERVICE);
        Account accounts[] = am.getAccountsByType(MainApp.getAccountType());
        Account currentAccount =
AccountUtils.getCurrentOwnCloudAccount(getApplicationContext());
        if (am.getAccountsByType(MainApp.getAccountType()).length == 0) {
            // Show create account screen if there isn't any account
            am.addAccount(MainApp.getAccountType(), null, null, null, this,
                    null,
                    null);
        }
        else {
```

```
            OwnCloudAccount oca;
            for (Account a : accounts) {
                RadioButtonPreference accountPreference = new
RadioButtonPreference(this);
                accountPreference.setKey(a.name);
                try {
                    oca = new OwnCloudAccount(a, this);
                    accountPreference.setTitle(
                        oca.getDisplayName() + " @ " +

DisplayUtils.convertIdn(a.name.substring(a.name.lastIndexOf("@") + 1), false)
                    );
                } catch (Exception e) {
                    Log_OC.w(
                        TAG,
                        "Account not found right after being read :\\ ; using account
name instead of display name"
                    );
                    // Handle internationalized domain names
                    accountPreference.setTitle(DisplayUtils.convertIdn(a.name,
false));
                }
                mAccountsPrefCategory.addPreference(accountPreference);
                // Check the current account that is being used
                if (a.name.equals(currentAccount.name)) {
                    accountPreference.setChecked(true);
                } else {
                    accountPreference.setChecked(false);
                }
                accountPreference.setOnPreferenceChangeListener(new
OnPreferenceChangeListener() {
                    @Override
                    public boolean onPreferenceChange(Preference preference, Object
newValue) {
                        String key = preference.getKey();
                        AccountManager am = (AccountManager)
getSystemService(ACCOUNT_SERVICE);
                        Account accounts[] =
am.getAccountsByType(MainApp.getAccountType());
                        for (Account a : accounts) {
                            RadioButtonPreference p =
                                    (RadioButtonPreference) findPreference(a.name);
                            if (key.equals(a.name)) {
                                boolean accountChanged = !p.isChecked();
                                p.setChecked(true);
                                AccountUtils.setCurrentOwnCloudAccount(
                                        getApplicationContext(),
                                        a.name
                                );
                                if (accountChanged) {
                                    // restart the main activity
                                    Intent i = new Intent(
                                            Preferences.this,
                                            FileDisplayActivity.class
                                    );
                                    i.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
                                    i.addFlags(Intent.FLAG_ACTIVITY_SINGLE_TOP);
                                    startActivity(i);
                                } else {
                                    finish();
                                }
                            } else {
```

```
                                p.setChecked(false);
                            }
                        }
                        return (Boolean) newValue;
                    }
                });
            }
            // Add Create Account preference at the end of account list if
            // Multiaccount is enabled
            if (getResources().getBoolean(R.bool.multiaccount_support)) {
                createAddAccountPreference();
            }
        }
    }
    /**
     * Create the preference for allow adding new accounts
     */
    private void createAddAccountPreference() {
        Preference addAccountPref = new Preference(this);
        addAccountPref.setKey("add_account");
        addAccountPref.setTitle(getString(R.string.prefs_add_account));
        mAccountsPrefCategory.addPreference(addAccountPref);
        addAccountPref.setOnPreferenceClickListener(new OnPreferenceClickListener() {
            @Override
            public boolean onPreferenceClick(Preference preference) {
                AccountManager am = AccountManager.get(getApplicationContext());
                am.addAccount(MainApp.getAccountType(), null, null, null,
Preferences.this,
                        null, null);
                return true;
            }
        });
    }
    /**
     * Load upload path set on preferences
     */
    private void loadInstantUploadPath() {
        SharedPreferences appPrefs =
                PreferenceManager.getDefaultSharedPreferences(getApplicationContext());
        mUploadPath = appPrefs.getString("instant_upload_path",
getString(R.string.instant_upload_path));
        mPrefInstantUploadPath.setSummary(mUploadPath);
    }
    /**
     * Save the "Instant Upload Path" on preferences
     */
    private void saveInstantUploadPathOnPreferences() {
        SharedPreferences appPrefs =
                PreferenceManager.getDefaultSharedPreferences(getApplicationContext());
        SharedPreferences.Editor editor = appPrefs.edit();
        editor.putString("instant_upload_path", mUploadPath);
        editor.commit();
    }
    /**
     * Load upload video path set on preferences
     */
    private void loadInstantUploadVideoPath() {
        SharedPreferences appPrefs =
                PreferenceManager.getDefaultSharedPreferences(getApplicationContext());
        mUploadVideoPath = appPrefs.getString("instant_video_upload_path",
getString(R.string.instant_upload_path));
```

```
            mPrefInstantVideoUploadPath.setSummary(mUploadVideoPath);
        }
        /**
         * Save the "Instant Video Upload Path" on preferences
         */
        private void saveInstantUploadVideoPathOnPreferences() {
            SharedPreferences appPrefs =
                    PreferenceManager.getDefaultSharedPreferences(getApplicationContext());
            SharedPreferences.Editor editor = appPrefs.edit();
            editor.putString("instant_video_upload_path", mUploadVideoPath);
            editor.commit();
        }
        // Methods for ComponetsGetter
        @Override
        public FileDownloader.FileDownloaderBinder getFileDownloaderBinder() {
            return mDownloaderBinder;
        }
        @Override
        public FileUploader.FileUploaderBinder getFileUploaderBinder() {
            return mUploaderBinder;
        }
        @Override
        public OperationsService.OperationsServiceBinder getOperationsServiceBinder() {
            return null;
        }
        @Override
        public FileDataStorageManager getStorageManager() {
            return null;
        }
        @Override
        public FileOperationsHelper getFileOperationsHelper() {
            return null;
        }
        protected ServiceConnection newTransferenceServiceConnection() {
            return new PreferencesServiceConnection();
        }
        /** Defines callbacks for service binding, passed to bindService() */
        private class PreferencesServiceConnection implements ServiceConnection {
            @Override
            public void onServiceConnected(ComponentName component, IBinder service) {
                if (component.equals(new ComponentName(Preferences.this,
FileDownloader.class))) {
                    mDownloaderBinder = (FileDownloader.FileDownloaderBinder) service;
                } else if (component.equals(new ComponentName(Preferences.this,
                        FileUploader.class))) {
                    Log_OC.d(TAG, "Upload service connected");
                    mUploaderBinder = (FileUploader.FileUploaderBinder) service;
                } else {
                    return;
                }
            }
            @Override
            public void onServiceDisconnected(ComponentName component) {
                if (component.equals(new ComponentName(Preferences.this,
FileDownloader.class))) {
                    Log_OC.d(TAG, "Download service suddenly disconnected");
                    mDownloaderBinder = null;
                } else if (component.equals(new ComponentName(Preferences.this,
                        FileUploader.class))) {
                    Log_OC.d(TAG, "Upload service suddenly disconnected");
                    mUploaderBinder = null;
```

```
            }
        }
    };
}
```

11.8 android/res/xml/preferences.xml

```
<?xml version="1.0" encoding="utf-8"?>
<!--
  ownCloud Android client application
  Copyright (C) 2012  Bartek Przybylski
  Copyright (C) 2015 ownCloud Inc.
  This program is free software: you can redistribute it and/or modify
  it under the terms of the GNU General Public License version 2,
  as published by the Free Software Foundation.
  This program is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
  GNU General Public License for more details.
  You should have received a copy of the GNU General Public License
  along with this program.  If not, see <http://www.gnu.org/licenses/>.
-->
<PreferenceScreen xmlns:android="http://schemas.android.com/apk/res/android" >
    <PreferenceCategory android:title="@string/prefs_category_accounts"
android:key="accounts_category">
    </PreferenceCategory>
   <PreferenceCategory android:title="@string/prefs_category_mqtt"
android:key="mqtt_category">
      <com.owncloud.android.utils.IPAddressPreference
          android:key="broker"
          android:title="@string/broker_address"
          android:summary="@string/broker_address_summary"
          android:defaultValue="192.168.1.6"
          android:inputType="text"/>
      <EditTextPreference android:key="topic"
          android:title="@string/topic_name"
          android:summary="@string/broker_address_summary"
          android:defaultValue="SYS/disk"
          android:inputType="text" />
   </PreferenceCategory>
    <PreferenceCategory android:title="@string/prefs_category_instant_uploading"
android:key="instant_uploading_category">
      <com.owncloud.android.ui.CheckBoxPreferenceWithLongTitle
android:key="instant_uploading"
                              android:title="@string/prefs_instant_upload"
                              android:summary="@string/prefs_instant_upload_summary"/>
          <com.owncloud.android.ui.PreferenceWithLongSummary
                        android:title="@string/prefs_instant_upload_path_title"
                        android:key="instant_upload_path" />
       <com.owncloud.android.ui.CheckBoxPreferenceWithLongTitle
                          android:title="@string/instant_upload_on_wifi"
                          android:key="instant_upload_on_wifi"/>
       <com.owncloud.android.ui.CheckBoxPreferenceWithLongTitle
android:key="instant_video_uploading"
                              android:title="@string/prefs_instant_video_upload"

android:summary="@string/prefs_instant_video_upload_summary" />
       <com.owncloud.android.ui.PreferenceWithLongSummary
                        android:title="@string/prefs_instant_video_upload_path_title"
```

```
                        android:key="instant_video_upload_path" />
        <com.owncloud.android.ui.CheckBoxPreferenceWithLongTitle
                        android:title="@string/instant_video_upload_on_wifi"
                        android:key="instant_video_upload_on_wifi"/>
       <com.owncloud.android.ui.dialog.OwnCloudListPreference
android:key="prefs_instant_behaviour"
          android:dialogTitle="@string/prefs_instant_behaviour_dialogTitle"
          android:title="@string/prefs_instant_behaviour_title"
          android:entries="@array/pref_behaviour_entries"
          android:entryValues="@array/pref_behaviour_entryValues"
          android:defaultValue="NOTHING"
          android:summary="%s"
          />
        <!-- DISABLED FOR RELEASE UNTIL FIXED
        CheckBoxPreference android:key="log_to_file"
                          android:title="@string/prefs_log_title"
                          android:summary="@string/prefs_log_summary"/>
       <Preference       android:key="log_history"
                          android:title="@string/prefs_log_title_history"
                          android:summary="@string/prefs_log_summary_history"/ -->

     </PreferenceCategory>
    <PreferenceCategory android:title="@string/prefs_category_details">
       <android.preference.CheckBoxPreference android:title="@string/prefs_passcode"
android:key="set_pincode" />
    </PreferenceCategory>

    <PreferenceCategory android:title="@string/prefs_category_more" android:key="more">
       <Preference android:title="@string/prefs_help" android:key="help" />
       <Preference android:title="@string/prefs_recommend" android:key="recommend" />
       <Preference android:title="@string/prefs_feedback" android:key="feedback" />
       <Preference android:title="@string/actionbar_logger" android:key="logger" />
       <Preference android:title="@string/prefs_imprint" android:key="imprint" />
       <Preference android:title="@string/about_title" android:id="@+id/about_app"
android:key="about_app" />
    </PreferenceCategory>

</PreferenceScreen>
```

12 Παράρτημα Β - Κώδικας Εξυπηρετητή

12.1 Αρχείο παραμετροποίησης του openssl

Το παρακάτω αρχείο πρέπει να βρίσκεται στη τοποθεσία /etc/ssl/openssl.cnf

```
#
# OpenSSL example configuration file.
# This is mostly being used for generation of certificate requests.
#

# This definition stops the following lines choking if HOME isn't
# defined.
HOME            = .
RANDFILE        = $ENV::HOME/.rnd

# Extra OBJECT IDENTIFIER info:
#oid_file       = $ENV::HOME/.oid
oid_section     = new_oids

# To use this configuration file with the "-extfile" option of the
# "openssl x509" utility, name here the section containing the
# X.509v3 extensions to use:
# extensions            =
# (Alternatively, use a configuration file that has only
# X.509v3 extensions in its main [= default] section.)

[ new_oids ]

# We can add new OIDs in here for use by 'ca', 'req' and 'ts'.
# Add a simple OID like this:
# testoid1=1.2.3.4
# Or use config file substitution like this:
# testoid2=${testoid1}.5.6

# Policies used by the TSA examples.
tsa_policy1 = 1.2.3.4.1
tsa_policy2 = 1.2.3.4.5.6
tsa_policy3 = 1.2.3.4.5.7

####################################################################
[ ca ]
default_ca  = CA_default            # The default ca section

####################################################################
[ CA_default ]

dir         = /root/SSLCertAuth     # Where everything is kept
certs       = $dir/certs            # Where the issued certs are kept
crl_dir           = $dir/crl        # Where the issued crl are kept
database    = $dir/index.txt  # database index file.
#unique_subject   = no              # Set to 'no' to allow creation of
                                # several ctificates with same subject.
```

```
new_certs_dir     = $dir/newcerts         # default place for new certs.

certificate = $dir/cacert.pem       # The CA certificate
serial            = $dir/serial           # The current serial number
crlnumber   = $dir/crlnumber  # the current crl number
                              # must be commented out to leave a V1 CRL
crl         = $dir/crl.pem          # The current CRL
private_key = $dir/private/cakey.pem# The private key
RANDFILE    = $dir/private/.rand    # private random number file

x509_extensions   = usr_cert        # The extentions to add to the cert

# Comment out the following two lines for the "traditional"
# (and highly broken) format.
name_opt    = ca_default            # Subject Name options
cert_opt    = ca_default            # Certificate field options

# Extension copying option: use with caution.
# copy_extensions = copy

# Extensions to add to a CRL. Note: Netscape communicator chokes on V2 CRLs
# so this is commented out by default to leave a V1 CRL.
# crlnumber must also be commented out to leave a V1 CRL.
# crl_extensions  = crl_ext

default_days      = 3650                  # how long to certify for
default_crl_days= 30                # how long before next CRL
default_md  = default         # use public key default MD
preserve    = no              # keep passed DN ordering

# A few difference way of specifying how similar the request should look
# For type CA, the listed attributes must be the same, and the optional
# and supplied fields are just that :-)
policy            = policy_match

# For the CA policy
[ policy_match ]
countryName       = match
stateOrProvinceName     = match
organizationName  = match
organizationalUnitName  = optional
commonName        = supplied
emailAddress            = optional

# For the 'anything' policy
# At this point in time, you must list all acceptable 'object'
# types.
[ policy_anything ]
countryName       = optional
stateOrProvinceName     = optional
localityName            = optional
organizationName  = optional
organizationalUnitName  = optional
commonName        = supplied
emailAddress            = optional

####################################################################
[ req ]
```

```
default_bits            = 2048
default_keyfile   = privkey.pem
distinguished_name      = req_distinguished_name
attributes        = req_attributes
x509_extensions   = v3_ca     # The extentions to add to the self signed cert

# Passwords for private keys if not present they will be prompted for
# input_password = secret
# output_password = secret

# This sets a mask for permitted string types. There are several options.
# default: PrintableString, T61String, BMPString.
# pkix       : PrintableString, BMPString (PKIX recommendation before 2004)
# utf8only: only UTF8Strings (PKIX recommendation after 2004).
# nombstr : PrintableString, T61String (no BMPStrings or UTF8Strings).
# MASK:XXXX a literal mask value.
# WARNING: ancient versions of Netscape crash on BMPStrings or UTF8Strings.
string_mask = utf8only

# req_extensions = v3_req # The extensions to add to a certificate request

[ req_distinguished_name ]
countryName             = Country Name (2 letter code)
countryName_default             = GR
countryName_min                 = 2
countryName_max                 = 2

stateOrProvinceName             = State or Province Name (full name)
stateOrProvinceName_default     = Some-State

localityName                    = Locality Name (eg, city)

0.organizationName              = Organization Name (eg, company)
0.organizationName_default      = Internet Widgits Pty Ltd

# we can do this but it is not needed normally :-)
#1.organizationName             = Second Organization Name (eg, company)
#1.organizationName_default     = World Wide Web Pty Ltd

organizationalUnitName          = Organizational Unit Name (eg, section)
#organizationalUnitName_default     =

commonName              = Common Name (e.g. server FQDN or YOUR name)
commonName_max                  = 64

emailAddress                    = Email Address
emailAddress_max        = 64

# SET-ex3               = SET extension number 3

[ req_attributes ]
challengePassword       = A challenge password
challengePassword_min           = 4
challengePassword_max           = 20

unstructuredName        = An optional company name

[ usr_cert ]
```

```
# These extensions are added when 'ca' signs a request.

# This goes against PKIX guidelines but some CAs do it and some software
# requires this to avoid interpreting an end user certificate as a CA.

basicConstraints=CA:FALSE

# Here are some examples of the usage of nsCertType. If it is omitted
# the certificate can be used for anything *except* object signing.

# This is OK for an SSL server.
# nsCertType                    = server

# For an object signing certificate this would be used.
# nsCertType = objsign

# For normal client use this is typical
# nsCertType = client, email

# and for everything including object signing:
# nsCertType = client, email, objsign

# This is typical in keyUsage for a client certificate.
# keyUsage = nonRepudiation, digitalSignature, keyEncipherment

# This will be displayed in Netscape's comment listbox.
nsComment                = "OpenSSL Generated Certificate"

# PKIX recommendations harmless if included in all certificates.
subjectKeyIdentifier=hash
authorityKeyIdentifier=keyid,issuer

# This stuff is for subjectAltName and issuerAltname.
# Import the email address.
# subjectAltName=email:copy
# An alternative to produce certificates that aren't
# deprecated according to PKIX.
# subjectAltName=email:move

# Copy subject details
# issuerAltName=issuer:copy

#nsCaRevocationUrl             = http://www.domain.dom/ca-crl.pem
#nsBaseUrl
#nsRevocationUrl
#nsRenewalUrl
#nsCaPolicyUrl
#nsSslServerName

# This is required for TSA certificates.
# extendedKeyUsage = critical,timeStamping

[ v3_req ]

# Extensions to add to a certificate request

basicConstraints = CA:FALSE
```

```
keyUsage = nonRepudiation, digitalSignature, keyEncipherment

[ v3_ca ]


# Extensions for a typical CA


# PKIX recommendation.

subjectKeyIdentifier=hash

authorityKeyIdentifier=keyid:always,issuer

# This is what PKIX recommends but some broken software chokes on critical
# extensions.
#basicConstraints = critical,CA:true
# So we do this instead.
basicConstraints = CA:true

# Key usage: this is typical for a CA certificate. However since it will
# prevent it being used as an test self-signed certificate it is best
# left out by default.
# keyUsage = cRLSign, keyCertSign

# Some might want this also
# nsCertType = sslCA, emailCA

# Include email address in subject alt name: another PKIX recommendation
# subjectAltName=email:copy
# Copy issuer details
# issuerAltName=issuer:copy

# DER hex encoding of an extension: beware experts only!
# obj=DER:02:03
# Where 'obj' is a standard or added object
# You can even override a supported extension:
# basicConstraints= critical, DER:30:03:01:01:FF

[ crl_ext ]

# CRL extensions.
# Only issuerAltName and authorityKeyIdentifier make any sense in a CRL.

# issuerAltName=issuer:copy
authorityKeyIdentifier=keyid:always

[ proxy_cert_ext ]
# These extensions should be added when creating a proxy certificate

# This goes against PKIX guidelines but some CAs do it and some software
# requires this to avoid interpreting an end user certificate as a CA.

basicConstraints=CA:FALSE

# Here are some examples of the usage of nsCertType. If it is omitted
# the certificate can be used for anything *except* object signing.
```

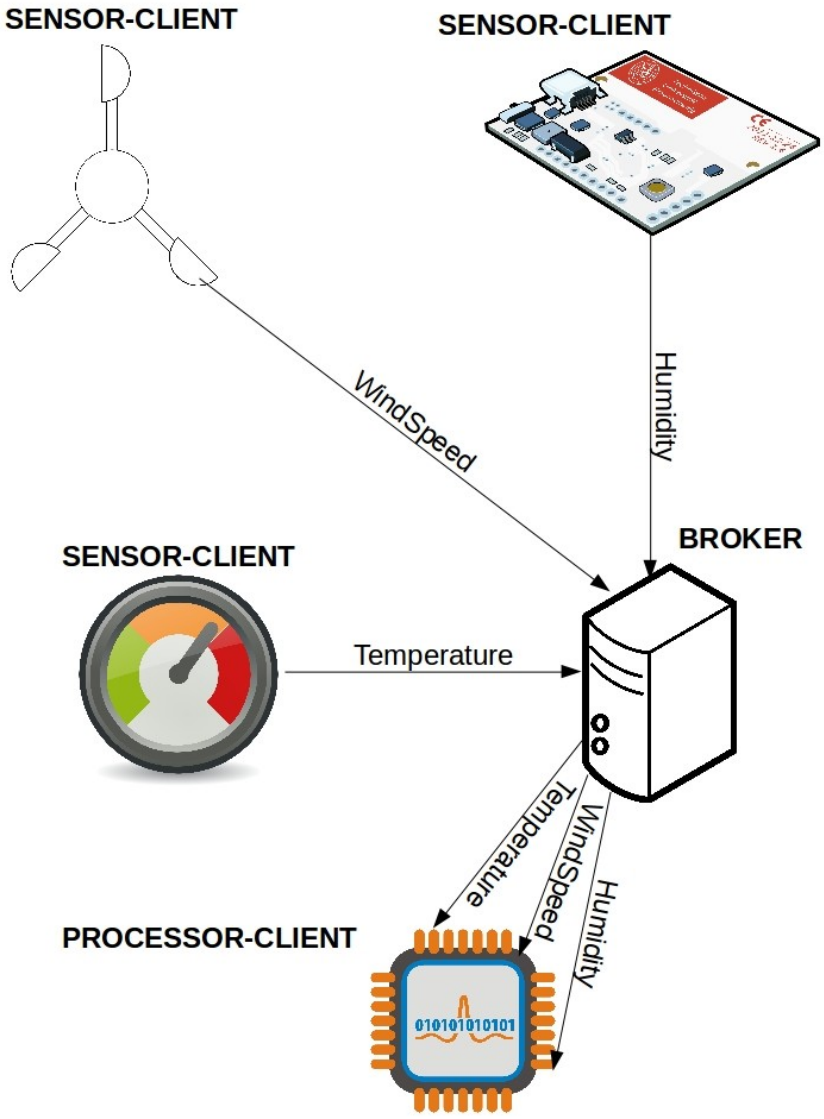
```
# This is OK for an SSL server.
# nsCertType                    = server

# For an object signing certificate this would be used.
# nsCertType = objsign

# For normal client use this is typical
# nsCertType = client, email

# and for everything including object signing:
# nsCertType = client, email, objsign

# This is typical in keyUsage for a client certificate.
# keyUsage = nonRepudiation, digitalSignature, keyEncipherment

# This will be displayed in Netscape's comment listbox.
nsComment               = "OpenSSL Generated Certificate"

# PKIX recommendations harmless if included in all certificates.
subjectKeyIdentifier=hash
authorityKeyIdentifier=keyid,issuer

# This stuff is for subjectAltName and issuerAltname.
# Import the email address.
# subjectAltName=email:copy
# An alternative to produce certificates that aren't
# deprecated according to PKIX.
# subjectAltName=email:move

# Copy subject details
# issuerAltName=issuer:copy

#nsCaRevocationUrl              = http://www.domain.dom/ca-crl.pem
#nsBaseUrl
#nsRevocationUrl
#nsRenewalUrl
#nsCaPolicyUrl
#nsSslServerName

# This really needs to be in place for it to be a proxy certificate.
proxyCertInfo=critical,language:id-ppl-anyLanguage,pathlen:3,policy:foo

####################################################################
[ tsa ]

default_tsa = tsa_config1     # the default TSA section

[ tsa_config1 ]

# These are used by the TSA reply generation only.
dir         = ./demoCA        # TSA root directory
serial            = $dir/tsaserial  # The current serial number (mandatory)
crypto_device     = builtin         # OpenSSL engine to use for signing
signer_cert = $dir/tsacert.pem      # The TSA signing certificate
                            # (optional)
certs       = $dir/cacert.pem # Certificate chain to include in reply
                            # (optional)
signer_key  = $dir/private/tsakey.pem # The TSA private key (optional)
```

```
default_policy    = tsa_policy1           # Policy if request did not specify it
                                  # (optional)
other_policies    = tsa_policy2, tsa_policy3    # acceptable policies (optional)
digests           = md5, sha1       # Acceptable message digests (mandatory)
accuracy    = secs:1, millisecs:500, microsecs:100    # (optional)
clock_precision_digits  = 0   # number of digits after dot. (optional)
ordering          = yes # Is ordering defined for timestamps?
                        # (optional, default: no)
tsa_name          = yes # Must the TSA name be included in the reply?
                        # (optional, default: no)
ess_cert_id_chain = no  # Must the ESS cert id chain be included?
                        # (optional, default: no)
```

13 Παράρτημα Γ - Το πρωτόκολλο MQTT

Το πρωτόκολλο MQTT είναι ένα ελαφρύ πρωτόκολλο βασιζόμενο στο TCP/IP που δημιουργήθηκε το 1999 από την IBM. Είναι σχεδιασμένο για χρήση από απομακρυσμένες τοποθεσίες, όπου απαιτείται η εκτέλεση ελάχιστου κώδικα, ή για περιπτώσεις που δεν υπάρχει μεγάλο εύρος ζώνης. Βασίζεται στην αρχιτεκτονική publish και subscribe, στην οποία αρκετοί πελάτες (Clients) επικοινωνούν με ένα διαμεσολαβητή (Broker). Ο διαμεσολαβητής, είναι υπεύθυνος για το διαμοιρασμό μηνυμάτων στους πελάτες, ανάλογα με το θέμα του κάθε μηνύματος.

Παρακάτω μπορούμε να δούμε ένα παράδειγμα μιας διάταξης με τρεις αισθητήρες, ο καθένας εκ των οποίων επικοινωνεί με τον Broker για ένα συγκεκριμένο θέμα, και με ένα σύστημα επεξεργασίας πληροφοριών που αντλεί δεδομένα από όλα τα θέματα.



Για την επίτευξη της επικοινωνίας μεταξύ MQTT client και Broker, το πρωτόκολλο χρησιμοποιεί τις παρακάτω μεθόδους:

Connect
Περιμένει να δημιουργηθεί μια σύνδεση με τον εξυπηρετητή.

Disconnect
Περιμένει να τελειώσει ο πελάτης MQTT οποιαδήποτε δραστηριότητά του, και να κλείσει η σύνοδος TCP/IP.

Subscribe
Δημιουργεί μία συνδρομή για ένα ή περισσότερα θέματα από τον πελάτη προς τον εξυπηρετητή.

UnSubscribe
Περιμένει να δημιουργηθεί μια σύνδεση με τον εξυπηρετητή.

Publish
Επιστρέφει αμέσως στο νήμα εκτέλεσης της εφαρμογής, αφού ζητήσει από τον πελάτη MQTT να δημοσιεύσει κάποιο μήνυμα.

Ένας client μπορεί να παρακολουθεί ένα ή περισσότερα θέματα (topics) τα οποία φιλοξενούνται στον broker. Η επικοινωνία ενός client με τον broker, και η αποστολή ενός μηνύματος που αλλάζει την τιμή ενός θέματος, έχει ως αποτέλεσμα να ανανεώνονται τα δεδομένα για όλους τους clients που έχουν συνδρομή (subscription) στο συγκεκριμένο θέμα του broker.

Ποιότητες υπηρεσίας

Το πρωτόκολλο MQTT, παρέχει διάφορα επίπεδα ποιότητας υπηρεσίας, 0,1 και 2, που μπορούμε να χρησιμοποιήσουμε ανάλογα με την εφαρμογή μας για την επικοινωνία του πελάτη με τον διαμεσολαβητή.

Η πιο απλή μέθοδος επικοινωνίας, που αντιστοιχεί στη ποιότητα υπηρεσίας 0, είναι η απλή αποστολή ενός μηνύματος, χωρίς επιβεβαίωση για την ορθή λήψη του μηνύματος. Όμως, το πρωτόκολλο MQTT παρέχει επιπλέον δυνατότητες για την εξασφάλιση της αποστολής των μηνυμάτων.

Μία πιο σύνθετη μέθοδος επικοινωνίας αντιστοιχεί στη ποιότητα υπηρεσίας 1. Τα μηνύματα που στέλνονται με αυτή τη ποιότητα υπηρεσίας, είναι βέβαιο ότι θα μεταφερθούν τουλάχιστον μία φορά από τον πελάτη προς τον εξυπηρετητή, όμως μπορεί να μεταφερθούν και παραπάνω φορές. Η χρήση αυτής της μεθόδου, επιφέρει μια επιπρόσθετη καθυστέρηση στην επικοινωνία.

Τέλος, η πιο σύνθετη και περισσότερο εγγυημένη μέθοδος επικοινωνίας, αντιστοιχεί στη ποιότητα υπηρεσίας 2. Εδώ εγγυόμαστε ότι το κάθε μήνυμα θα αποσταλεί ακριβώς μία φορά. Αυτή η εγγύηση είναι σημαντική σε περιπτώσεις που η αποστολή του μηνύματος παραπάνω από μία φορές θα είχε αρνητικά αποτελέσματα στην παρεχόμενη υπηρεσία (π.χ. έναυση συναγερμού, εκκίνηση κινητήρα κλπ.)

Δυνατότητες κρυπτογράφησης

Το πρωτόκολλο παρέχει τη δυνατότητα κρυπτογράφησης των μηνυμάτων που ανταλλάσσονται μεταξύ του πελάτη και του εξυπηρετητή, ώστε να μην είναι δυνατή η ανάγνωση των περιεχομένων τους. Ωστόσο, συνιστάται η χρήση ειδικών αλγορίθμων στη περίπτωση που χρησιμοποιούν το πρωτόκολλο συσκευές χαμηλών επιδόσεων.

Συμπεράσματα

Για τις παραπάνω ιδιότητές του, το πρωτόκολλο MQTT χρησιμοποιείται κατά κόρον σε δίκτυα αυτοματισμών και αισθητήρων. Η αξιοπιστία του και η ευελιξία του είναι οι λογοι για τους οποίους χρησιμοποιείται στην εργασία μας, ως μέρος της αρχιτεκτονικής για τον έλεγχο της πρόσβασης στα δεδομένα από μια συσκευή Android.

Παραπομπές Παραρτήματος Γ'

http://www.hivemq.com/blog/mqtt-essentials-part-6-mqtt-quality-of-service-levels
http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html#_Toc398718063
https://en.wikipedia.org/wiki/MQTT