Εθνικο Μετσοβιο Πολυτεχνειο
Σχολη Ηλεκτρολογων Μηχανικων και Μηχανικων Υπολογιστων
Τομέας Τεχνολογίας Πληροφορικής και Υπολογιστών

Πλατφόρμα Διαχείρησης Ετερογενών Δεδομένων σε Πολλαπλά Αποθετήρια με Υποστήριξη Ιστορικών και Ροών Πραγματικού Χρόνου

Διπλωματικη Εργασια

της

ΣΠΕΝΤΖΟΥ ΔΑΝΑΗΣ

Επιβλέπων: Παναγιώτης Τσανάκας
Καθηγητής Ε.Μ.Π.

Αθήνα, Οκτώβριος 2025

Εθνικο Μετσοβιο Πολυτεχνειο
Σχολη Ηλεκτρολογων Μηχανικων και Μηχανικων Υπολογιστων
Τομέας Τεχνολογίας Πληροφορικής και Υπολογιστών

Πλατφόρμα Διαχείρησης Ετερογενών Δεδομένων σε Πολλαπλά Αποθετήρια με Υποστήριξη Ιστορικών και Ροών Πραγματικού Χρόνου

Διπλωματικη Εργασια

της

ΣΠΕΝΤΖΟΥ ΔΑΝΑΗΣ

Επιβλέπων: Παναγιώτης Τσανάκας
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 28α Οκτωβρίου 2025.

(Υπογραφή)	*(Υπογραφή)*	*(Υπογραφή)*
.....	.....	.....
Παναγιώτης Τσανάκας	Ανδρέας-Γεώργιος Σταφυλοπάτης	Βασίλειος Βεσκούκης
Καθηγητής Ε.Μ.Π.	Ομότιμος Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.

Αθήνα, Οκτώβριος 2025

Εθνικο Μετσοβιο Πολυτεχνειο
Σχολη Ηλεκτρολογων Μηχανικων και Μηχανικων Υπολογιστων
Τομέας Τεχνολογίας Πληροφορικής και Υπολογιστών

ΔΗΛΩΣΗ ΜΗ ΛΟΓΟΚΛΟΠΗΣ ΚΑΙ ΑΝΑΛΗΨΗΣ ΠΡΟΣΩΠΙΚΗΣ ΕΥΘΥΝΗΣ

Με πλήρη επίγνωση των συνεπειών του νόμου περί πνευματικών δικαιωμάτων, δηλώνω ενυπογράφως ότι είμαι αποκλειστικός συγγραφέας της παρούσας Πτυχιακής Εργασίας, για την ολοκλήρωση της οποίας κάθε βοήθεια είναι πλήρως αναγνωρισμένη και αναφέρεται λεπτομερώς στην εργασία αυτή. Έχω αναφέρει πλήρως και με σαφείς αναφορές, όλες τις πηγές χρήσης δεδομένων, απόψεων, θέσεων και προτάσεων, ιδεών και λεκτικών αναφορών, είτε κατά κυριολεξία είτε βάσει επιστημονικής παράφρασης. Αναλαμβάνω την προσωπική και ατομική ευθύνη ότι σε περίπτωση αποτυχίας στην υλοποίηση των ανωτέρω δηλωθέντων στοιχείων, είμαι υπόλογος έναντι λογοκλοπής, γεγονός που σημαίνει αποτυχία στην Πτυχιακή μου Εργασία και κατά συνέπεια αποτυχία απόκτησης του Τίτλου Σπουδών, πέραν των λοιπών συνεπειών του νόμου περί πνευματικών δικαιωμάτων. Δηλώνω, συνεπώς, ότι αυτή η Πτυχιακή Εργασία προετοιμάστηκε και ολοκληρώθηκε από εμένα προσωπικά και αποκλειστικά και ότι, αναλαμβάνω πλήρως όλες τις συνέπειες του νόμου στην περίπτωση κατά την οποία αποδειχθεί, διαχρονικά, ότι η εργασία αυτή ή τμήμα της δεν μου ανήκει διότι είναι προϊόν λογοκλοπής άλλης πνευματικής ιδιοκτησίας.

(Υπογραφή)

.....
Δανάη Σπέντζου

Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών ΕΜΠ
28 Οκτωβρίου 2025

Περίληψη

Οι εφαρμογές IoT χρησιμοποιούνται όλο και περισσότερο σε διάφορους τομείς, από τις έξυπνες πόλεις έως τη βιομηχανία, παρέχοντας συνεχή ροή δεδομένων από αισθητήρες, κάμερες και drones. Στον τομέα των ορυχείων, η συνεχής παρακολούθηση και ο έλεγχος των διαδικασιών αποτελούν καθοριστικούς παράγοντες για την ασφάλεια, την επιχειρησιακή αποτελεσματικότητα και την έγκαιρη λήψη αποφάσεων. Ωστόσο, τα δεδομένα που συλλέγονται είναι ετερογενή — περιλαμβάνοντας αριθμητικές μετρήσεις, εικόνες και βίντεο —, προέρχονται από πολλαπλές πηγές και αποθηκεύονται σε διαφορετικά αποθετήρια και μορφές, γεγονός που καθιστά δύσκολη τη συλλογή, την ενοποίηση και την αξιοποίησή τους.

Για την αντιμετώπιση αυτής της πολυπλοκότητας, αναπτύχθηκε μια ενιαία πλατφόρμα διαχείρισης ετερογενών δεδομένων, η οποία λειτουργεί ως κεντρικό σημείο πρόσβασης και ενοποίησης πολλαπλών πηγών. Η πλατφόρμα υποστηρίζει αποθήκευση, μεταφορά και ανάκτηση δεδομένων τόσο σε πραγματικό χρόνο όσο και από ιστορικά αρχεία, ενοποιώντας διαφορετικούς τύπους πληροφορίας κάτω από μια κοινή αρχιτεκτονική.

Με αυτήν την προσέγγιση, οι πληροφορίες αξιοποιούνται άμεσα για παρακολούθηση, πρόβλεψη προβλημάτων και λήψη κρίσιμων αποφάσεων σε πραγματικό χρόνο. Ειδικά στη μεταλλευτική βιομηχανία, η λύση αυτή ενισχύει την ασφάλεια, τη λειτουργική αποδοτικότητα και την ανθεκτικότητα των υποδομών.

Λέξεις Κλειδιά

Big Data, Batch Processing, Real-time Streaming, API

Abstract

IoT applications are increasingly being used in critical domains such as smart cities and industry, providing a continuous stream of data from sensors, cameras, and drones. In the mining sector, continuous monitoring and process control are key factors for ensuring safety, operational efficiency, and timely decision-making.

However, the collected data is highly heterogeneous — including numerical measurements, images, and videos — originating from multiple sources and stored in different repositories and formats. This fragmentation makes the collection, integration, and effective use of data particularly challenging.

To address this complexity, a unified data management platform has been developed, serving as a central access point for integrating multiple data sources. The platform supports the storage, transfer, and retrieval of both real-time and historical data, consolidating various data types under a common architecture.

This approach enables information to be instantly utilized for monitoring, problem prediction, and critical decision-making in real time. In the mining industry in particular, this solution enhances safety, operational efficiency, and infrastructure resilience.

Keywords

Big Data, Batch Processing, Real-time Streaming, API

στην οικογένεια μου

Ευχαριστίες

Ξεκινώντας, Θα ήθελα να ευχαριστήσω θερμά τον επιβλέποντα καθηγητή κ. Κόνιαρη για την πολύτιμη καθοδήγηση, την αμέριστη υποστήριξη και την εξαιρετική συνεργασία καθ' όλη τη διάρκεια της διπλωματικής μου εργασίας. Οι γνώσεις του, η επιμονή του και κυρίως το ενδιαφέρον του για εμένα συνέβαλαν καθοριστικά στη βελτίωση των ιδεών μου και στην επιτυχή ολοκλήρωση της εργασίας αυτής.

Επιπλέον, θα ήθελα να εκφράσω την ιδιαίτερη ευγνωμοσύνη μου προς τον κ. Κούζα, ο οποίος με εμπιστεύτηκε να φέρω εις πέρας αυτό το απαιτητικό έργο. Οι εβδομαδιαίες μας συναντήσεις και η έμφαση που έδινε στη λεπτομέρεια υπήρξαν πολύτιμες.

Τέλος, θα ήθελα να πω ένα μεγάλο ευχαριστώ στην οικογένειά μου και στους φίλους μου για την αμέριστη στήριξη και ενθάρρυνσή τους, αλλά πάνω από όλα στη γιαγιά μου Άννα για την αγάπη και την ηθική συμπαράσταση που μου προσέφερε όλα αυτά τα χρόνια. Χωρίς την πίεση και την επιμονή της, ίσως να μην είχα φτάσει τόσο γρήγορα στην ολοκλήρωση των σπουδών μου.

Αθήνα, Οκτώβριος 2025

Δανάη Σπέντζου

Περιεχόμενα

Περίληψη **5**

Abstract **7**

Ευχαριστίες **11**

1 Εισαγωγή **18**
1.1 Περιγραφή Προβλήματος . . . 18
1.2 Αντικείμενο Διπλωματικής . . . 19
1.2.1 Συνεισφορά . . . 19
1.3 Οργάνωση Κειμένου . . . 19

2 Θεωρητικό υπόβαθρο και Σχετικές Εργασίες **21**
2.1 Θεωρητικό Υπόβαθρο . . . 21
2.1.1 Internet of Things (IoT) . . . 21
2.1.2 Μεγάλα Δεδομένα (Big Data) . . . 21
2.1.3 Θεώρημα (CAP (Consistency, Availability, Partition Tolerance)) . . . 24
2.1.4 Service Oriented Architecture (SOA) . . . 24
2.1.5 Software As A Service (SaaS) . . . 25
2.1.6 Architectural Models . . . 26
2.1.7 REST API . . . 28
2.1.8 Documenting REST APIs . . . 30
2.1.9 Enterprise Integration Patterns . . . 30
2.2 Σχετικές Εργασίες και Παραδείγματα . . . 31
2.2.1 Παρουσίαση Τεχνολογιών . . . 32
2.2.2 Ετερογενή Δεδομένα από Πολλαπλά Αποθετήρια . . . 33
2.2.3 Διαχείριση Ιστορικών Δεδομένων (Batch Processing) . . . 33
2.2.4 Ροές Δεδομένων και Επεξεργασία σε Πραγματικό Χρόνο (Stream Processing) . . . 34
2.2.5 Σύνοψη Τεχνολογιών . . . 34
2.2.6 Παρατηρήσεις . . . 35

3 Ανάλυση και Σχεδίαση **38**
3.1 Ανάλυση απαιτήσεων . . . 38
3.2 Σενάρια Χρήσης . . . 39
3.3 Περιγραφή αρχιτεκτονικής . . . 41
3.4 Διαχωρισμός υποσυστημάτων . . . 43

3.5 Περιγραφή υποσυστημάτων 44
3.6 Μοντέλο Δεδομένων &Μεταδεδομένων 48
3.6.1 Μοντέλο Οντοτήτων Συσχετίσεων 48
3.6.2 Συσχετίσεις μεταξύ των Οντοτήτων 51

4 Υλοποίηση **53**
4.1 Λεπτομέρειες Υλοποίησης 53
4.2 Υλοποίηση Connectors 53
4.3 Υλοποίηση Βασικών Λειτουργιών της πλατφόρμας: Data Upload, Data Retrieve, Flows 55
4.4 Υλοποίηση Frontend (UI) 59
4.5 Deployment - Deployment diagram 65

5 Αξιολόγηση Συστήματος **67**
5.1 Μεθοδολογία Αξιολόγησης 67
5.2 Αναλυτική παρουσίαση ελέγχου 68
5.2.1 Προεργασία για την Εκτέλεση των Σεναρίων 68
5.2.2 Σενάριο Α: Ροή δεδομένων και αρχείων από Kafka προς αποθετήριο τύπου MongoDB 72
5.2.3 Σενάριο Β: Ανάκτηση αρχείων από αποθετήριο τύπου MongoDB 76
5.2.4 Σενάριο Γ: Προσθήκη ιστορικών δεδομένων σε αποθετήριο τύπου MongoDB 78
5.2.5 Σύνοψη Αξιολόγησης 80
5.2.6 Σχόλια και Βελτιώσεις 80

6 Επίλογος **82**
6.1 Σύνοψη και Γενικά Συμπεράσματα 82
6.2 Μελλοντικές Επεκτάσεις 83

Βιβλιογραφία **88**

Κατάλογος Σχημάτων

2.1 Τα επτά βασικά χαρακτηριστικά των Μεγάλων Δεδομένων (7V's of Big Data). Το σχήμα παρουσιάζει τις κύριες ιδιότητες που βοηθούν στην κατανόηση και τη διαχείριση μεγάλου όγκου δεδομένων. 22
2.2 Το Waterfall Model παρουσιάζει διαδοχικά στάδια ανάπτυξης. Έχει γραμμική ροή και έλλειψη επιστροφής σε προηγούμενα βήματα. 26
2.3 Το Iterative Model βασίζεται σε επαναλήψεις. Υπάρχει κυκλική διαδικασία ανάπτυξης και βελτίωσης. 26
2.4 Το V-Model συνδέει κάθε φάση σχεδίασης με δοκιμές. Έχει συμμετρία σχεδίασης–ελέγχου. 27
2.5 Το Agile Model βασίζεται σε σύντομα sprints. Υπάρχει ευελιξία και συνεχή ανατροφοδότηση. 27
2.6 Το Prototype Model χρησιμοποιεί πρώιμο πρωτότυπο. Υπάρχει ροή μεταξύ σχεδίασης, αξιολόγησης και βελτίωσης. 28
2.7 Το Spiral Model εστιάζει στη σταδιακή ανάπτυξη με έλεγχο ρίσκου. Οι σπείρες αντιστοιχούν στους κύκλους ανάπτυξης. 28

3.1 Το σχήμα απεικονίζει τα βασικά σενάρια χρήσης της πλατφόρμας και τις ροές δεδομένων μεταξύ των data producers και data consumers. 40
3.2 Το σχήμα παρουσιάζει τα πέντε επίπεδα αρχιτεκτονικής. 42
3.3 Η αρχιτεκτονική του συστήματος, με έμφαση στις ροές δεδομένων μεταξύ των υποσυστημάτων και την αλληλεπίδραση μεταξύ τους. 43
3.4 Αναπαράσταση οντοτήτων και συσχετίσεων του μοντέλου δεδομένων της πλατφόρμας 48

4.1 UML Sequence Diagram για Data Upload σε βάση τύπου MongoDB. 56
4.2 UML Sequence Diagram για Data Retrieve σε βάση τύπου MongoDB. . . . 57
4.3 Αρχιτεκτονική της Big Data Platform. 66

Κατάλογος Εικόνων

4.1 Αρχική Σελίδα . . . 59
4.2 Σελίδα Διαχείρισης Assets . . . 60
4.3 Σελίδα διαχείρισης Assets με προβολή των Asset Details. . . . 60
4.4 Σελίδα διαχείρισης Assets με φόρμα προσθήκης νέου Asset. . . . 61
4.5 Σελίδα Διαχείρισης Connectors . . . 62
4.6 Σελίδα διαχείρισης Connectors με προβολή των Connector Details. . . . 62
4.7 Σελίδα Διαχείρισης Connectors με άνοιγμα προσθήκης ενός νέου Connector. 63
4.8 Σελίδα Data Retrieve . . . 63
4.9 Σελίδα Data Upload . . . 64
4.10 Σελίδα διαχείρισης Data Flows με φόρμα δημιουργίας νέου flow. . . . 64
4.11 Πίνακας διαχείρισης Data Flows με ενεργά και ανενεργά flows. . . . 65

5.1 Edge Counters . . . 70
5.2 Kafka topic sensorial-test . . . 70
5.3 Kafka topic short-test . . . 71
5.4 Kafka topic parquet-test . . . 71
5.5 Asset Catalog . . . 71
5.6 Connector Catalog . . . 72
5.7 Creation of Data Flow: sensorial-test (Kafka) -> sensorial-test (MongoDB) . 73
5.8 Collection Sensorial-test at MongoDB . . . 73
5.9 Creation of Data Flow: short-test (Kafka) -> short-test (MongoDB) . . . 74
5.10 Collection Short-test at MongoDB . . . 74
5.11 Creation of Data Flow: parquet-test (Kafka) -> parquet-test (MongoDB) . . 75
5.12 Collection Parquet-test at MongoDB . . . 75
5.13 Parquet Files Retrieval from MongoDB as .zip . . . 76
5.14 Parquet Files Directory . . . 77
5.15 File Count in Parquet Directory after unzip . . . 77
5.16 Data Retrieve from collection sensorial-test (MongoDB) . . . 78
5.17 Data Upload to collection sensorial-test (MongoDB) . . . 79
5.18 Data Count in Sensorial-test Collection at MongoDB . . . 79

Κατάλογος Πινάκων

2.1 Σύγκριση τεχνολογιών με βάση τους κεντρικούς άξονες. 35

3.1 Περιγραφή υποσυστημάτων της αρχιτεκτονικής της Big Data Platform 44
3.2 Λειτουργίες Discoverability του Big Data Platform 45
3.3 Λειτουργίες Discoverability για data assets 46
3.4 Λειτουργίες Discoverability για connectors 46
3.5 Λειτουργίες Data exchange του Connector. 47
3.6 Λειτουργίες Discoverability για data flows 48
3.7 Πεδία της οντότητας Asset 49
3.8 Πεδία της οντότητας Connector 50
3.9 Πεδία της οντότητας Flow 51

4.1 Λειτουργίες Data exchange του MongoDB Connector 54
4.2 Βασικές τεχνολογίες και βιβλιοθήκες που χρησιμοποιούνται στην πλατφόρμα 59

5.1 Σύνοψη Αξιολόγησης για το Σενάριο Α 76
5.2 Σύνοψη Αξιολόγησης για το Σενάριο Β 78
5.3 Σύνοψη Αξιολόγησης για το Σενάριο Γ 80
5.4 Σύνοψη Αξιολόγησης για όλα τα Σενάρια 80

Κεφάλαιο 1

Εισαγωγή

1.1 Περιγραφή Προβλήματος

Στη σύγχρονη βιομηχανία, η διαχείριση μεγάλου όγκου δεδομένων αποτελεί κρίσιμο παράγοντα για την αποτελεσματική παρακολούθηση, τον έλεγχο και τη βελτιστοποίηση των παραγωγικών διαδικασιών. Ένα χαρακτηριστικό παράδειγμα τέτοιου περιβάλλοντος είναι η μεταλλευτική βιομηχανία.

Κατά τη διάρκεια της λειτουργίας ενός ορυχείου παράγεται τεράστιος όγκος δεδομένων, που σχετίζεται με διαφορετικά στάδια της παραγωγικής αλυσίδας: από την αρχική γεωλογική εξερεύνηση και την εξόρυξη των πρώτων υλών, έως την επεξεργασία, τη διαχείριση των αποβλήτων και την παρακολούθηση της περιβαλλοντικής επίπτωσης. Τα δεδομένα αυτά προέρχονται από ποικίλες πηγές — αισθητήρες, συστήματα παρακολούθησης και λογισμικά σχεδίασης — και συνήθως διαφέρουν ως προς τη μορφή, τη συχνότητα, την ποιότητα και το πρωτόκολλο επικοινωνίας [1].

Η ετερογένεια αυτή καθιστά εξαιρετικά δύσκολη τη διαλειτουργικότητα μεταξύ των συστημάτων. Κάθε ορυχείο ή μονάδα παραγωγής μπορεί να εφαρμόζει διαφορετικές τεχνολογίες, να χρησιμοποιεί δικά του εργαλεία και να έχει διαφοροποιημένες υποδομές IT, γεγονός που οδηγεί σε κατακερματισμό των πληροφοριών. Η απουσία ενιαίων προτύπων και η έλλειψη κοινής γλώσσας μεταφοράς και επεξεργασίας δεδομένων επιβαρύνουν περαιτέρω την κατάσταση, καθιστώντας δύσκολη την ολοκληρωμένη αξιοποίηση των πληροφοριών [2]. Ενδέχεται, λοιπόν, μεγάλο μέρος των διαθέσιμων δεδομένων να παραμένει ανεκμετάλλευτο, με αποτέλεσμα να περιορίζονται οι δυνατότητες βελτιστοποίησης και αυτοματοποίησης των διαδικασιών.

Η κατάσταση αυτή έχει άμεσες επιπτώσεις στην επιχειρησιακή λειτουργία. Οι εταιρείες δυσκολεύονται να λάβουν αποφάσεις σε πραγματικό χρόνο, να εντοπίσουν εγκαίρως αποκλίσεις και κινδύνους, να αξιολογήσουν με ακρίβεια την αποδοτικότητα ή να συμμορφωθούν με τις απαιτήσεις των ρυθμιστικών αρχών. Τέλος, η έλλειψη καθαρών και καλά δομημένων δεδομένων παρεμποδίζει την εφαρμογή τεχνολογιών τεχνητής νοημοσύνης και μηχανικής μάθησης, που θα μπορούσαν να συμβάλουν στη βελτίωση της παραγωγικότητας και της ασφάλειας [3, 4].

1.2 Αντικείμενο Διπλωματικής

Η παρούσα διπλωματική εργασία στοχεύει στην αντιμετώπιση των προκλήσεων που σχετίζονται με τη διαχείριση ετερογενών δεδομένων στην εξορυκτική βιομηχανία, μέσω της ανάπτυξης μιας ενιαίας και επεκτάσιμης πλατφόρμας συλλογής, οργάνωσης, αποθήκευσης και ανάκτησης πληροφοριών. Η υλοποίηση της εντάσσεται στο πλαίσιο του ευρωπαϊκού έργου MINE.IO[1], το οποίο προωθεί την ψηφιοποίηση και τη διασύνδεση κρίσιμων διαδικασιών, με στόχο την ενίσχυση της αποδοτικότητας, της διαφάνειας και της τεκμηριωμένης λήψης αποφάσεων.

Η προτεινόμενη πλατφόρμα λειτουργεί ως ένα ενιαίο σημείο πρόσβασης σε δεδομένα που προέρχονται από πολλαπλές πηγές και υφιστάμενα υποσυστήματα, αντιμετωπίζοντας αποτελεσματικά τον κατακερματισμό των πληροφοριών. Ο σχεδιασμός της δίνει έμφαση στην επεκτασιμότητα και την παραμετροποίηση, έτσι ώστε να μπορεί να συνδεθεί με ετερογενείς τεχνολογικές υποδομές χωρίς να τις αντικαθιστά, αλλά ενοποιώντας τις σε ένα συνεκτικό περιβάλλον διαχείρισης και αξιοποίησης δεδομένων. Παράλληλα, παρέχεται δημόσιο API, μέσω του οποίου καθίσταται δυνατή η προγραμματιστική πρόσβαση στα δεδομένα, με δυνατότητες προσθήκης, μεταφοράς και αναζήτησης περιεχομένου.

1.2.1 Συνεισφορά

Η συνεισφορά της πλατφόρμας συνοψίζεται στα εξής σημεία:

1. Μελέτη της υφιστάμενης κατάστασης και των απαιτήσεων που σχετίζονται με τη συλλογή και διαχείριση ετερογενών δεδομένων στα ορυχεία, με έμφαση στην ανάγκη για ενοποιημένη πρόσβαση και αξιοποίηση.

2. Σχεδιασμός και ανάπτυξη μιας ενιαίας και επεκτάσιμης πλατφόρμας για την αποθήκευση, μεταφορά και ανάκτηση δεδομένων από πολλαπλές πηγές, με υποστήριξη τόσο ιστορικών δεδομένων όσο και ροών σε πραγματικό χρόνο.

3. Υλοποίηση μηχανισμών διασύνδεσης με πολλαπλά ετερογενή αποθετήρια, με στόχο τη διευκόλυνση της διαλειτουργικότητας μεταξύ συστημάτων.

4. Ανάπτυξη ευρετηρίου και δημόσιου API, μέσω των οποίων καθίσταται εφικτή η εύκολη αναζήτηση και αξιοποίηση των δεδομένων από εξωτερικά συστήματα και αναλυτικά εργαλεία.

5. Δημιουργία μιας λειτουργικής υποδομής που μπορεί να υποστηρίξει την ανάλυση δεδομένων, τη λήψη αποφάσεων και την παρακολούθηση κρίσιμων διαδικασιών σε ορυχεία ή άλλες βιομηχανικές εφαρμογές.

1.3 Οργάνωση Κειμένου

Η εργασία αυτή είναι οργανωμένη σε έξι κεφάλαια: Στο Κεφάλαιο 2 δίνεται το θεωρητικό υπόβαθρο των βασικών τεχνολογιών που σχετίζονται με τη διπλωματική και παραδείγματα

[1] MINE.IO: https://mineio-horizon.eu

σχετικών εργασιών που μελετήθηκαν για τον σχεδιασμό της αρχιτεκτονικής της πλατφόρμας. Στο Κεφάλαιο 3 αναλύονται οι απαιτήσεις του συστήματος, η αρχιτεκτονική του και ο διαχωρισμός τους σε υποσυστήματα. Στο Κεφάλαιο 4 περιγράφεται η υλοποίηση των επιμέρους υποσυστημάτων και η διασύνδεση των δεδομένων μέσω API. Στο Κεφάλαιο 5 παρουσιάζεται η μεθοδολογία αξιολόγησης του συστήματος με αναλυτική παρουσίαση των ελέγχων που πραγματοποιήθηκαν. Η εργασία ολοκληρώνεται με το Κεφάλαιο 6, στο οποίο διατυπώνονται τα συνολικά συμπεράσματα και προτείνονται ενδεικτικές κατευθύνσεις για μελλοντική έρευνα.

Κεφάλαιο 2

Θεωρητικό υπόβαθρο και Σχετικές Εργασίες

Στο κεφάλαιο αυτό παρουσιάζονται συνοπτικά οι βασικές έννοιες και τεχνολογίες που αποτελούν το θεωρητικό υπόβαθρο για την ανάπτυξη της πλατφόρμας. Στο δεύτερο μέρος, περιλαμβάνονται οι σχετικές τεχνολογίες και προσεγγίσεις που αξιοποιήθηκαν και συνέβαλαν στον σχεδιασμό της αρχιτεκτονικής της πλατφόρμας.

2.1 Θεωρητικό Υπόβαθρο

2.1.1 Internet of Things (IoT)

Το Internet of Things (IoT) [5], γνωστό και ως Διαδίκτυο των Πραγμάτων, αναφέρεται σε δίκτυα συσκευών που επικοινωνούν και ανταλλάσσουν πληροφορίες αυτόματα χωρίς την ανθρώπινη παρέμβαση. Πρόκειται για συσκευές που συνδέονται ασύρματα είτε στο διαδίκτυο είτε σε τοπικά δίκτυα και διαθέτουν αισθητήρες ή μηχανισμούς, επιτρέποντας την αυτόματη συλλογή, μετάδοση και επεξεργασία δεδομένων. Παραδείγματα τέτοιων συσκευών είναι τα αυτοκίνητα, οι αισθητήρες, καθώς και μικρότερες έξυπνες συσκευές όπως ρολόγια, drones ή ηχεία. Η ανάπτυξη του IoT έχει οδηγήσει στην εξέλιξη απλών αντικειμένων σε ένα πολύπλοκο οικοσύστημα που παρέχει ολοκληρωμένες υπηρεσίες, όπως μετάδοση δεδομένων, ανάλυση πληροφοριών, και επικοινωνία μεταξύ των συσκευών. Η διαρκής αύξηση αυτών των συσκευών έχει οδηγήσει σε σημαντική παραγωγή δεδομένων, που χαρακτηρίζονται ως «Μεγάλα Δεδομένα» (Big Data) [6].

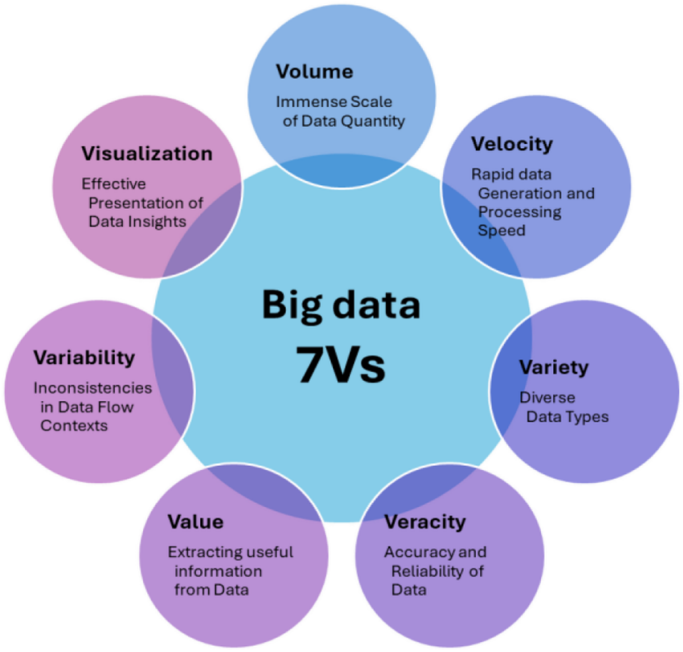
2.1.2 Μεγάλα Δεδομένα (Big Data)

Ο όρος «Μεγάλα Δεδομένα» (Big Data) [7] αναφέρεται σε δεδομένα εξαιρετικά μεγάλου όγκου και πολυπλοκότητας, τα οποία δεν μπορούν να αποθηκευτούν ή να αναλυθούν αποτελεσματικά με παραδοσιακές μεθόδους. Παραδείγματα συναντώνται στα χρηματιστήρια, όπου παράγονται τεράστιες ροές δεδομένων καθημερινά [8], αλλά και στα κοινωνικά δίκτυα όπως το Facebook, που αποθηκεύουν εκατοντάδες terabytes φωτογραφιών, βίντεο και κειμένων κάθε μέρα [9].

Αρχικά, τα Big Data περιγράφονταν από τα τρία βασικά χαρακτηριστικά τους, γνωστά και ως 3Vs [10]: τον όγκο (Volume), την ταχύτητα (Velocity) και την ποικιλία (Variety). Ο όγκος αφορά την τεράστια κλίμακα δεδομένων που φτάνει σε επίπεδα zettabytes, η ταχύτητα τη συνεχή και ταχύτατη ροή νέων δεδομένων, ενώ η ποικιλία σχετίζεται με τις διαφορετικές

μορφές τους, που μπορεί να είναι δομημένες (όπως δεδομένα αισθητήρων), αδόμητες (όπως εικόνες και βίντεο) ή ημι-δομημένες (όπως αρχεία JSON[1] ή XML[2]).

Με την πάροδο του χρόνου, όπως φαίνεται και στο παρακάτω Σχήμα 2.1, το αρχικό αυτό μοντέλο επεκτάθηκε ώστε να περιλαμβάνει περισσότερες διαστάσεις, όπως η αξιοπιστία των δεδομένων (Veracity), η αξία τους (Value), η μεταβλητότητα (Variability) και η εγκυρότητα (Validity), συνθέτοντας έτσι το πλαίσιο των 7V [11].



Σχήμα 2.1: *Τα επτά βασικά χαρακτηριστικά των Μεγάλων Δεδομένων (7V's of Big Data). Το σχήμα παρουσιάζει τις κύριες ιδιότητες που βοηθούν στην κατανόηση και τη διαχείριση μεγάλου όγκου δεδομένων.*

Η **ποικιλία (Variety)** των δεδομένων συνδέεται άμεσα με την ετερογένειά τους. Η ετερογένεια εμφανίζεται σε τρία επίπεδα. Αρχικά, υπάρχει η μορφολογική ετερογένεια, η οποία αφορά τους διαφορετικούς τύπους δεδομένων. Υπάρχουν (α) τα δομημένα δεδομένα (structured data), τα οποία έχουν σαφώς καθορισμένο σχήμα, όπως πίνακες, (β) τα ημι-δομημένα δεδομένα (semi-structured data), τα οποία έχουν ευέλικτο σχήμα, όπως αρχεία JSON ή XML, και (γ) τα μη δομημένα δεδομένα (unstructured data), όπως εικόνες, βίντεο ή ελεύθερο κείμενο, που δεν ακολουθούν συγκεκριμένο σχήμα. Στη συνέχεια, υπάρχει η συντακτική ετερογένεια, η οποία σχετίζεται με τους διαφορετικούς τρόπους οργάνωσης των δεδομένων, όπως πίνακες, αρχεία JSON ή CSV. Τέλος, η σημασιολογική ετερογένεια αφορά τις διαφορετικές έννοιες ή ερμηνείες που μπορεί να έχουν τα ίδια δεδομένα σε διαφορετικά πλαίσια.

Η ύπαρξη διαφορετικών τύπων δεδομένων οδηγεί αναπόφευκτα στη χρήση πολλαπλών αποθετηρίων (polyglot persistence), δηλαδή στη συνδυαστική αξιοποίηση διαφορετικών τε-

[1]JSON: https://www.json.org/json-en.html
[2]XML: https://aws.amazon.com/what-is/xml/
[1]Πηγή Σχήματος 2.1: https://www.researchgate.net/publication/383825321_Machine_learning_in_polymer_additive_manufacturing_a_review

χνολογιών βάσεων για την κάλυψη ετερογενών αναγκών. Κάθε τύπος αποθετηρίου εξυπηρετεί συγκεκριμένο σκοπό: (α) οι σχεσιακές βάσεις (SQL) είναι ιδανικές για δομημένα δεδομένα με σαφές σχήμα και πολύπλοκα ερωτήματα, (β) οι μη σχεσιακές βάσεις (NoSQL) προσφέρουν ευελιξία και υποστήριξη για μεγάλης κλίμακας ή μη δομημένα δεδομένα [12], (γ) οι βάσεις χρονοσειρών (time-series databases) είναι βελτιστοποιημένες για δεδομένα που εξελίσσονται στον χρόνο [13], και (δ) τα **αποθετήρια αντικειμένων** (object stores) επιτρέπουν την αποθήκευση και ανάκτηση μεγάλων αρχείων και πολυμέσων με υψηλή επεκτασιμότητα. Ο κατάλληλος συνδυασμός αυτών των τεχνολογιών καθιστά δυνατή την αποτελεσματική διαχείριση πολύπλοκων, ετερογενών δεδομένων.

Ιδιαίτερη σημασία έχουν τα **δεδομένα με χρονική διάσταση** (temporal data), τα οποία περιλαμβάνουν πληροφορίες σχετικές με τον χρόνο, όπως χρονικές σφραγίδες, χρονικά διαστήματα ή εκδόσεις (versioning). Σε αντίθεση με τα στατικά δεδομένα, τα temporal δεδομένα αποτυπώνουν την εξέλιξη φαινομένων μέσα στον χρόνο, επιτρέποντας την αναπαραγωγή παλαιότερων καταστάσεων του συστήματος και την ανάλυση τάσεων. Για την αποθήκευση και επεξεργασία τους χρησιμοποιούνται εξειδικευμένα αποθετήρια, όπως οι time-series databases, που υποστηρίζουν λειτουργίες όπως windowing και downsampling για αποδοτική διαχείριση μεγάλου όγκου δεδομένων.

Η **ταχύτητα (Velocity)** σχετίζεται με τον ρυθμό με τον οποίο παράγονται, μεταφέρονται και καταναλώνονται τα δεδομένα. Ανάλογα με τον ρυθμό αυτό, διακρίνουμε δύο βασικές κατηγορίες: (α) **δεδομένα πραγματικού χρόνου** (real-time data) και (β) **ιστορικά δεδομένα** (historical data). Η διάκριση αυτή είναι κρίσιμη, καθώς καθορίζει τον τρόπο συλλογής, αποθήκευσης και ανάλυσης των δεδομένων.

Σε πολλές εφαρμογές, τα δεδομένα παράγονται συνεχώς, απαιτώντας ευέλικτες και αποδοτικές αρχιτεκτονικές επεξεργασίας. Δύο κύριες προσεγγίσεις χρησιμοποιούνται:

- **Stream processing**: επικεντρώνεται στην ανάλυση δεδομένων τη στιγμή που παράγονται. Είναι ιδανική για περιβάλλοντα όπου η έγκαιρη πληροφόρηση είναι κρίσιμη.

- **Batch processing**: αφορά την επεξεργασία μεγάλων ποσοτήτων δεδομένων εκ των υστέρων, όταν δεν απαιτείται άμεση απόκριση. Τα δεδομένα συγκεντρώνονται και αποθηκεύονται σε ομάδες προκαθορισμένου μεγέθους (batches), οι οποίες επεξεργάζονται περιοδικά.

Τα **δεδομένα ροής** (streaming data) φτάνουν συνεχώς και με απρόβλεπτο ρυθμό, χαρακτηρίζονται από χαμηλό λανθάνοντα χρόνο (low latency) και συνεχή παραγωγή γεγονότων (events). Η επεξεργασία τους επιτρέπει την άμεση αντίδραση σε αλλαγές και την υποστήριξη λειτουργιών σε πραγματικό χρόνο [14].

Παρά τα οφέλη που υπόσχονται, τα Big Data συνοδεύονται από σημαντικές προκλήσεις [15]. Η αποθήκευση και διαχείριση τεράστιων όγκων δεδομένων απαιτεί κατανεμημένες υποδομές και υψηλής απόδοσης υπολογιστικούς πόρους, ενώ η διασφάλιση της ποιότητας, της αξιοπιστίας και της προστασίας τους αποτελεί κρίσιμο ζήτημα — ιδίως όταν περιλαμβάνονται ευαίσθητες πληροφορίες, όπως προσωπικά δεδομένα (GDPR[3]).

[3]GDPR: https://gdpr-info.eu

Για την αποτελεσματική αντιμετώπιση αυτών των προκλήσεων απαιτούνται εξειδικευμένες τεχνολογίες σε όλα τα στάδια του κύκλου ζωής των δεδομένων — από τη συλλογή και μεταφορά έως την αποθήκευση, την επεξεργασία και την οπτικοποίησή τους. Ενδεικτικά παραδείγματα περιλαμβάνουν το Kafka[4] για τη διαχείριση ροών δεδομένων, το Spark[5] και το Flink[6] για επεξεργασία σε πραγματικό χρόνο ή σε παρτίδες, το HDFS[7] για κατανεμημένη αποθήκευση, καθώς και εργαλεία όπως τα Grafana[8] και Superset[9] για οπτικοποίηση και ζωντανή παρακολούθηση των δεδομένων.

Συνολικά, τα Μεγάλα Δεδομένα δεν αποτελούν απλώς ένα τεχνολογικό φαινόμενο αλλά και ένα σύνθετο ερευνητικό πεδίο, το οποίο συνδυάζει τεχνολογικές, οργανωτικές και κοινωνικές διαστάσεις. Η επιτυχής αξιοποίησή τους προϋποθέτει την ανάπτυξη νέων αρχιτεκτονικών, καινοτόμων εργαλείων ανάλυσης και αποτελεσματικών πολιτικών που θα διασφαλίζουν τόσο την αποδοτική όσο και την ασφαλή διαχείρισή τους.

2.1.3 Θεώρημα (CAP (Consistency, Availability, Partition Tolerance))

Το θεώρημα CAP (Consistency, Availability, Partition Tolerance) [16] αναφέρει ότι ένα κατανεμημένο σύστημα δεν μπορεί να παρέχει ταυτόχρονα και τις τρεις βασικές ιδιότητες: τη συνέπεια (Consistency), όπου όλοι οι κόμβοι βλέπουν τα ίδια δεδομένα κάθε στιγμή, τη διαθεσιμότητα (Availability), όπου κάθε αίτημα λαμβάνει απάντηση είτε επιτυχίας είτε αποτυχίας, και την ανοχή διαμερισμού (Partition Tolerance), που σημαίνει ότι το σύστημα συνεχίζει να λειτουργεί σωστά παρά τυχόν βλάβες ή διαχωρισμούς στο δίκτυο. Δεδομένου ότι κάθε κατανεμημένο σύστημα αντιμετωπίζει αναπόφευκτες βλάβες, η ανοχή διαμερισμού είναι απαραίτητη. Συνεπώς, τα συστήματα διαχωρίζονται ανάλογα με το ποιες ή ποια από τις άλλες δύο ιδιότητες θυσιάζουν. Τα συστήματα τύπου CP θυσιάζουν τη διαθεσιμότητα, διατηρώντας τη συνέπεια και την ανοχή διαμερισμού. Σε περίπτωση βλάβης, σταματούν τη λειτουργία των μη συνεπών κόμβων μέχρι να αποκατασταθεί η σύνδεση. Αντίθετα, τα συστήματα τύπου AP θυσιάζουν τη συνέπεια προκειμένου να διατηρήσουν τη διαθεσιμότητα, με όλους τους κόμβους να παραμένουν ενεργοί και να συγχρονίζονται αργότερα. Τέλος, τα συστήματα τύπου CA θυσιάζουν την ανοχή διαμερισμού για να διατηρήσουν τη συνέπεια και τη διαθεσιμότητα, όμως, υπάρχουν μόνο θεωρητικά, καθώς η ανοχή διαμερισμού είναι απαραίτητη στα πραγματικά κατανεμημένα συστήματα [17].

2.1.4 Service Oriented Architecture (SOA)

Η SOA (Service-Oriented Architecture) [18] είναι μια αρχιτεκτονική προσέγγιση όπου το λογισμικό χωρίζεται σε μικρές, ανεξάρτητες υπηρεσίες που επικοινωνούν μεταξύ τους μέσω κοινών προτύπων διεπαφών. Κάθε υπηρεσία εκτελεί μια συγκεκριμένη λειτουργία και μπορεί να χρησιμοποιηθεί ξανά σε διαφορετικά συστήματα [19].

Η χρήση της SOA επιτρέπει τον επαναχρησιμοποίηση κώδικα και την ταχύτερη ανάπτυξη

[4]Kafka: https://kafka.apache.org
[5]Spark: https://spark.apache.org
[6]Flink: https://flink.apache.org
[7]HDFS: https://hadoop.apache.org
[8]Grafana: https://grafana.com
[9]Superset: https://superset.apache.org

εφαρμογών. Για παράδειγμα, μια κοινή υπηρεσία ταυτοποίησης μπορεί να χρησιμοποιηθεί από πολλές εφαρμογές, χωρίς να χρειάζεται να ξαναγραφτεί. Παρόμοια, στην πλατφόρμα της παρούσας εργασίας, η πρόσβαση σε δεδομένα αποθετηρίων γίνεται μέσω της υπηρεσίας Asset Cataloguing API, αντί να υλοποιείται κάθε φορά από την αρχή.

Η SOA έχει πλεονεκτήματα σε σχέση με τις μονολιθικές αρχιτεκτονικές, καθώς οι υπηρεσίες είναι ευκολότερο να αναπτυχθούν, να συντηρηθούν και να επεκταθούν [20]. Πριν την καθιέρωσή της, οι ενοποιήσεις γίνονταν μέσω πολύπλοκων σημείο-προς-σημείο (point-to-point) συνδέσεων. Στην SOA, η τροποποίηση μιας υπηρεσίας δεν επηρεάζει την υπόλοιπη λειτουργία του συστήματος, κάτι που την καθιστά πιο ευέλικτη και προσαρμόσιμη.

Συχνά, η SOA υποστηρίζεται από έναν Enterprise Service Bus (ESB) [21], ο οποίος λειτουργεί ως κεντρικό σημείο επικοινωνίας μεταξύ υπηρεσιών. Ο ESB δρομολογεί αιτήματα, μετατρέπει πρωτόκολλα και διασφαλίζει την ομαλή συνεργασία διαφορετικών τεχνολογιών [22]. Αν και η SOA μπορεί να υλοποιηθεί και χωρίς ESB, αυτό συνεπάγεται αυξημένη πολυπλοκότητα και δυσκολία συντήρησης λόγω απευθείας συνδέσεων.

Τέλος, αν και η SOA και η αρχιτεκτονική μικροϋπηρεσιών (microservices) [23] χρησιμοποιούν παρόμοια ορολογία, λειτουργούν σε διαφορετικά επίπεδα και εξυπηρετούν διαφορετικές ανάγκες.

2.1.5 Software As A Service (SaaS)

Το Software as a Service (SaaS) είναι ένα μοντέλο στο οποίο οι χρήστες έχουν πρόσβαση σε εφαρμογές απευθείας από το διαδίκτυο, χωρίς να χρειάζεται να τις εγκαταστήσουν στους υπολογιστές τους. Όλη η υποδομή, η λειτουργία και οι ενημερώσεις του λογισμικού διαχειρίζονται από τον πάροχο, οπότε ο χρήστης χρειάζεται μόνο έναν browser ή ένα API για να το χρησιμοποιήσει [24].

Το κύριο πλεονέκτημά του είναι ότι μειώνει το κόστος και την πολυπλοκότητα για τον τελικό χρήστη, αφού δεν απαιτείται τοπική εγκατάσταση ή συντήρηση. Παράλληλα, επιτρέπει ευελιξία, εύκολες αναβαθμίσεις και προσαρμογή των πόρων ανάλογα με τις ανάγκες. Σε σχέση με την Service-Oriented Architecture (SOA), η οποία επικεντρώνεται στην οργάνωση ενός συστήματος σε ανεξάρτητες υπηρεσίες, το SaaS εστιάζει κυρίως στον τρόπο με τον οποίο παραδίδεται το λογισμικό στον χρήστη. Στην πράξη, το SaaS μπορεί να βασιστεί σε αρχές SOA, αλλά το πρώτο αφορά τον τρόπο με τον οποίο το λογισμικό παραδίδεται στον τελικό χρήστη, ενώ το δεύτερο τη δομή και την εσωτερική οργάνωση του [25].

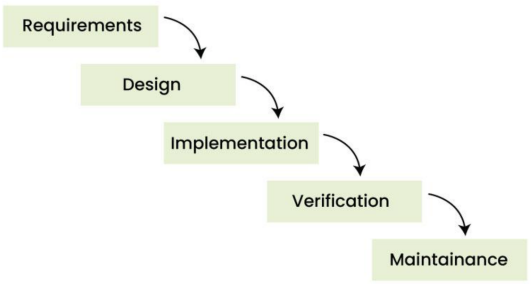
Παρ' όλα αυτά, το SaaS παρουσιάζει και πολλές προκλήσεις. Η ταυτόχρονη χρήση από πολλούς χρήστες [26] απαιτεί ιδιαίτερη προσοχή για να διασφαλιστεί η ιδιωτικότητα των δεδομένων και η σταθερή απόδοση. Επίσης, η διαθεσιμότητα του συστήματος πρέπει να είναι συνεχής, με εφεδρικά αντίγραφα και μηχανισμούς ανάκτησης σε περίπτωση βλάβης. Τέλος, η ανάγκη για Scalability σημαίνει ότι πρέπει να χρησιμοποιούνται σύγχρονες τεχνικές, όπως load balancing, κατανεμημένη αποθήκευση και fault tolerance, ώστε το σύστημα να ανταποκρίνεται χωρίς καθυστερήσεις.

Συνολικά, το SaaS αποτελεί έναν σύγχρονο τρόπο διάθεσης λογισμικού που διευκολύνει τους χρήστες, ενώ παράλληλα απαιτεί καινοτόμες τεχνικές για να ανταπεξέλθει στις αυξημένες απαιτήσεις ασφάλειας, αξιοπιστίας και επεκτασιμότητας.

2.1.6 Architectural Models

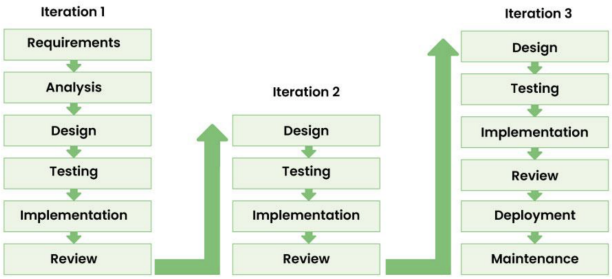
Η ανάπτυξη λογισμικού μπορεί να ακολουθήσει διάφορα μοντέλα αρχιτεκτονικής, τα οποία οργανώνουν τη διαδικασία ανάπτυξης και βοηθούν στη διαχείριση της πολυπλοκότητας. Κάθε μοντέλο έχει διαφορετική προσέγγιση, πλεονεκτήματα και περιορισμούς, και η επιλογή του εξαρτάται από το μέγεθος και τις απαιτήσεις του έργου [27].

Το Waterfall Model είναι σειριακό και απλό, με σαφή στάδια (ανάλυση απαιτήσεων, σχεδίαση, υλοποίηση, δοκιμή, συντήρηση). Είναι κατάλληλο για έργα με σταθερές απαιτήσεις, αλλά δεν προσφέρει ευελιξία σε αλλαγές [28]. Στην πράξη, αν προκύψει νέα απαίτηση, η διαδικασία πρέπει να επιστρέψει στην αρχή, αυξάνοντας χρόνο και κόστος.



Σχήμα 2.2: *Το Waterfall Model παρουσιάζει διαδοχικά στάδια ανάπτυξης. Έχει γραμμική ροή και έλλειψη επιστροφής σε προηγούμενα βήματα.*

Το Iterative Model αναπτύσσει το σύστημα σε μικρούς κύκλους, επιτρέποντας ευελιξία και συνεχή ανατροφοδότηση [29]. Σε κάθε κύκλο δημιουργείται μια βελτιωμένη λειτουργική έκδοση του συστήματος, που προσαρμόζεται με βάση τα σχόλια πριν ξεκινήσει ο επόμενος.

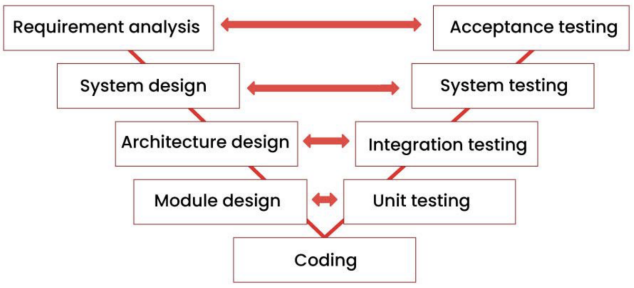


Σχήμα 2.3: *Το Iterative Model βασίζεται σε επαναλήψεις. Υπάρχει κυκλική διαδικασία ανάπτυξης και βελτίωσης.*

Το V-Model δίνει έμφαση στη δοκιμή σε κάθε στάδιο, συνδέοντας σχεδίαση και επαλήθευση [30]. Ενδεικτικά, οι απαιτήσεις αντιστοιχούν σε ελέγχους αποδοχής, η σχεδίαση συστήματος σε έλεγχο της συνολικής λειτουργίας και η λεπτομερής σχεδίαση σε ελέγχους επιμέρους τμημάτων του κώδικα.

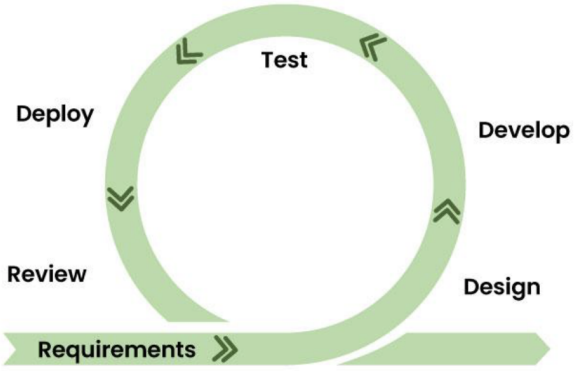
[2]Πηγή Σχήματος 2.2: https://www.geeksforgeeks.org/software-engineering/top-8-software-development-models-used-in-industry/

[3]Πηγή Σχήματος 2.3: https://www.geeksforgeeks.org/software-engineering/top-8-software-development-models-used-in-industry/



Σχήμα 2.4: *Το V-Model συνδέει κάθε φάση σχεδίασης με δοκιμές. Έχει συμμετρία σχεδίασης-ελέγχου.*

Το Agile Model βασίζεται σε μικρούς κύκλους ανάπτυξης και επιτρέπει γρήγορη προσαρμογή σε αλλαγές. Δίνει έμφαση στη συνεργασία με τον πελάτη και στη συνεχή βελτίωση του προϊόντος. Στο τέλος κάθε sprint, η ομάδα επανεξετάζει τις προτεραιότητες με βάση τις ανάγκες και τα σχόλια των χρηστών [31].

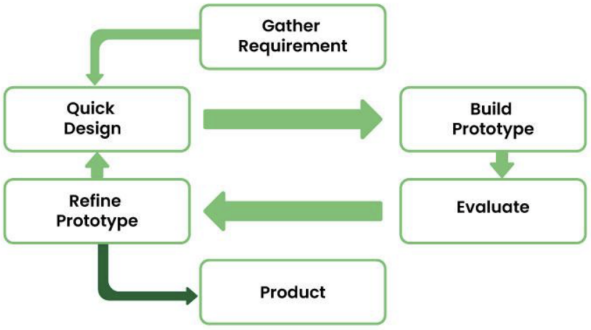


Σχήμα 2.5: *Το Agile Model βασίζεται σε σύντομα sprints. Υπάρχει ευελιξία και συνεχή ανατροφοδότηση.*

Το Prototype Model δημιουργεί ένα απλοποιημένο πρωτότυπο ώστε να οριστούν καλύτερα οι απαιτήσεις [32]. Με αυτόν τον τρόπο μειώνεται ο κίνδυνος παρερμηνείας πριν επενδυθεί προσπάθεια σε πλήρη υλοποίηση.

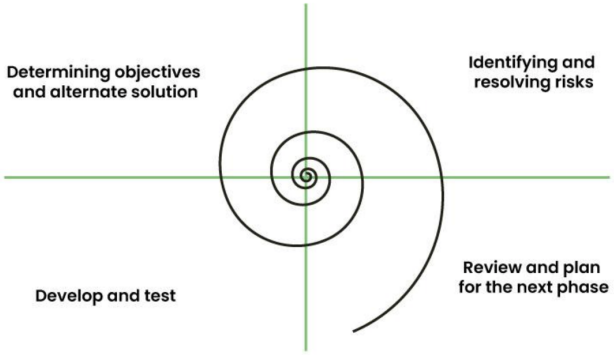
[4]Πηγή Σχήματος 2.4: https://www.geeksforgeeks.org/software-engineering/top-8-software-development-models-used-in-industry/

[5]Πηγή Σχήματος 2.5: https://www.geeksforgeeks.org/software-engineering/top-8-software-development-models-used-in-industry/



Σχήμα 2.6: *Το Prototype Model χρησιμοποιεί πρώιμο πρωτότυπο. Υπάρχει ροή μεταξύ σχεδίασης, αξιολόγησης και βελτίωσης.*

Το Spiral Model συνδυάζει στοιχεία από Waterfall και Iterative, με ιδιαίτερη έμφαση στη διαχείριση ρίσκου [33]. Κάθε κύκλος ξεκινά με εντοπισμό πιθανών κινδύνων και ολοκληρώνεται με την ανάπτυξη ενός μέρους του συστήματος και τον σχεδιασμό του επόμενου κύκλου.



Σχήμα 2.7: *Το Spiral Model εστιάζει στη σταδιακή ανάπτυξη με έλεγχο ρίσκου. Οι σπείρες αντιστοιχούν στους κύκλους ανάπτυξης.*

2.1.7 REST API

Μια Διεπαφή Προγραμματισμού Εφαρμογών (API) είναι ένα σύνολο κανόνων που καθορίζει πώς δύο προγράμματα επικοινωνούν μεταξύ τους. Με απλά λόγια, το API είναι η γέφυρα που επιτρέπει σε έναν client (π.χ. εφαρμογή ή χρήστη μέσω browser) να ζητήσει δεδομένα ή υπηρεσίες από έναν server και να λάβει απάντηση με συγκεκριμένο τρόπο. Οι πόροι (resources) που παρέχει το API μπορεί να είναι δεδομένα, αρχεία ή λειτουργίες, και κάθε υπηρεσία εκθέτει endpoints τα οποία μπορούν να κληθούν για ανάκτηση ή αποστολή πληροφορίας [34].

[6]Πηγή Σχήματος 2.6: https://www.geeksforgeeks.org/software-engineering/top-8-software-development-models-used-in-industry/

[7]Πηγή Σχήματος 2.7: https://www.geeksforgeeks.org/software-engineering/top-8-software-development-models-used-in-industry/

Οι Πελάτες (Clients) μπορεί να είναι είτε άνθρωποι μέσω ενός frontend (π.χ. browser, UI), είτε άλλες εφαρμογές/υπηρεσίες που στέλνουν αιτήματα στο API. Για παράδειγμα, ένας πελάτης μπορεί να τραβήξει δεδομένα που αφορούν ένα αποθετήριο (asset) από το Asset Cataloguing API, ή απλά να επισκεφθεί την ιστοσελίδα της πλατφόρμας στον browser του.

Η επικοινωνία γίνεται μέσω HTTP μηνυμάτων. Κάθε αίτημα (HTTP request) που στέλνει ο client περιλαμβάνει: τη μέθοδο HTTP, τον μοναδικό αναγνωριστή πόρου (URI), τυχόν headers με μεταδεδομένα (π.χ. Content-Type, Authorization), καθώς και σώμα δεδομένων (body) σε μορφή όπως JSON, XML ή CSV όταν απαιτείται. Οι πιο συνηθισμένες μέθοδοι είναι: GET για ανάκτηση δεδομένων, POST για δημιουργία νέων πόρων, PUT για πλήρη αντικατάσταση ενός πόρου, PATCH για μερική ενημέρωση και DELETE για διαγραφή [35]. Οι παράμετροι ενός αιτήματος μπορεί να είναι διαφορετικών τύπων. Οι path parameters βρίσκονται στο ίδιο το URL και προσδιορίζουν συγκεκριμένο πόρο (π.χ. GET /assets/asset_id). Οι query parameters προστίθενται μετά το ερωτηματικό και χρησιμοποιούνται για φιλτράρισμα ή αναζήτηση (π.χ. GET /api/data-retrieve?asset_id=67ced1...). Οι body parameters περιλαμβάνονται στο σώμα του αιτήματος και μεταφέρουν τα δεδομένα που αποστέλλει ο client, ενώ οι cookie parameters χρησιμοποιούνται κυρίως για σκοπούς αυθεντικοποίησης ή διατήρησης συνεδρίας.

Η απάντηση (HTTP response) που επιστρέφει ο server περιλαμβάνει τρία κύρια στοιχεία: τη γραμμή κατάστασης (status line), το σώμα απάντησης (message body) και τα headers. Η **γραμμή κατάστασης** περιέχει την έκδοση του πρωτοκόλλου HTTP, έναν τριψήφιο κωδικό κατάστασης και ένα λεκτικό μήνυμα, π.χ. HTTP/1.1 200 OK. Το **σώμα απάντησης** περιέχει τα δεδομένα που ζήτησε ο client, συνήθως σε JSON ή XML. Τα **headers** περιλαμβάνουν επιπλέον μεταδεδομένα, όπως τον τύπο περιεχομένου (Content-Type), τις οδηγίες προσωρινής αποθήκευσης (Cache-Control), την ημερομηνία και ώρα αποστολής ή πληροφορίες για τον server[10].

Η ασφάλεια στα REST APIs βασίζεται στην αυθεντικοποίηση (authentication) και την εξουσιοδότηση (authorization). Η αυθεντικοποίηση επιβεβαιώνει την ταυτότητα του client, ενώ η εξουσιοδότηση ελέγχει τα δικαιώματα πρόσβασης στους πόρους. Υπάρχουν διάφορες μέθοδοι για την εφαρμογή τους [36]. Η απλούστερη είναι η Basic Authentication, όπου το όνομα χρήστη και ο κωδικός αποστέλλονται σε κάθε αίτημα — πρακτική εύκολη αλλά όχι ιδιαίτερα ασφαλής. Η Bearer Authentication χρησιμοποιεί tokens, τα οποία αποστέλλονται στα headers και εκδίδονται από τον server μετά από επιτυχημένη είσοδο. Παρόμοια, τα API Keys είναι μοναδικά κλειδιά που ταυτοποιούν τον client, αλλά είναι πιο ευάλωτα σε υποκλοπή. Πιο προηγμένες λύσεις περιλαμβάνουν το OAuth, ένα διαδεδομένο πρωτόκολλο που βασίζεται σε προσωρινά access tokens, και τα JWT (JSON Web Tokens), τα οποία είναι αυτοπεριγραφικά και περιλαμβάνουν ενσωματωμένες πληροφορίες για τον χρήστη και τα δικαιώματά του, επιτρέποντας ασφαλή και αποδοτική διαχείριση πρόσβασης χωρίς διατήρηση κατάστασης στον server.

Για παράδειγμα, σε ένα αίτημα ανάκτησης δεδομένων από αποθετήριο, ο client στέλνει το αίτημα στο data-retrieve-endpoint της πλατφόρμας μαζί με το token. Ο server, αφού ελέγξει το token και βεβαιωθεί ότι ο χρήστης έχει τα κατάλληλα δικαιώματα, επιστρέφει την

[10] https://www.ibm.com/think/topics/rest-apis

απάντηση σε μορφή αρχείου (JSON ή άλλη). Έτσι διασφαλίζεται ότι μόνο οι εξουσιοδοτημένοι χρήστες μπορούν να έχουν πρόσβαση στους πόρους. Συνολικά, η χρήση ασφαλών μεθόδων αυθεντικοποίησης και εξουσιοδότησης ενισχύει την αξιοπιστία των REST APIs, ενώ παράλληλα επιτρέπει τον ακριβή έλεγχο της πρόσβασης, περιορίζοντας τη χρήση σε συγκεκριμένους χρήστες ή υπηρεσίες, ανάλογα με τα δικαιώματά τους.

2.1.8 Documenting REST APIs

Όπως αναφέρθηκε και προηγουμένως, για να μπορεί κάποιος να χρησιμοποιεί σωστά και αποτελεσματικά ένα API, είναι απαραίτητο να υπάρχει αναλυτική και κατανοητή τεκμηρίωση. Οι δύο πιο διαδεδομένοι τρόποι τεκμηρίωσης REST APIs είναι τα Postman Collections και τα OpenAPI Specifications.

Το Postman είναι μια πλατφόρμα που βοηθά στον σχεδιασμό, την υλοποίηση, τη χρήση και τον έλεγχο των APIs, κάνοντας τη διαχείρισή τους πιο εύκολη και αποδοτική. Τα Postman Collections είναι σύνολα από αιτήματα (requests) προς ένα API, όπου για κάθε αίτημα περιλαμβάνονται πληροφορίες όπως το σχήμα των δεδομένων που στέλνονται ή λαμβάνονται, οι πιθανοί παράμετροι (query και path), τα headers, τα στοιχεία ταυτοποίησης και παραδείγματα πραγματικών κλήσεων. Τα Postman Collections μπορούν να εξαχθούν σε μορφή JSON, ώστε να αναλυθούν εύκολα και να περιέχουν όλες τις σχετικές πληροφορίες για κάθε αίτημα, μαζί με διαθέσιμα παραδείγματα [37].

Οι OpenAPI Specifications (OAS) αποτελούν ανοιχτό πρότυπο τεκμηρίωσης σε YAML ή JSON, το οποίο μπορεί να χρησιμοποιηθεί τόσο σε Code-First όσο και σε Design-First προσεγγίσεις. Περιγράφουν ξεκάθαρα τις λειτουργίες και τους πόρους ενός API, από τον σχεδιασμό μέχρι την υλοποίηση και τον έλεγχο.

Το Swagger [38] είναι από τα πιο γνωστά εργαλεία υλοποίησης OpenAPI. Μέσα από τα αρχεία του (.json ή .yaml), καθορίζονται τα διαθέσιμα endpoints, οι παράμετροι και τα σχήματα δεδομένων. Το Swagger UI επιτρέπει την απευθείας οπτικοποίηση και δοκιμή των APIs μέσα από τον browser, διευκολύνοντας τους χρήστες να κατανοήσουν και να δοκιμάσουν τις λειτουργίες χωρίς επιπλέον εργαλεία. Κατά την ανάπτυξη των REST APIs, το έγγραφο Swagger δημοσιεύεται αυτόματα μέσω HTTP και ενημερώνεται δυναμικά ώστε να αντικατοπτρίζει τη ζωντανή υπηρεσία.

2.1.9 Enterprise Integration Patterns

Τα Enterprise Integration Patterns (EIPs) είναι design patterns που προσφέρουν δομημένες λύσεις σε κοινές προκλήσεις integration σε enterprise systems [39]. Βασικές έννοιες είναι το Message Channel, μέσω του οποίου οι εφαρμογές επικοινωνούν στέλνοντας μηνύματα (message queues, publish-subscribe), το Message, δηλαδή η δομή δεδομένων που μεταφέρει πληροφορίες, και το Message Endpoint, τα σημεία εισόδου και εξόδου των μηνυμάτων. Κάθε επικοινωνία περιλαμβάνει έναν Message Producer και έναν Message Consumer. Τα κυριότερα patterns που χρησιμοποιούνται είναι το Point-to-Point Channel, το Publish-Subscribe Channel, ο Message Router, ο Message Translator και ο Message Aggregator. Στο Point-to-Point Channel, υπάρχει ένας sender και ένας receiver. Για παράδειγμα, ένας Kafka producer στέλνει δεδομένα σε ένα Kafka topic και ένας consumer

τα αποθηκεύει σε MongoDB. Στο Publish-Subscribe Channel, ένας sender μεταδίδει μηνύματα σε πολλούς subscribers. Έτσι, διαφορετικοί consumers μπορούν να επεξεργάζονται τα ίδια δεδομένα παράλληλα — π.χ. ένας τα αποθηκεύει για ιστορικότητα και άλλος κάνει real-time analytics ή ενεργοποιεί alarms.

Επόμενο βασικό μοτίβο είναι ο Message Router, ο οποίος δρομολογεί τα μηνύματα σε διαφορετικά channels ανάλογα με το περιεχόμενό τους. Υπάρχουν τρεις κύριοι τύποι:(α) Content-Based Router, που επιλέγει τη διαδρομή βάσει περιεχομένου, (β) Message Filter, που φιλτράρει μηνύματα με βάση κριτήρια (π.χ. ημερομηνία στη MongoDB), και (γ) Recipient List, που στέλνει μηνύματα σε πολλούς παραλήπτες. Στην πλατφόρμα, για παράδειγμα, όταν δημιουργείται ένα data flow ανάμεσα σε δύο αποθετήρια (assets), ο κώδικας ελέγχει τον τύπο της πηγής (π.χ. MongoDB, Kafka, External API) και δρομολογεί τα δεδομένα στην κατάλληλη υπηρεσία.

Ο Message Translator μετατρέπει τη μορφή των δεδομένων για να διασφαλίσει συμβατότητα μεταξύ συστημάτων. Για παράδειγμα, κατά την ανάλυση CSV αρχείων στον Mongodb Connector, τα δεδομένα μετατρέπονται από CSV σε JSON format για αποθήκευση στη MongoDB.

Ο Message Aggregator συγκεντρώνει πολλά μηνύματα σε ένα. Χρησιμοποιείται για batch processing, ή correlation σχετικών μηνυμάτων, όπως στο Mongodb Connector, κατά την ανάλυση αρχείων (CSV ή JSON), όπου τα δεδομένα συλλέγονται σε batches και στη συνέχεια εισάγονται μαζικά στη MongoDB με μία μόνο λειτουργία insertMany(). Ένα ακόμη παράδειγμα είναι το Kafka Message Aggregation. Τα Kafka μηνύματα συλλέγονται σε batches πριν την αποστολή με τη βοήθεια ενός μετρητή.

Το Asynchronous Messaging υλοποιείται μέσω Kafka για real-time streaming, όπου οι producers στέλνουν δεδομένα αισθητήρων και οι consumers τα επεξεργάζονται ασύγχρονα, χωρίς συγχρονισμό. Τέλος, ένα τελευταίο μοτίβο που χρησιμοποιείται είναι το Microservices Architecture, που επιτρέπει στα services να είναι ανεξάρτητα και να επικοινωνούν μέσω REST APIs.

Η χρήση των EIPs προσφέρει Reusability, έτοιμες λύσεις σε κοινά προβλήματα integration, Maintainability, αφού πρόκειται για καθαρά patterns που κάνουν τα συστήματα ευκολότερα στην κατανόηση και τη συντήρησή τους, Reliability αφού έχουν Built-in error handling και recovery mechanisms και τέλος Flexibility επιτρέποντας συνδυασμό patterns για πολύπλοκα σενάρια.

2.2 Σχετικές Εργασίες και Παραδείγματα

Η μελέτη και ανάπτυξη πλατφορμών Big Data προσελκύει ολοένα και μεγαλύτερο ερευνητικό ενδιαφέρον, καθώς οι οργανισμοί καλούνται να διαχειριστούν τεράστιους όγκους ετερογενών δεδομένων από διαφορετικές πηγές. Η πρόκληση πλέον δεν περιορίζεται στην αποθήκευση, αλλά και στην ενοποιημένη πρόσβαση, την ανάλυση σε πραγματικό χρόνο και τη διασφάλιση ιστορικότητας. Στο πλαίσιο αυτό, θα εξετάσουμε επιλεγμένες τεχνολογίες — Hadoop, Spark, Flink, Apache Sqoop και Apache NiFi — με βάση τέσσερις βασικούς άξονες: ενοποίηση ετερογενών δεδομένων από πολλαπλά αποθετήρια, διαχείριση ιστορικών δεδομένων, επεξεργασία ροών σε πραγματικό χρόνο, και συνδυασμό batch και streaming

αρχιτεκτονικών.

2.2.1 Παρουσίαση Τεχνολογιών

Οι υποενότητες που ακολουθούν παρουσιάζουν τις επιλεγμένες τεχνολογίες και συνοψίζουν τα κύρια χαρακτηριστικά και τις δυνατότητες τους.

Apache Hadoop

Το Apache Hadoop αποτελεί ένα από τα πιο διαδεδομένα πρότυπα για την αποθήκευση και επεξεργασία δεδομένων σε περιβάλλοντα Big Data. Πρόκειται για μια πλατφόρμα ανοικτού κώδικα που επιτρέπει την κατανεμημένη εκτέλεση υπολογιστικών διεργασιών σε πολλούς διακομιστές. Αποτελείται από δύο βασικά μέρη: το MapReduce, που εκτελεί τις διεργασίες, και το HDFS (Hadoop Distributed File System), που αποθηκεύει τα δεδομένα. Τα πλεονεκτήματά του είναι η ευελιξία, η κλιμακωσιμότητα, το χαμηλό κόστος και η αξιοπιστία στη διαχείριση μεγάλου όγκου δομημένων και αδόμητων δεδομένων [40].

Apache Spark

Το Apache Spark [41] αποτελεί μια ενιαία πλατφόρμα για κατανεμημένη επεξεργασία δεδομένων. Χρησιμοποιεί ένα προγραμματιστικό μοντέλο παρόμοιο με το MapReduce, αλλά το επεκτείνει με μια δομή που ονομάζεται Resilient Distributed Datasets (RDDs) που επιτρέπει την κοινή χρήση και αποδοτική επεξεργασία δεδομένων. Ένα από τα βασικά του πλεονεκτήματα είναι ότι εκτελεί υπολογισμούς στη μνήμη, μειώνοντας σημαντικά τις καθυστερήσεις από την πρόσβαση στον δίσκο και βελτιώνοντας την ταχύτητα επαναλαμβανόμενων διεργασιών [42].

Apache Flink

Το Apache Flink [43] είναι ένα λογισμικό ανοικτού κώδικα για κατανεμημένη επεξεργασία δεδομένων σε μορφή batch και streaming [44]. Βασίζεται στην ιδέα ότι πολλές κατηγορίες εφαρμογών, όπως ανάλυση σε πραγματικό χρόνο, επεξεργασία ιστορικών δεδομένων και επαναληπτικοί αλγόριθμοι, μπορούν να υλοποιηθούν ως αξιόπιστες ροές δεδομένων. Μπορεί να λειτουργήσει αυτόνομα ή σε συνδυασμό με HDFS και YARN, αξιοποιώντας επεξεργασία στη μνήμη για καλύτερες επιδόσεις.

Apache Sqoop

Το Apache Sqoop είναι ένα εργαλείο ανοικτού κώδικα που χρησιμοποιείται για τη μαζική μεταφορά δεδομένων μεταξύ σχεσιακών βάσεων δεδομένων (RDBMS) και του οικοσυστήματος Hadoop. Διασυνδέεται με JDBC (Java Database Connectivity) για να εισάγει ολόκληρους πίνακες ή αποτελέσματα ερωτημάτων SQL σε HDFS, καθώς και να εξάγει δεδομένα πίσω σε RDBMS. Υποστηρίζει παράλληλο φόρτωμα (parallel import/export) και incremental φορτώσεις, επιτρέποντας τη μεταφορά μεγάλου όγκου δομημένων δεδομένων με αποδοτικό τρόπο [45].

Apache NiFi

Το Apache NiFi [46] είναι μια πλατφόρμα ανοικτού κώδικα για την αυτοματοποίηση και διαχείριση ροών δεδομένων (dataflows) μεταξύ ετερογενών συστημάτων. Υποστηρίζει πολλούς τύπους δεδομένων και πρωτοκόλλων, επιτρέποντας συλλογή, μετασχηματισμό και παράδοση δεδομένων σε πραγματικό χρόνο ή batch μορφή. Αν και είναι ιδιαίτερα ευέλικτο μέσω των processors, διαθέτει ένα αρκετά πολύπλοκο περιβάλλον χρήσης και απαιτεί προσεκτικό setup και maintenance, γεγονός που μπορεί να δυσκολέψει τη διαχείριση μεγάλων ροών δεδομένων.

2.2.2 Ετερογενή Δεδομένα από Πολλαπλά Αποθετήρια

Το Hadoop υποστηρίζει την αποθήκευση μεγάλου όγκου δεδομένων διαφόρων τύπων — δομημένων, ημιδομημένων και αδόμητων — μέσω του HDFS (Hadoop Distributed File System). Το σύστημα επιτρέπει την αποθήκευση δεδομένων από πολλές πηγές με ενιαίο τρόπο, χωρίς να απαιτείται προκαθορισμένο σχήμα.

Το Spark δεν προσφέρει δικό του σύστημα αποθήκευσης, αλλά ενσωματώνεται εύκολα με διαφορετικές πηγές δεδομένων. Μπορεί να διαβάζει και να επεξεργάζεται δεδομένα από HDFS, MapR File System, Cassandra, Amazon S3, καθώς και από προσαρμοσμένες πηγές. Με αυτόν τον τρόπο παρέχει ευελιξία στην πρόσβαση σε ετερογενή δεδομένα.

Αντίστοιχα, το Flink υποστηρίζει την ενσωμάτωση ετερογενών συνόλων δεδομένων, συμπεριλαμβανομένων δομημένων δεδομένων από σχεσιακές βάσεις, ημιδομημένων δεδομένων και αδόμητου κειμένου. Συνδέεται επίσης με HDFS και μπορεί να διαλειτουργήσει με πολλά διαφορετικά συστήματα αποθήκευσης, χωρίς να διαθέτει δικό του πρωτεύον σύστημα αποθήκευσης.

Το Apache Sqoop εστιάζει αποκλειστικά στη μεταφορά δομημένων δεδομένων από σχεσιακές βάσεις (RDBMS) προς το οικοσύστημα Hadoop και αντίστροφα. Δεν υποστηρίζει ημιδομημένα ή αδόμητα δεδομένα, γεγονός που το καθιστά ακατάλληλο για ετερογενή περιβάλλοντα.

Τέλος, το Apache NiFi μπορεί να υποστηρίζει ετερογενή δεδομένα κάθε μορφής — δομημένα, ημιδομημένα και αδόμητα μέσω ενός μεγάλου συνόλου ενσωματωμένων processors. Μπορεί να επεξεργάζεται δεδομένα από διαφορετικές πηγές, όπως βάσεις δεδομένων, APIs, αρχεία, message queues και αισθητήρες, καθώς και να τα παραδίδει σε πλήθος προορισμών.

2.2.3 Διαχείριση Ιστορικών Δεδομένων (Batch Processing)

Η επεξεργασία ιστορικών δεδομένων αποτελεί βασικό στοιχείο σε κάθε σύγχρονη πλατφόρμα δεδομένων. Συνήθως εφαρμόζεται σε διαδικασίες ETL (Extract, Transform and Load), συγκεντρωτική επεξεργασία δεδομένων, καθώς και στην εκπαίδευση και ενημέρωση μοντέλων μηχανικής μάθησης.

Το Hadoop υιοθετήθηκε ευρέως για batch processing, χάρη στην υλοποίηση του MapReduce, που επιτρέπει την κατανεμημένη επεξεργασία δεδομένων σε συστάδες υπολογιστών με πολλούς κόμβους. Ωστόσο, το Spark έχει πλέον καθιερωθεί ως η κυρίαρχη τεχνολογία για επεξεργασία μεγάλων δεδομένων, καθώς προσφέρει ταχύτατη επεξεργασία στη μνήμη (in-

memory), ξεπερνώντας τα προβλήματα καθυστερήσεων που προκαλούνται από τις λειτουργίες ανάγνωσης και εγγραφής του Hadoop [45]. Το Flink υποστηρίζει επίσης batch processing με υψηλή απόδοση, παρέχοντας μηχανισμούς για αποδοτική επεξεργασία μεγάλων συνόλων δεδομένων σε κατανεμημένα περιβάλλοντα. Επιπλέον, το Apache NiFi μπορεί να χρησιμοποιηθεί ως εργαλείο εισαγωγής και μεταφοράς δεδομένων σε batch μορφή, επιτρέποντας την αυτόματη συλλογή και δρομολόγηση δεδομένων από πολλαπλές πηγές προς αποθηκευτικά ή υπολογιστικά συστήματα, λειτουργώντας έτσι συμπληρωματικά σε διαδικασίες ETL [47].

2.2.4 Ροές Δεδομένων και Επεξεργασία σε Πραγματικό Χρόνο (Stream Processing)

Η επεξεργασία δεδομένων σε πραγματικό χρόνο αποτελεί βασικό στοιχείο των σύγχρονων πληροφοριακών συστημάτων, καθώς επιτρέπει την άμεση ανάλυση και αντίδραση σε μεταβαλλόμενες συνθήκες. Το Spark Streaming μπορεί να λαμβάνει ζωντανά δεδομένα εισόδου και να τα χωρίζει σε micro-batches, τα οποία επεξεργάζεται η μηχανή του Spark για να παράγει την τελική ροή αποτελεσμάτων σε μορφή παρτίδων. Η τεχνική του micro-batching επιτρέπει την αντιμετώπιση μιας ροής δεδομένων ως διαδοχικών μικρών παρτίδων, αλλά εισάγει επιπλέον υπολογιστικό κόστος λόγω του προγραμματισμού και της διαχείρισης εργασιών. Αντίθετα, το Flink προσφέρει τα πλεονεκτήματα της προσωρινής αποθήκευσης (buffering) χωρίς αυτό το επιπλέον κόστος, επιτυγχάνοντας υψηλή απόδοση σε σενάρια πραγματικού ή σχεδόν πραγματικού χρόνου, όπου τα αποτελέσματα πρέπει να είναι διαθέσιμα σχεδόν ταυτόχρονα με τη δημιουργία των δεδομένων [45]. Το Apache NiFi μπορεί επίσης να υποστηρίξει ροές δεδομένων σε πραγματικό χρόνο, λειτουργώντας ως εργαλείο συνεχούς εισαγωγής και δρομολόγησης δεδομένων από πολλαπλές πηγές σε διαφορετικούς προορισμούς, με δυνατότητες backpressure και ευέλικτης διαχείρισης ρυθμού ροής. Αντίθετα, το Apache Sqoop έχει σχεδιαστεί κυρίως για μαζικές batch μεταφορές δεδομένων και δεν υποστηρίζει εγγενώς streaming λειτουργίες, γεγονός που το καθιστά ακατάλληλο για σενάρια πραγματικού χρόνου [47].

2.2.5 Σύνοψη Τεχνολογιών

Στο παρακάτω Πίνακα 2.1, παρουσιάζονται συνοπτικά τα κύρια χαρακτηριστικά των τεχνολογιών που εξετάστηκαν, ταξινομημένα με βάση την ικανότητά τους να διαχειρίζονται ετερογενή δεδομένα, να συνδέονται με πολλαπλά αποθετήρια και τον τύπο επεξεργασίας που υποστηρίζουν.

Εργαλείο	Ετερογενή Δεδομένα	Πολλαπλά Αποθετήρια	Τύπος Επεξεργασίας
Hadoop	Δομημένα, Ημιδομημένα, Αδόμητα	APIs - HDFS	Batch
Spark	Δομημένα, Ημιδομημένα, Αδόμητα	HDFS, S3, Cassandra, JDBC βάσεις κ.ά.	Micro-batch
Flink	Δομημένα, Ημιδομημένα, Αδόμητα	HDFS, Kafka, JDBC, Elasticsearch, S3 κ.ά.	Batch & Streaming
Apache Sqoop	Δομημένα	RDBMS - HDFS	Batch
Apache NiFi	Δομημένα, Ημιδομημένα, Αδόμητα	Βάσεις, APIs, queues, filesystems, cloud	Batch & Streaming

Πίνακας 2.1: *Σύγκριση τεχνολογιών με βάση τους κεντρικούς άξονες.*

2.2.6 Παρατηρήσεις

Οι τρέχουσες τάσεις στα συστήματα Big Data δίνουν όλο και μεγαλύτερη έμφαση στην ενοποίηση ετερογενών δεδομένων, στον συνδυασμό ροών σε πραγματικό χρόνο με ιστορικά δεδομένα και σε λύσεις που προσαρμόζονται εύκολα σε διαφορετικά περιβάλλοντα. Στις προηγούμενες ενότητες παρουσιάστηκαν μερικές από τις υπάρχουσες λύσεις για την ενοποίηση ετερογενών δεδομένων που αποκαλύπτει ένα σύνθετο τοπίο τεχνολογιών και προσεγγίσεων, καθένα από τα οποία εξυπηρετεί συγκεκριμένες ανάγκες. Ωστόσο, η πρακτική εφαρμογή αυτών των λύσεων σε βιομηχανικά περιβάλλοντα, όπως η βιομηχανία εξόρυξης, αποκαλύπτει σημαντικά κενά και προκλήσεις που δεν καλύπτονται αποτελεσματικά από τις γενικές πλατφόρμες.

Εντοπισμός Κενών στις Υπάρχουσες Λύσεις

- **Γενικός σχεδιασμός:** Οι περισσότερες πλατφόρμες είναι γενικής χρήσης και απαιτούν εκτενή προσαρμογή για σενάρια βιομηχανικού IoT. Δεν υποστηρίζουν εγγενώς βιομηχανικά πρωτόκολλα και δεν συνδυάζονται εύκολα με υπάρχοντα συστήματα, όπως τα PLM.

- **Περιορισμοί εργαλείων ανά τύπο δεδομένων:** Το Sqoop εξυπηρετεί αποκλειστικά δομημένα δεδομένα από RDBMS (batch μόνο). Το NiFi υποστηρίζει ετερογενή δεδομένα αλλά απαιτεί προσεκτικό setup/maintenance. Το Hadoop δεν προσφέρει streaming.

- **Διαχείριση αποθετηρίων και σχήματος δεδομένων:** Οι πλατφόρμες δεν προσφέρουν ενιαίο μηχανισμό για να διαχειρίζονται αλλαγές στη δομή των δεδομένων (schema evolution) ή στη μορφή και τον τύπο των αποθετηρίων που τα φιλοξενούν. Για τον σκοπό αυτό απαιτούνται πρόσθετα εργαλεία (π.χ. Asset & Schema Registry, κανόνες ελέγχου συνέπειας).

- **Παρακολούθηση Δεδομένων:** Η παρακολούθηση της ροής δεδομένων και ο εντοπισμός σφαλμάτων δεν παρέχονται εξίσου σε όλα τα εργαλεία. Το Apache NiFi διαθέτει ενσωματωμένο μηχανισμό ιχνηλασιμότητας (data provenance) και πλήρη ορατότητα

μέσα στο ίδιο περιβάλλον, κάτι που διευκολύνει τη διαχείριση. Αντίθετα, εργαλεία όπως Hadoop, Spark και Flink απαιτούν εξωτερικά συστήματα παρακολούθησης, ενώ το Sqoop παρέχει μόνο βασικά λογς.

- **Επιχειρησιακή πολυπλοκότητα και κόστος:** Το NiFi διαθέτει ισχυρό περιβάλλον εργασίας που μπορεί να γίνει πολύπλοκο και όχι ιδιαίτερα φιλικό για τον χρήστη όσον αφορά απλά και συγκεκριμένα σενάρια χρήσης. Αντίθετα, τα Spark και Flink δεν παρέχουν γραφικό περιβάλλον από μόνα τους, γεγονός που αυξάνει την πολυπλοκότητα στη ρύθμιση και διαχείρισή τους.

Συνεισφορά της Πλατφόρμας

Η πλατφόρμα αντιμετωπίζει αυτά τα κενά προσφέροντας μια ενιαία αρχιτεκτονική ενοποίησης ειδικά σχεδιασμένη για τη βιομηχανία εξόρυξης.

- **Ενοποίηση δεδομένων:** Η πλατφόρμα χρησιμοποιεί connectors —δηλαδή εξειδικευμένα εργαλεία διασύνδεσης— για την απευθείας σύνδεση με βιομηχανικές πηγές δεδομένων, όπως Kafka, MongoDB και PLM συστήματα. Με αυτόν τον τρόπο, επιτρέπει την εύκολη ενσωμάτωση υπαρχουσών υποδομών και την παράλληλη αξιοποίηση δεδομένων σε πραγματικό χρόνο. Η δυνατότητα δυναμικής διαχείρισης connectors χωρίς διακοπή λειτουργίας μειώνει δραστικά τον χρόνο εγκατάστασης και συντήρησης.

- **Διαχείριση μορφών δεδομένων:** Η πλατφόρμα μπορεί να διαβάζει και να επεξεργάζεται πολλαπλές μορφές δεδομένων (JSON, CSV, Parquet, αρχεία), αναγνωρίζοντας αυτόματα το είδος τους και μετασχηματίζοντάς τα σε κοινό σχήμα. Παράλληλα, προσφέρει ενιαία πρόσβαση σε διαφορετικούς τύπους βάσεων δεδομένων (relational, NoSQL, time-series). Έτσι αποφεύγεται η χρήση πολλών διαφορετικών εργαλείων και μειώνεται η καθυστέρηση στην επεξεργασία.

- **Υβριδική αρχιτεκτονική:** Συνδυάζει batch και streaming επεξεργασία, οργανώνοντας δεδομένα σε αποθετήρια. Έτσι, η πλατφόρμα μπορεί να παρακολουθεί τι συμβαίνει σε πραγματικό χρόνο και να αναλύει εύκολα πώς εξελίσσονται οι τάσεις με την πάροδο του χρόνου.

- **Απλοποίηση διασύνδεσης:** Μέσω RESTful APIs και αρχιτεκτονικής microservices, η πλατφόρμα διευκολύνει την επικοινωνία μεταξύ διαφορετικών συστημάτων, μειώνοντας την πολυπλοκότητα και το λειτουργικό κόστος. Επιπλέον, η χρήση containerization (π.χ. Docker, Kubernetes) επιτρέπει την ταχεία εγκατάσταση και προσαρμογή της πλατφόρμας σε διαφορετικά περιβάλλοντα.

Οι σχεδιαστικές αυτές επιλογές τοποθετούν την προτεινόμενη πλατφόρμα στο σημείο συνάντησης της εφαρμοσμένης τεχνολογίας με τις ανοιχτές ερευνητικές προκλήσεις. Συνδυάζοντας λειτουργίες που ήδη προτείνουν τα υπάρχοντα πρότυπα, προσφέρει μια ενοποιημένη, δυναμική και παραμετροποιήσιμη λύση, η οποία καλύπτει τις ανάγκες των σύγχρονων εξελίξεων και ταυτόχρονα παρέχει εξειδίκευση στη μεταλλευτική βιομηχανία, κάτι που κανένα

από τα τρία πρότυπα δεν καλύπτει πλήρως. Παράλληλα, δημιουργεί το υπόβαθρο για μελλοντικές δυνατότητες και την εξέλιξη ενοποιημένων και επεκτάσιμων πλατφορμών Big Data, ικανών να αξιοποιηθούν σε ένα ευρύ φάσμα επιστημονικών και βιομηχανικών πεδίων.

Κεφάλαιο 3

Ανάλυση και Σχεδίαση

Στο κεφάλαιο αυτό παρουσιάζεται η αρχιτεκτονική της πλατφόρμας, γίνεται ο διαχωρισμός του στα επιμέρους υποσυστήματα, ενώ στη συνέχεια περιγράφονται οι εφαρμογές της πλατφόρμας.

3.1 Ανάλυση απαιτήσεων

Το Big Data Platform έχει σχεδιαστεί με βάση ένα σύνολο λειτουργικών και μη λειτουργικών απαιτήσεων, ώστε να εξασφαλίζει την αποδοτική διαχείριση και επεξεργασία των δεδομένων IoT. Η ανάλυση πραγματοποιήθηκε στο πλαίσιο του ευρωπαϊκού έργου MINE.IO, σε συνεργασία με τους πιλότους του έργου — δηλαδή τα ορυχεία που συμμετέχουν σε αυτό — μέσα από συζητήσεις και ανάλυση κοινών αναγκών. Η διαδικασία αυτή ανέδειξε δύο βασικές ομάδες απαιτήσεων για την ανάπτυξη της πλατφόρμας: (α) απαιτήσεις σχετικές με τα δεδομένα και (β) απαιτήσεις από την πλευρά των πιλότων.

1. Απαιτήσεις σχετικές με τα δεδομένα. Στους πιλότους του έργου διαπιστώθηκε ότι τα δεδομένα που παράγονται στα ορυχεία παρουσιάζουν μεγάλη ετερογένεια ως προς το είδος και τη μορφή τους (Data Type Versatility). Πρόκειται για μετρήσεις από αισθητήρες, γεωχωρικά δεδομένα, εικόνες, αρχεία JSON ή CSV, ακόμα και απλά αρχεία καταγραφής. Η πλατφόρμα, επομένως, πρέπει να μπορεί να διαχειριστεί αυτήν την ποικιλία, ώστε να καλύπτει τις διαφορετικές ανάγκες τόσο των παραγωγικών διαδικασιών όσο και των ίδιων των ορυχείων. Επιπλέον, επειδή κάθε πιλότος χρησιμοποιεί διαφορετικές τεχνολογίες παραγωγής και αποθήκευσης, προκύπτει η ανάγκη η πλατφόρμα να μην τις αντικαθιστά, αλλά να συνδέεται μαζί τους και να παρέχει τον καταλληλότερο τρόπο αποθήκευσης ανά περίπτωση (Individual Data Storage Solutions). Στο ίδιο πλαίσιο, κρίνεται απαραίτητη η δρομολόγηση των δεδομένων να γίνεται ανάλογα με την πηγή τους (source-type routing), ώστε οι πληροφορίες να οδηγούνται στο σωστό αποθετήριο, είτε αυτό αφορά ροές δεδομένων που προέρχονται από Kafka, είτε εξωτερικά APIs, είτε άλλες βάσεις δεδομένων.

2. Απαιτήσεις από την πλευρά των πιλότων. Παράλληλα, τα δεδομένα θα πρέπει να είναι άμεσα προσβάσιμα και αξιοποιήσιμα από τους πιλότους (Discoverability). Αυτό σημαίνει ότι θα πρέπει να μπορούν να τα αναζητήσουν, να τα ανακτήσουν, να τα μεταφέρουν ή να εισάγουν ιστορικά δεδομένα σε συγκεκριμένα αποθετήρια. Για να επιτευχθεί αυτό, η πλατφόρμα πρέπει να σχεδιαστεί με γνώμονα την αξιοπιστία (Reliability). Ένα χαρακτηριστικό παράδειγμα αποτελεί η παρακολούθηση της θερμοκρασίας σε ένα ορυχείο μέσω αισθητήρα:

αν η σύνδεση χαθεί προσωρινά, το σύστημα οφείλει να επαναφέρει αυτόματα την επικοινωνία (connection retry) και να συνεχίσει τη ροή δεδομένων χωρίς απώλειες. Σε περίπτωση σφάλματος, απαιτούνται μηχανισμοί ανθεκτικότητας (fault tolerance) και καταγραφής (logging), ώστε το πρόβλημα να εντοπίζεται άμεσα και να αποκαθίσταται η λειτουργία (Maintainability). Για τον λόγο αυτόν, είναι αναγκαία μια αρθρωτή αρχιτεκτονική (modular architecture) με σαφώς ορισμένα REST APIs και δυνατότητες συνεχούς παρακολούθησης (monitoring).

Εξίσου σημαντική είναι και η υψηλή απόδοση (High Performance). Σε περίπτωση που ο αισθητήρας θερμοκρασίας παρουσιάσει απότομη μεταβολή, τα δεδομένα πρέπει να διακινηθούν αμέσως στο σύστημα μέσω ασύγχρονης επικοινωνίας (asynchronous messaging), ώστε να παραχθεί ειδοποίηση σε πραγματικό χρόνο (Real-Time Streaming Support) και να αποτραπεί πιθανή βλάβη ή ατύχημα. Χωρίς τέτοιες δυνατότητες, κρίσιμες μετρήσεις ενδέχεται να καθυστερήσουν, με σοβαρές επιπτώσεις στην παραγωγή και την ασφάλεια. Παράλληλα, για εφαρμογές που απαιτούν ανάλυση σε βάθος χρόνου, η πλατφόρμα οφείλει να υποστηρίζει την αποθήκευση και αξιοποίηση ιστορικών δεδομένων (Historical Data Storage). Έτσι, διαμορφώνονται δύο βασικά κανάλια επικοινωνίας: ένα βασισμένο σε HTTP για την αποθήκευση και χρήση ιστορικών δεδομένων, και ένα δεύτερο, προσανατολισμένο σε ανταλλαγή μηνυμάτων (Message-oriented), για τη ροή δεδομένων σε πραγματικό χρόνο.

Αξίζει να σημειωθεί ότι η παραπάνω λογική ευθυγραμμίζεται με το CAP theorem (βλ. 2.1.3): η πλατφόρμα ακολουθεί την προσέγγιση AP (Availability + Partition Tolerance), δίνοντας προτεραιότητα στη διαθεσιμότητα και την αντοχή σε σφάλματα δικτύου, ενώ αποδέχεται ένα επίπεδο eventual consistency. Στο πλαίσιο εφαρμογών IoT, όπου η συνεχής ροή δεδομένων είναι πιο κρίσιμη από την άμεση συνέπεια σε όλα τα αποθετήρια, αυτή η επιλογή εξασφαλίζει ότι η πλατφόρμα συνεχίζει να λειτουργεί ομαλά και χωρίς διακοπές, ακόμη κι αν παρουσιαστούν δυσκολίες ή βλάβες στο δίκτυο.

Συνολικά, οι παραπάνω απαιτήσεις οδηγούν σε μια ευέλικτη και προσαρμόσιμη πλατφόρμα, ικανή να ανταποκρίνεται σε διαφορετικά σενάρια και να ενσωματώνεται εύκολα σε πολλές εφαρμογές και υπηρεσίες.

3.2 Σενάρια Χρήσης

Στο Σχήμα 3.1 παρουσιάζονται ενδεικτικά σενάρια χρήσης της πλατφόρμας, τα οποία καλύπτουν τις απαιτήσεις που αναλύθηκαν παραπάνω και αναδεικνύουν τις βασικές λειτουργίες της. Η επικοινωνία με την πλατφόρμα πραγματοποιείται μέσω διαφορετικών τύπων συστημάτων, όπως οι υπηρεσίες (Services) και τα Gateways που βρίσκονται στο πεδίο. Τα συστήματα αυτά, ανάλογα με το σενάριο χρήσης, μπορούν να λειτουργούν με δύο τρόπους: ως παραγωγοί δεδομένων (data producers), όταν τροφοδοτούν την πλατφόρμα με δεδομένα, και ως καταναλωτές δεδομένων (data consumers), όταν αξιοποιούν τα δεδομένα που αυτή αποθηκεύει ή διαχειρίζεται.

Σε πολλές περιπτώσεις, τα ίδια συστήματα ενσωματώνουν και τους δύο ρόλους. Για παράδειγμα, ένα predictive maintenance module στο cloud καταναλώνει δεδομένα αισθητήρων, τα επεξεργάζεται και επιστρέφει προβλέψεις πίσω στην πλατφόρμα. Αντίστοιχα, ένα gateway συλλέγει μετρήσεις από τον εξοπλισμό και τις αποθηκεύει, αλλά μπορεί επίσης να λάβει δεδομένα από την πλατφόρμα και να τα προωθήσει ξανά στο πεδίο. Με αυτόν τον τρόπο,

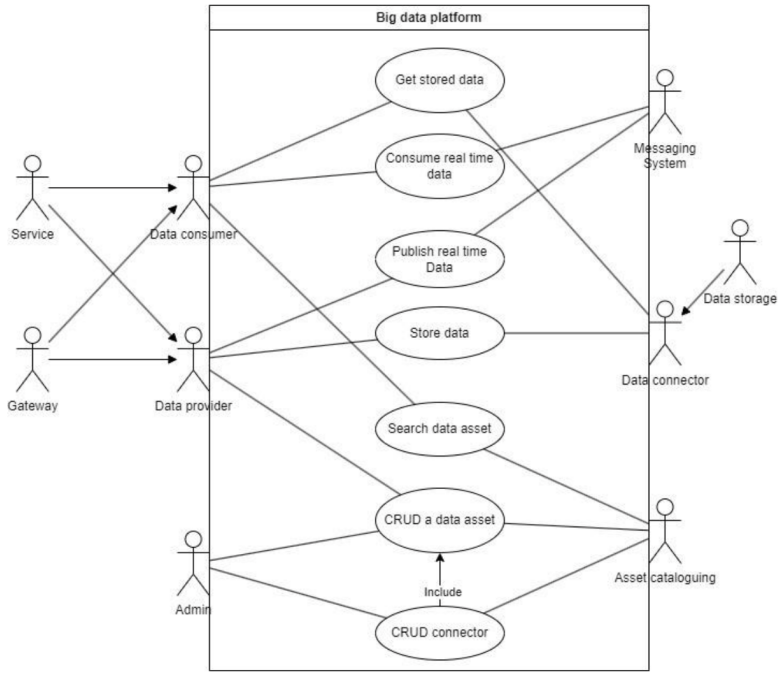
οι ρόλοι των Service και Gateway συνδέονται άμεσα με τους ρόλους των data producers και data consumers, υποστηρίζοντας σενάρια αμφίδρομης ροής δεδομένων.

Στη συνέχεια, όπως αναφέρθηκε στη προηγούμενη υποενότητα, κάθε πιλότος χρησιμοποιεί διαφορετικές τεχνολογίες. Αυτό οδηγεί σε διαφορετικές λύσεις αποθήκευσης. Έτσι, η πλατφόρμα πρέπει να υποστηρίζει ένα γενικό και ευέλικτο μοντέλο, ικανό να συνδεθεί με πολλαπλά αποθετήρια. Η επικοινωνία αυτή διατηρείται αφαιρετική μέσω εξειδικευμένων connectors, οι οποίοι συνδέουν την πλατφόρμα με κάθε αποθηκευτική λύση.

Παράλληλα, ακολουθώντας τις απαιτήσεις που αναλύθηκαν, τα δεδομένα διαχωρίζονται σε δύο βασικές κατηγορίες: δεδομένα πραγματικού χρόνου (real-time data), τα οποία διακινούνται και επεξεργάζονται μέσω messaging systems, και ιστορικά δεδομένα (historical/-batch data), τα οποία αποθηκεύονται μέσω Data connectors σε βάσεις δεδομένων (Data Storage), όπως η MongoDB, για να είναι διαθέσιμα σε αναλύσεις μεγάλης διάρκειας.

Για να διασφαλιστεί η εύκολη αναζήτηση και αξιοποίηση των δεδομένων (discoverability), κάθε πληροφορία συνοδεύεται από μια αναλυτική περιγραφή (data asset), η οποία περιλαμβάνει τα μεταδεδομένα και τα στοιχεία αποθήκευσης. Οι περιγραφές αυτές οργανώνονται σε έναν ειδικό κατάλογο (Asset Cataloguing), ο οποίος επιτρέπει στους χρήστες να εντοπίζουν και να αξιοποιούν γρήγορα τα δεδομένα που χρειάζονται.

Συνολικά, οι βασικές λειτουργίες της πλατφόρμας – όπως ο ορισμός και η διαχείριση data assets, η δημιουργία και παραμετροποίηση connectors, η αναζήτηση δεδομένων, η αποθήκευση και ανάκτηση πληροφοριών, καθώς και η διαχείριση ροών πραγματικού χρόνου – υλοποιούν στην πράξη τις απαιτήσεις που τέθηκαν, εξασφαλίζοντας την ενιαία και αποτελεσματική αξιοποίηση των ετερογενών δεδομένων. Οι λειτουργίες αυτές παρουσιάζονται συνοπτικά στο παρακάτω σχήμα.



Σχήμα 3.1: *Το σχήμα απεικονίζει τα βασικά σενάρια χρήσης της πλατφόρμας και τις ροές δεδομένων μεταξύ των data producers και data consumers.*

Με βάση το παραπάνω σχήμα, παρουσιάζεται ένα ενδεικτικό σενάριο χρήσης για την καλύτερη κατανόηση της ροής δεδομένων. Ένας αισθητήρας (data provider) σε ένα ορυχείο ενός πιλότου του έργου παράγει δεδομένα σε πραγματικό χρόνο. Τα δεδομένα αυτά συλλέγονται και καταναλώνονται από μια υπηρεσία (service, π.χ. Kafka) που λειτουργεί ως data consumer. Όταν η ποσότητα των δεδομένων ξεπεράσει ένα προκαθορισμένο όριο, αποστέλλεται περιοδικά ένα batch αρχείο στην Big Data Platform για περαιτέρω επεξεργασία ή αποθήκευση.

Ο data consumer αναζητά αρχικά το αντίστοιχο asset στον κατάλογο Asset Cataloguing, ώστε να εντοπίσει τον προκαθορισμένο προορισμό των δεδομένων σύμφωνα με τις ανάγκες του πιλότου (π.χ. αποθήκευση σε βάση δεδομένων ή αποστολή σε εξωτερικό σύστημα). Για τον σκοπό αυτό, η πλατφόρμα στέλνει αίτημα Search data asset στο Asset Cataloguing, το οποίο με χρήση λειτουργιών CRUD εντοπίζει το κατάλληλο data asset, αντλεί τα μεταδεδομένα του και αναγνωρίζει τον τύπο του αποθετηρίου.

Στη συνέχεια, γίνεται ανάκτηση των δεδομένων μέσω της λειτουργίας Get stored data και ενεργοποιείται ο αντίστοιχος Data Connector, ο οποίος λειτουργεί ταυτόχρονα ως data consumer και data provider: λαμβάνει τα δεδομένα από την πλατφόρμα και τα προωθεί στο κατάλληλο Data Storage. Με αυτόν τον τρόπο, διασφαλίζεται η αυτοματοποιημένη και ομοιογενής διαχείριση της ροής δεδομένων από το πεδίο έως το τελικό σημείο αποθήκευσης.

3.3 Περιγραφή αρχιτεκτονικής

Με βάση τις απαιτήσεις και τα σενάρια χρήσης που παρουσιάστηκαν στις προηγούμενες ενότητες, η εσωτερική αρχιτεκτονική του Big Data Platform απεικονίζεται στο Σχήμα 3.2. Η αρχιτεκτονική οργανώνεται σε πέντε επίπεδα.

Στο ανώτερο επίπεδο εντάσσεται η επικοινωνία με τα υπόλοιπα υποσυστήματα της πλατφόρμας. Η επικοινωνία αυτή υποστηρίζεται με δύο τρόπους: (α) μέσω της λογικής request–response (http-based), και (β) μέσω της λογικής message-oriented, δηλαδή με ανταλλαγή μηνυμάτων σε πραγματικό χρόνο μέσω Real-time streams (Kafka).

Το δεύτερο επίπεδο είναι το επίπεδο διασύνδεσης, στο οποίο περιλαμβάνονται το REST API, που υλοποιεί τον μηχανισμό αιτήσεων και απαντήσεων και το Web Inteface (UI), το οποίο χρησιμοποιεί ο χρήστης για να επικοινωνήσει με τη πλατφόρμα.

Ακολουθεί το επίπεδο διαχείρισης δεδομένων (Asset Cataloguing), το οποίο λειτουργεί ως μητρώο όπου αποθηκεύονται οι περιγραφές και τα μεταδεδομένα των διαθέσιμων data assets.

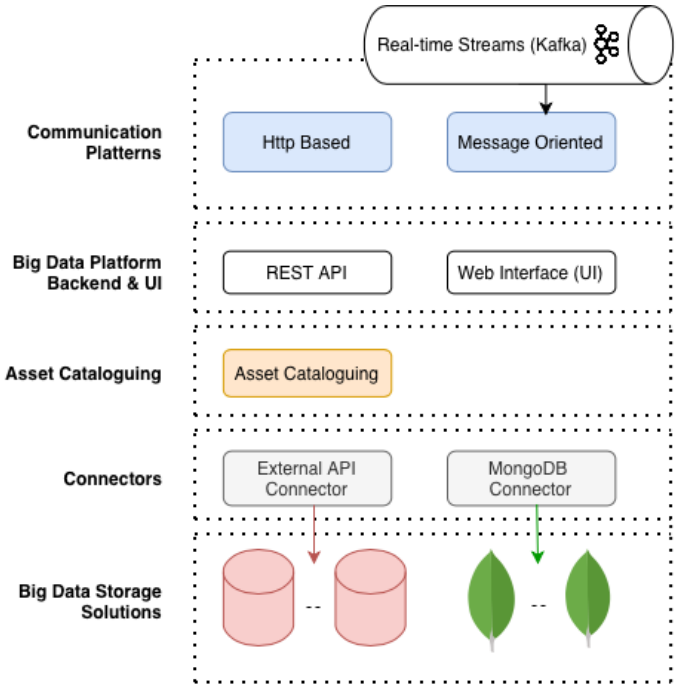
Στο προτελευταίο επίπεδο βρίσκονται οι connectors. Οι connectors είναι εξειδικευμένα εργαλεία που έχουν σχεδιαστεί για να συνδέονται με συγκεκριμένες λύσεις αποθήκευσης δεδομένων και να εκτελούν λειτουργίες ανάκτησης, προσθήκης ή μεταφοράς δεδομένων. Στο Σχήμα 3.2 απεικονίζονται με βελάκια ανοιχτού ροζ ή πράσινου χρώματος. Τα βελάκια με ανοιχτό ροζ χρώμα αντιστοιχούν στους γενικούς connectors, οι οποίοι υποστηρίζουν τη διασύνδεση με External APIs ή λύσεις Big Data Storage. Αντίστοιχα, τα βελάκια με ανοιχτό πράσινο χρώμα αντιπροσωπεύουν τους MongoDB connectors, οι οποίοι συνδέονται αποκλειστικά με βάσεις δεδομένων MongoDB (πράσινα φύλλα).

Τέλος, στο κατώτερο επίπεδο βρίσκεται το επίπεδο αποθήκευσης (Storage Solutions),

το οποίο περιλαμβάνει ετερογενείς λύσεις, όπως βάσεις δεδομένων, εξωτερικά APIs και συστήματα αποθήκευσης μεγάλου όγκου δεδομένων. Η σύνδεσή τους με το Big Data Platform γίνεται μέσω των εξειδικευμένων connectors, γεγονός που επιτρέπει τη δυναμική ενσωμάτωση πολλαπλών αποθετηρίων χωρίς την ανάγκη αντικατάστασής τους. Με αυτόν τον τρόπο καλύπτεται η απαίτηση για ευελιξία και επεκτασιμότητα στη διαχείριση δεδομένων.

Η συνολική αυτή δομή βασίζεται στη φιλοσοφία της Service-Oriented Architecture (SOA) (βλ. 2.1.4). Σύμφωνα με την προσέγγιση αυτή, το σύστημα διαχωρίζεται σε ανεξάρτητες υπηρεσίες που επικοινωνούν μεταξύ τους μέσω καλά ορισμένων διεπαφών (APIs). Η επικοινωνία υλοποιείται με τρεις κύριες μεθόδους: (α) μέσω REST APIs για αιτήσεις HTTP, (β) μέσω Message Queues, όπως το Kafka, για ασύγχρονη ανταλλαγή μηνυμάτων, και (γ) μέσω απευθείας συνδέσεων σε βάσεις δεδομένων για πρόσβαση σε δεδομένα.

Για παράδειγμα, όταν ένας χρήστης επιθυμεί να ανακτήσει δεδομένα από ένα διαθέσιμο asset, η διαδικασία ξεκινά με την αποστολή ενός HTTP-βασισμένου αιτήματος προς το κεντρικό REST API της πλατφόρμας. Το αίτημα αυτό προωθείται στην υπηρεσία Asset Cataloguing, η οποία είναι υπεύθυνη για τον εντοπισμό της τοποθεσίας του συγκεκριμένου asset και την ανάκτηση των στοιχείων σύνδεσης με αυτό. Αφού εντοπιστούν τα στοιχεία, επιστρέφονται πίσω στο REST API, το οποίο πλέον γνωρίζει ποιος connector είναι κατάλληλος για τη συγκεκριμένη πηγή δεδομένων. Στη συνέχεια, το REST API ενεργοποιεί τον αντίστοιχο connector, ο οποίος συνδέεται με την κατάλληλη βάση ή εξωτερικό σύστημα, ανακτά τα ζητούμενα δεδομένα και τα επιστρέφει πίσω στο κεντρικό REST API. Κάθε ένα από τα παραπάνω βήματα υλοποιείται ως ανεξάρτητη υπηρεσία, γεγονός που εξασφαλίζει ευελιξία, δυνατότητα αυτόνομης λειτουργίας και υψηλή αξιοπιστία σε επίπεδο συστήματος.

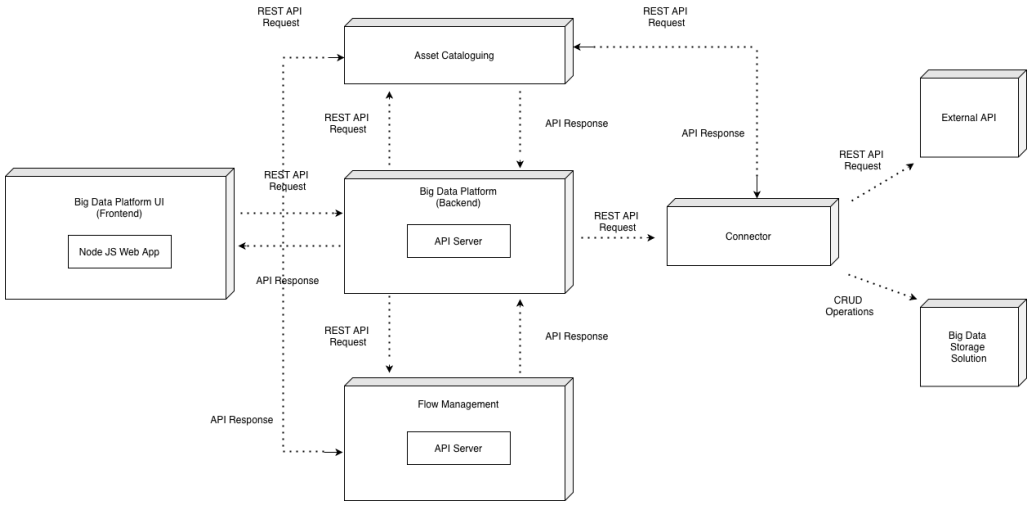


Σχήμα 3.2: *Το σχήμα παρουσιάζει τα πέντε επίπεδα αρχιτεκτονικής.*

3.4 Διαχωρισμός υποσυστημάτων

Στην προηγούμενη ενότητα παρουσιάστηκαν τα τέσσερα επίπεδα αρχιτεκτονικής της πλατφόρμας, τα οποία περιγράφουν αφαιρετικά τις βασικές λειτουργίες και τις μεταξύ τους σχέσεις. Σε αυτό το σημείο, τα ίδια αυτά επίπεδα μετασχηματίζονται σε διακριτά υποσυστήματα, με σαφώς καθορισμένους ρόλους και διεπαφές. Στο κέντρο της αρχιτεκτονικής βρίσκεται ο κόμβος διαχειριστής — το Big Data Platform Backend — ο οποίος λειτουργεί ως κεντρικός μηχανισμός ενορχήστρωσης και επικοινωνίας με όλα τα υπόλοιπα υποσυστήματα. Τα βασικά υποσυστήματα της πλατφόρμας είναι τα εξής: Big Data Platform Backend, Big Data Platform UI, Asset Cataloguing, Flow Management API και Connectors.

Το Σχήμα 3.3 αποτυπώνει τον τρόπο με τον οποίο διαχωρίζονται τα υποσυστήματα και παρουσιάζει με σαφήνεια τις ροές δεδομένων και την κατεύθυνση της επικοινωνίας μεταξύ τους.



Σχήμα 3.3: *Η αρχιτεκτονική του συστήματος, με έμφαση στις ροές δεδομένων μεταξύ των υποσυστηματων και την αλληλεπίδραση μεταξύ τους.*

Στη συνέχεια, ο Πίνακας 3.4 ακολουθώντας το Σχήμα 3.3 συνοψίζει για κάθε υποσύστημα το ρόλο, τις βασικές του υπευθυνότητες και τις διασυνδέσεις του με τα υπόλοιπα μέρη του συστήματος.

Υποσύστημα	**Περιγραφή**	**Διασυνδέσεις**
Big Data Platform Backend	Κεντρικός μηχανισμός ενορχήστρωσης που διαχειρίζεται την επικοινωνία μεταξύ όλων των υποσυστημάτων.	Big Data Platform UI, Asset Cataloguing, Flow Management, Connectors
Big Data Platform UI (Frontend)	Φιλικό web interface (UI), μέσω του οποίου οι χρήστες μπορούν να εκτελούν όλες τις λειτουργίες της πλατφόρμας	Big Data Platform Backend

Υποσύστημα	Περιγραφή	Διασυνδέσεις
Asset Cataloguing	Κεντρικό αποθετήριο που καταγράφει και διαχειρίζεται τα assets και connectors της πλατφόρμας.	Big Data Platform UI, Flow Management, Connectors
Flow Management API	Υποσύστημα υπεύθυνο για τη διαχείριση ροών ιστορικών δεδομένων ή δεδομένων πραγματικού χρόνου.	Big Data Platform Backend, Asset Cataloguing
Connectors	Περιλαμβάνει τις υπηρεσίες που επιτρέπουν στην πλατφόρμα να συνδεθεί με τους πιλότους και τα αποθετήριά τους για συλλογή, ανάκτηση και μεταφορά δεδομένων.	Big Data Platform Backend, Storage Solutions

Πίνακας 3.1: *Περιγραφή υποσυστημάτων της αρχιτεκτονικής της Big Data Platform*

Στην επόμενη ενότητα δίνεται λεπτομερής περιγραφή για καθένα από τα συστήματα που αναφέραμε. Η περιγραφή αυτή γίνεται με βάση τη λειτουργία τους όπως αυτή φαίνεται στο Σχήμα 3.3 και στον Πίνακα 3.4.

3.5 Περιγραφή υποσυστημάτων

Υποσύστημα Big Data Platform: Backend & Frontend (UI)

Το Big Data Platform λειτουργεί ως ο κεντρικός διαχειριστής (orchestrator) της πλατφόρμας. Αποτελεί τον βασικό πυρήνα (backend και frontend) που συντονίζει όλες τις λειτουργίες και τη ροή δεδομένων, παρέχοντας στους χρήστες ένα ενιαίο σημείο πρόσβασης. Ο ρόλος του είναι να ενορχηστρώνει την επικοινωνία μεταξύ όλων των επιμέρους υπηρεσιών, όπως το Asset Cataloguing, οι Connectors και το Flow Management API.

Μέσα από το Big Data Platform παρέχονται endpoints για όλες τις βασικές λειτουργίες: διαχείριση των assets, δημιουργία και παρακολούθηση ροών δεδομένων (data flows), ανάκτηση ή αποστολή δεδομένων, καθώς και διασύνδεση με εξωτερικά συστήματα μέσω των Connectors. Για παράδειγμα, το Big Data Platform API επικοινωνεί με τους connectors μέσω HTTP requests και αποστέλλοντας τις κατάλληλες παραμέτρους μπορεί να λαμβάνει τα αντίστοιχα δεδομένα. Με αυτόν τον τρόπο, ο χρήστης δεν χρειάζεται να αλληλεπιδρά ξεχωριστά με κάθε υπηρεσία, αλλά έχει ένα ενοποιημένο σημείο διαχείρισης.

Στον παρακάτω πίνακα παρουσιάζονται συνοπτικά τα βασικά API endpoints και η λειτουργία τους:

Λειτουργία	Περιγραφή
HTTP [GET /api/asset-cataloguing	Λίστα με όλα τα καταχωρημένα assets.
HTTP [GET /api/flows	Λίστα με όλα τα data flows.
HTTP [POST /api/data-flow/	Δημιουργία ροής δεδομένων μεταξύ δύο assets
HTTP [POST api/stop-data-flow/id	Διακοπή ροής με συγκεκριμένο ID
HTTP [DELETE /api/data-flow/id	Διαγραφή ροής με συγκεκριμένο ID
HTTP [GET /api/data-retrieve/asset_id	Ανάκτηση δεδομένων από αποθετήριο.
HTTP [POST /api/data-upload/asset_id	Αποστολή δεδομένων από αρχείο σε αποθετήριο.

Πίνακας 3.2: *Λειτουργίες Discoverability του Big Data Platform*

Επιπλέον, το Big Data Platform διαθέτει ένα φιλικό web interface (UI), μέσω του οποίου οι χρήστες μπορούν να εκτελούν όλες τις παραπάνω λειτουργίες με ευκολία. Από το περιβάλλον αυτό, μπορούν να δημιουργούν και να διαχειρίζονται ροές δεδομένων, να επιλέγουν αποθετήρια (assets) και connectors, να ορίζουν παραμέτρους όπως το batch size και το format, καθώς και να ξεκινούν ή να σταματούν ροές με λίγα μόνο κλικ. Με τον τρόπο αυτό, η πλατφόρμα απλοποιεί σημαντικά τη διασύνδεση με εξωτερικά συστήματα (όπως MongoDB, Kafka) και τη διαχείριση αρχείων, καθιστώντας τη χρήση της εφικτή ακόμα και από μη εξειδικευμένους χρήστες.

Υποσύστημα Asset Cataloguing

Το Asset Cataloguing είναι το κεντρικό αποθετήριο που καταγράφει και διαχειρίζεται όλα τα assets της πλατφόρμας. Πρόκειται για μία βάση δεδομένων τύπου MongoDB, στην οποία κάθε IoT συσκευή ή αισθητήρας καταχωρείται ως ένα asset με συγκεκριμένα metadata με βάση ένα κατάλληλο σχήμα. Παρέχει δυνατότητα εύκολης αναζήτησης, διαχείρισης και ενημέρωσης των στοιχείων, επιτρέποντας στους χρήστες να εντοπίζουν γρήγορα τα διαθέσιμα αποθετήρια και να ενημερώνονται για τα χαρακτηριστικά και τις δυνατότητές τους. Το αντίστοιχο μοντέλο δεδομένων που υλοποιεί το υποσύστημα παρουσιάζεται αναλυτικά στην Ενότητα 3.6.1. Για την επικοινωνία των assets με την πλατφόρμα χρησιμοποιούνται connectors, όπως ο MongoDB Connector για επικοινωνία με βάσεις δεδομένων τύπου MongoDB, ο External API Connector για σύνδεση με εξωτερικά APIs και ο Kafka Connector για διαχείριση ροών δεδομένων σε πραγματικό χρόνο.

Η χρήση της MongoDB ως asset registry προσφέρει ευελιξία στην αποθήκευση και επέκταση των μεταδεδομένων κάθε asset, ενώ η δομή των connectors διασφαλίζει ότι η πλατφόρμα μπορεί να υποστηρίξει μελλοντικά και νέους τύπους αποθετηρίων ή APIs, απλώς προσθέτοντας νέους connectors με τα κατάλληλα endpoints και παραμέτρους. Η διασύνδεση μεταξύ asset και connector διατηρεί την ανεξαρτησία και την επεκτασιμότητα του συστήματος, καθώς κάθε asset μπορεί να αλλάξει connector ή να ενημερώσει τις παραμέτρους του χωρίς να επηρεάζει τα υπόλοιπα assets ή τη συνολική λειτουργία της πλατφόρμας. Συνολικά, το Asset Cataloguing λειτουργεί ως το βασικό σημείο οργάνωσης, διασύνδεσης και διαχείρισης όλων των δεδομένων και υπηρεσιών της πλατφόρμας.

Οι connectors είναι APIs που καλούν τα κατάλληλα requests για να αντλούν ή να αποθηκεύουν δεδομένα. Αποτελούν τον τρόπο με τον οποίο η πλατφόρμα συνδέεται με τα

assets του Asset Catalog ώστε να έχει πρόσβαση στα δεδομένα. Κάθε connector είναι προκαθορισμένος και εξειδικευμένος για έναν συγκεκριμένο τύπο asset. Για παράδειγμα, ένας MongoDB connector μπορεί να συνδέεται αποκλειστικά με MongoDB βάσεις δεδομένων. Ένας Kafka connector μπορεί να συνδέεται μόνο με συστήματα Kafka για πραγματικές ροές δεδομένων.

Έτσι, το REST API του Asset Cataloguing διακρίνεται από μία βασική λειτουργία, το Discoverability, η οποία επιτρέπει την εγγραφή data assets που αποθηκεύονται στην πλατφόρμα Big Data, παρέχοντας λειτουργίες τύπου CRUD για τη διαχείριση τους. Στον παρακάτω πίνακα παρουσιάζονται αναλυτικά οι διαθέσιμες λειτουργίες:

Λειτουργία	**Περιγραφή**
HTTP [GET /cataloguing/asset	Λίστα με όλα τα καταχωρημένα assets.
HTTP [POST /cataloguing/asset	Καταχώρηση ενός νέου asset.
HTTP [GET /cataloguing/asset/id	Πληροφορίες asset με συγκεκριμένο id.
HTTP [DELETE /cataloguing/asset/id	Διαγραφή ενός asset.
HTTP [GET /cataloguing/asset/Pilot/PilotName	Λίστα assets συγκεκριμένου πιλότου.

Πίνακας 3.3: *Λειτουργίες Discoverability για data assets*

Η ενότητα Connectors παρέχει τις αντίστοιχες λειτουργίες διαχείρισης των connectors, οι οποίες περιγράφονται στον επόμενο πίνακα:

Λειτουργία	**Περιγραφή**
HTTP [GET /cataloguing/connector	Λίστα με όλα τα καταχωρημένα connectors.
HTTP [POST /cataloguing/connector	Καταχώρηση νέου connector.
HTTP [GET /cataloguing/connector/id	Πληροφορίες ενός connector.
HTTP [DELETE /cataloguing/connector/id	Διαγραφή ενός connector.
HTTP [GET /cataloguing/connector/Type	Λίστα των διαθέσιμων τύπων connector.

Πίνακας 3.4: *Λειτουργίες Discoverability για connectors*

Υποσύστημα Connector

Το υποσύστημα Connector περιλαμβάνει τις υπηρεσίες που επιτρέπουν στην πλατφόρμα να συνδεθεί με τους πιλότους και τα αποθετήριά τους. Με απλά λόγια, κάθε connector λειτουργεί ως ενδιάμεσο επίπεδο διασύνδεσης (integration layer) μεταξύ των αποθετηρίων και της πλατφόρμας, ώστε τα δεδομένα να μπορούν να ανταλλάσσονται με έναν τυποποιημένο τρόπο. Ανάλογα με τον τύπο του αποθετηρίου που χρησιμοποιεί ένας πιλότος, απαιτείται και ο αντίστοιχος connector. Το αντίστοιχο μοντέλο δεδομένων που υλοποιεί το υποσύστημα παρουσιάζεται αναλυτικά στην Ενότητα 3.6.1. Για παράδειγμα, αν ένας πιλότος αποθηκεύει δεδομένα σε μια βάση MongoDB, τότε χρησιμοποιείται ο MongoDB Connector, ο οποίος αναλαμβάνει τη σύνδεση με τέτοιου τύπου βάσεις. Αν ένας πιλότος παράγει συνεχώς δεδομένα σε πραγματικό χρόνο μέσω Kafka, τότε χρησιμοποιείται ο Kafka Connector, ο οποίος μπορεί

να συνδεθεί με τις συσκευές του πιλότου (π.χ. αισθητήρες, κάμερες) και να προωθήσει τα δεδομένα στο κατάλληλο αποθετήριο.

Παρόλο που οι connectors διαφοροποιούνται ανάλογα με το είδος των αποθετηρίων, όλοι μοιράζονται ένα κοινό σύνολο βασικών λειτουργιών, οι οποίες έχουν δηλωθεί στο υποσύστημα Asset Cataloguing. Κάθε connector περιλαμβάνει τέσσερα βασικά endpoints, τα οποία καλύπτουν την ανάκτηση και αποθήκευση δεδομένων και αρχείων. Τα endpoints αυτά ορίζονται μέσα στο Asset Cataloguing και μπορεί να έχουν διαφορετική υλοποίηση, αρκεί να καλύπτουν τις βασικές λειτουργίες που παρουσιάζονται στον παρακάτω πίνακα.

Λειτουργία	**Περιγραφή**
HTTP [GET] [retrieve]	Προσθήκη δεδομένων σε υπάρχον asset.
HTTP [GET] [file_retrieve]	Προσθήκη αρχείων σε υπάρχον asset.
HTTP [POST] [store]	Αποθήκευση δεδομένων σε νέο asset.
HTTP [POST] [file_upload]	Αποθήκευση αρχείων σε νέο asset.

Πίνακας 3.5: *Λειτουργίες Data exchange του Connector.*

Η λογική αυτή είναι ιδιαίτερα δυναμική και ενιαία, καθώς στην περίπτωση που προστεθεί ένας νέος πιλότος με διαφορετική τεχνολογία, η πλατφόρμα δεν χρειάζεται να αλλάξει. Το μόνο που απαιτείται είναι η ανάπτυξη ενός νέου Connector, ο οποίος θα γνωρίζει πώς να συνδέεται με την τεχνολογία αυτή και να εκτελεί τις βασικές λειτουργίες (retrieve, store, file_upload, file_retrieve). Μόλις ο νέος Connector καταχωρηθεί στο Asset Cataloguing, τα δεδομένα του πιλότου μπορούν να ενσωματωθούν απρόσκοπτα στην πλατφόρμα.

Υποσύστημα Flow Management API

Το υποσύστημα αυτό δημιουργεί και διαχειρίζεται τη ροή των δεδομένων. Η πλατφόρμα υποστηρίζει την αυτοματοποιημένη δρομολόγηση των δεδομένων προς τα κατάλληλα αποθετήρια βάσει των χαρακτηριστικών των δεδομένων και των απαιτήσεων του κάθε asset. Το αντίστοιχο μοντέλο δεδομένων που υλοποιεί το υποσύστημα παρουσιάζεται αναλυτικά στην Ενότητα 3.6.1. Μέσω του Data Flow module, η πλατφόρμα επιτρέπει τη δημιουργία αυτοματοποιημένων ροών δεδομένων που μπορούν να μεταφέρουν δεδομένα από ένα asset (πηγή) προς ένα destination (προορισμό). Αυτό είναι ιδιαίτερα χρήσιμο για την προώθηση δεδομένων σε επιπλέον συστήματα επεξεργασίας, αναλυτικά εργαλεία, ή άλλα αποθετήρια. Η πλατφόρμα διασφαλίζει την αξιοπιστία της μεταφοράς δεδομένων μέσω batch processing και error handling mechanisms και παρέχει λύσεις για την αποθήκευση και ανάλυση των IoT δεδομένων. Η υποστήριξη για χρονοσειρές δεδομένων είναι ιδιαίτερα σημαντική για IoT εφαρμογές, καθώς πολλοί αισθητήρες παράγουν δεδομένα σε τακτά χρονικά διαστήματα.

Το REST API του Flow Management αποσκοπεί στο Discoverability, δηλαδή επιτρέπει την εγγραφή των data flows και παρέχει λειτουργίες τύπου CRUD για τη διαχείριση τους. Στον παρακάτω πίνακα παρουσιάζονται αναλυτικά οι διαθέσιμες λειτουργίες:

Λειτουργία	Περιγραφή
HTTP [GET] /api/management/flow/new	Καταχώρηση ενός νέου data flow
HTTP [POST] /api/management/flow/new/with_files	Καταχώρηση ενός νέου file flow
HTTP [GET] /api/management/flows	Λίστα με όλα τα καταχωρημένα flows.
HTTP [POST] /api/management/flow/stop/id	Διακοπή ενός asset.
HTTP [DELETE] /api/management/flow/delete/id	Διαγραφή ενός flow.

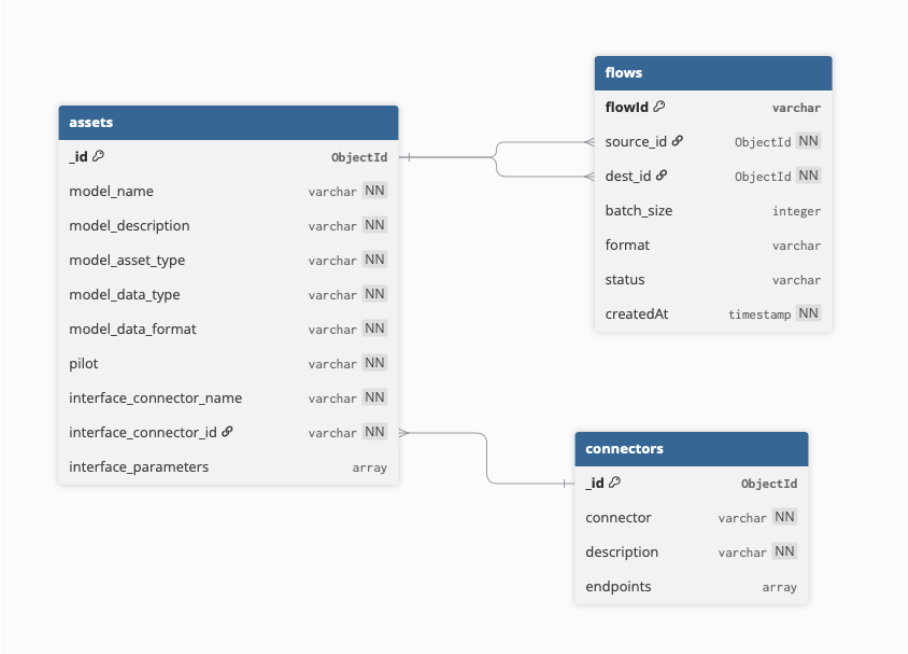
Πίνακας 3.6: *Λειτουργίες Discoverability για data flows*

3.6 Μοντέλο Δεδομένων & Μεταδεδομένων

Με βάση τα παραπάνω, το κεντρικό αποθετήριο της πλατφόρμας οργανώνεται γύρω από τρεις βασικές οντότητες. Στην ενότητα αυτή παρουσιάζουμε αναλυτικά το μοντέλο δεδομένων και τη βάση που υλοποιήθηκε ώστε να υποστηρίζει τη λειτουργικότητα της πλατφόρμας, όπως αυτή περιγράφηκε προηγουμένως. Ειδικότερα, περιγράφουμε τις οντότητες assets, connectors, flows, τα πεδία/μεταδεδομένα που τις συγκροτούν, καθώς και τις μεταξύ τους σχέσεις.

3.6.1 Μοντέλο Οντοτήτων Συσχετίσεων

Στην παρούσα ενότητα παρουσιάζεται το διάγραμμα οντοτήτων-συσχετίσεων (Entity - Relationship Diagram) το οποίο απεικονίζει τη συνολική δομή των βασικών οντοτήτων και των μεταξύ τους σχέσεων, όπως αυτές έχουν διαμορφωθεί στο πλαίσιο της πλατφόρμας.



Σχήμα 3.4: *Αναπαράσταση οντοτήτων και συσχετίσεων του μοντέλου δεδομένων της πλατφόρμας*

Σχήμα της οντότητας Asset

Η οντότητα Asset περιγράφει τις επιμέρους πηγές ή προορισμούς δεδομένων (π.χ. βάσεις δεδομένων, APIs, αισθητήρες). Κάθε εγγραφή αντιστοιχεί σε ένα αποθετήριο ή ένα σύστημα με το οποίο η πλατφόρμα μπορεί να επικοινωνήσει. Το μοντέλο δεδομένων των assets χωρίζεται σε τρία βασικά μέρη: το μοντέλο (model), τον πιλότο (pilot) και τη διασύνδεση (interface).
Τα βασικά μεταδεδομένα που περιλαμβάνει είναι:

- **_id**: Μοναδικό αναγνωριστικό του asset.
- **model_name**: Το όνομα του asset, χρησιμοποιείται για την αναγνώριση από το UI και τις ροές.
- **model_description**: Περιγραφή του asset που παρέχει πληροφορίες για τη χρήση του.
- **model_asset_type**: Κατηγορία του asset (π.χ. historical, streaming).
- **model_data_type**: Τύπος των δεδομένων που διακινεί το asset (π.χ. time series, sensorial, image, geospatial).
- **model_data_format**: Μορφή δεδομένων (πχ. JSON, CSV, File, RAW).
- **pilot**: Πιλότος στον οποίο ανήκει το asset.
- **interface_connector_name**: Όνομα του συνδεδεμένου connector.
- **interface_connector_id**: Αναγνωριστικό του connector που χρησιμοποιεί.
- **interface_parameters**: Παράμετροι σύνδεσης που διαφέρουν ανάλογα με τον τύπο του asset: για MongoDB απαιτούνται server, port, username, password, database, collection, για Kafka απαιτούνται bootstrap_servers, port, group_id, client_id, username, password, topic, ενώ για ένα εξωτερικό API συνήθως server, port, token και ό,τι άλλο καθορίζεται από το ίδιο το API.

Όνομα πεδίου	**Τύπος**	**Περιγραφή**
_id	String	Μοναδικό αναγνωριστικό του asset
model_name	String	Όνομα του asset
model_description	String	Περιγραφή του asset
model_asset_type	String	Κατηγορία asset
model_data_type	String	Τύπος δεδομένων
model_data_format	String	Μορφή δεδομένων
pilot	String	Πιλότος στον οποίο ανήκει
interface_connector_name	String	Όνομα συνδεδεμένου connector
interface_connector_id	String	Αναγνωριστικό connector
interface_parameters	Object	Παράμετροι σύνδεσης

Πίνακας 3.7: *Πεδία της οντότητας Asset*

Σχήμα της οντότητας Connector

Η οντότητα Connector περιγράφει τον μηχανισμό διασύνδεσης της πλατφόρμας με εξωτερικές πηγές και προορισμούς δεδομένων. Κάθε connector αντιστοιχεί σε έναν συγκεκριμένο τύπο αποθετηρίου και περιλαμβάνει τις απαραίτητες πληροφορίες για την επικοινωνία μαζί του. Το μοντέλο δεδομένων ενός connector περιλαμβάνει ένα id, ένα όνομα, μία περιγραφή και τα endpoints. Το τελευταίο πρόκειται για έναν πίνακα endpoints, ο οποίος περιλαμβάνει τέσσερα βασικά που πρέπει να δηλώνονται με συγκεκριμένη σειρά και να επιτρέπουν την εκτέλεση των αντίστοιχων βασικών λειτουργιών της πλατφόρμας. Κάθε endpoint διαθέτει τρία πεδία όνομα, μέθοδο και endpoint URL.
Τα βασικά μεταδεδομένα είναι:

- **_id**: Μοναδικό αναγνωριστικό connector.
- **connector**: Όνομα/τύπος connector (πχ MongoDB, Kafka, REST API).
- **description**: Περιγραφή του connector και της λειτουργίας του.
- **endpoints_name**: Λίστα με τα διαθέσιμα endpoints (retrieve, store, search, file_upload,file_retrieve).
- **endpoints_method**: Μέθοδοι HTTP (πχ `GET`, `POST`, `PUT`, `DELETE`) που υποστηρίζονται από κάθε endpoint.
- **endpoints_url**: URL κλήσης για κάθε endpoint.

Όνομα πεδίου	**Τύπος**	**Περιγραφή**
_id	String	Μοναδικό αναγνωριστικό connector
connector	String	Όνομα/τύπος σοννεστορ
description	String	Περιγραφή του σοννεστορ
endpoints_name	String	Λίστα διαθέσιμων endpoints
endpoints_method	String	Μέθοδοι HTTP για κάθε endpoint
endpoints_url	String	URLs κλήσης των endpoints

Πίνακας 3.8: *Πεδία της οντότητας Connector*

Οι connectors είναι κρίσιμο στοιχείο της πλατφόρμας, καθώς καθιστούν δυνατή την ενοποιημένη διασύνδεση με διαφορετικές τεχνολογίες χωρίς αλλαγή στον πυρήνα της αρχιτεκτονικής.

Σχήμα της οντότητας Flow

Το Flow αποτελεί μία βασική οντότητα της πλατφόρμας, καθώς περιγράφει τη ροή δεδομένων μεταξύ δύο assets — της πηγής και του προορισμού. Στην πράξη, κάθε flow καθορίζει πώς τα δεδομένα μεταφέρονται, σε ποια μορφή, και με ποια συχνότητα. Το σχήμα της οντότητας έχει σχεδιαστεί με τρόπο ώστε να υποστηρίζει τόσο ροές πραγματικού χρόνου όσο και ροές ιστορικών δεδομένων.
Η οντότητα Flow αποτυπώνει τις ακόλουθες βασικές ιδιότητες (μεταδεδομένα):

- **flowId**: Μοναδικό αναγνωριστικό της ροής δεδομένων. Δημιουργείται αυτόματα κατά την εγγραφή (`flow_$Date.now()`) και επιτρέπει την ταχεία αναζήτηση και διαχείριση ροών.
- **source_id**: Αναγνωριστικό της πηγής των δεδομένων. Αναφέρεται στο `_id` ενός asset και δηλώνει από ποιο αποθετήριο ή σύστημα προέρχονται τα δεδομένα.
- **dest_id**: Αναγνωριστικό του προορισμού των δεδομένων. Αναφέρεται επίσης σε asset και καθορίζει πού θα σταλούν τα δεδομένα.
- **batch_size**: Ορίζει τον αριθμό εγγραφών που μεταφέρονται σε κάθε παρτίδα (batch). Η προεπιλεγμένη τιμή είναι 60. Επιτρέπει τη βελτιστοποίηση της μεταφοράς δεδομένων.
- **format**: Καθορίζει τη μορφή των δεδομένων που θα διακινηθούν. Υποστηρίζονται `json`, `csv` και `files`.
- **status**: Δηλώνει την τρέχουσα κατάσταση της ροής, η οποία μπορεί να είναι `active` ή `stopped`, επιτρέποντας έτσι την ενεργοποίηση/απενεργοποίηση ροών χωρίς διαγραφή τους.
- **createdAt**: Χρονοσφραγίδα που δηλώνει τη στιγμή δημιουργίας της ροής. Χρησιμοποιείται για την παρακολούθηση και τον έλεγχο του κύκλου ζωής των ροών.

Όνομα πεδίου	**Τύπος**	**Περιγραφή**
flowId	String	Μοναδικό αναγνωριστικό της ροής
source_id	String	Αναγνωριστικό του asset-πηγή
dest_id	String	Αναγνωριστικό του asset-προορισμού
batch_size	Integer	Μέγεθος batch ανά αποστολή
format	String	Μορφή δεδομένων
status	String	Κατάσταση ροής
createdAt	Date	Χρονοσφραγίδα δημιουργίας ροής

Πίνακας 3.9: *Πεδία της οντότητας Flow*

3.6.2 Συσχετίσεις μεταξύ των Οντοτήτων

Οι οντότητες του συστήματος συνδέονται μέσω σχέσεων που επιτρέπουν συνεπή οργάνωση και ανάλυση των δεδομένων. Παρακάτω αναλύονται οι κύριες συσχετίσεις μεταξύ των οντοτήτων:

- **Asset - Connector** (πολλά–προς–ένα)

  Η οντότητα Asset σχετίζεται άμεσα με την Connector, καθώς κάθε asset χρησιμοποιεί έναν connector για τη διασύνδεση με το αντίστοιχο αποθετήριο δεδομένων (**assets.interface_connector_id - connectors._id**). Πολλά ασσετς μπορούν να μοιράζονται τον ίδιο connector.

- **Flow - Asset** (πολλά–προς–ένα)

Κάθε flow συνδέεται με δύο assets — ένα ως πηγή και ένα ως προορισμό. Η σύνδεση αυτή υλοποιεί τη δρομολόγηση των δεδομένων εντός της πλατφόρμας, επιτρέποντας την αυτοματοποίηση της μεταφοράς τους χωρίς την παρέμβαση του χρήστη. Το source_id ενός flow δείχνει στο asset-πηγή (**flows.source_id - assets._id**). Το dest_id δείχνει στο asset-προορισμό (**flows.dest_id - assets._id**).

Κεφάλαιο 4

Υλοποίηση

Στο κεφάλαιο αυτό περιγράφεται η υλοποίηση του συστήματος, με βάση την αρχιτεκτονική που περιγράφηκε στο Κεφάλαιο 3. Αρχικά παρουσιάζεται η πλατφόρμα και τα προγραμματιστικά εργαλεία που χρησιμοποιήθηκαν. Στη συνέχεια, δίνονται οι λεπτομέρειες υλοποίησης για την ανάπτυξη των επιμέρους συστημάτων και των βασικών λειτουργιών της πλατφόρμας: συλλογή, αποθήκευση και μεταφορά δεδομένων.

4.1 Λεπτομέρειες Υλοποίησης

Για την υλοποίηση της πλατφόρμας επιλέχθηκε το V-Model (βλ. 2.1.6), ώστε να εξασφαλιστεί η σαφής αντιστοίχιση μεταξύ σχεδιασμού και ελέγχων σε κάθε στάδιο ανάπτυξης. Κάθε microservice αναπτύχθηκε ανεξάρτητα και υποβλήθηκε πρώτα σε τοπικά unit tests. Η τεκμηρίωση και τα API κάθε υπηρεσίας έγιναν διαθέσιμα μέσω Swagger (βλ. 2.1.8), διευκολύνοντας την αυτοτελή επαλήθευση. Αφού όλες οι υπηρεσίες πέρασαν επιτυχώς τους αρχικούς ελέγχους, ενσωματώθηκαν στο κεντρικό container της πλατφόρμας, όπου πραγματοποιήθηκαν integration tests για τη διασφάλιση της ομαλής επικοινωνίας και συνεργασίας μεταξύ τους, όπως στη δημιουργία ροών δεδομένων. Τέλος, αναπτύχθηκε το front-end με σκοπό να προσφέρει στον χρήστη μια ολοκληρωμένη και ομαλή εμπειρία. Με αυτήν την προσέγγιση, διασφαλίστηκε ότι κάθε φάση σχεδίασης αντιστοιχεί σε μια αντίστοιχη φάση δοκιμών, σύμφωνα με τις αρχές του V-Model. Παρακάτω αναλύονται οι βασικές λειτουργίες της πλατφόρμας και ο τρόπος με τον οποίο υλοποιούνται βήμα βήμα μέσα από την αλληλεπίδραση πολλαπλών υπηρεσιών και την ορθή επικοινωνία μεταξύ τους.

4.2 Υλοποίηση Connectors

Στο προηγούμενο κεφάλαιο, στο πλαίσιο της αρχιτεκτονικής, περιγράφηκε το υποσύστημα των Connectors ως ο μηχανισμός που επιτρέπει στη πλατφόρμα να συνδέεται με ετερογενείς πηγές δεδομένων. Στην υλοποίηση, η θεωρητική αυτή λογική αποτυπώνεται με την ανάπτυξη συγκεκριμένων τύπων Connectors, οι οποίοι ακολουθούν το ίδιο ενιαίο μοντέλο λειτουργιών που ορίστηκε μέσω του Asset Cataloguing. Στο πλαίσιο του έργου υλοποιήθηκαν δύο βασικοί τύποι Connector: (α) ο MongoDB Connector, που επιτρέπει την ανάγνωση και αποθήκευση δεδομένων σε βάσεις MongoDB, και (β) ο External API Connector, που παρέχει δυναμική διασύνδεση με εξωτερικά συστήματα μέσω παραμέτρων που δηλώνονται

στο Asset Cataloguing.

Mongodb Connector

Ο MongoDB Connector που υλοποιήθηκε είναι μια ευέλικτη υπηρεσία που επιτρέπει την ανάγνωση και εγγραφή δεδομένων σε οποιαδήποτε βάση MongoDB, αρκεί να δοθούν τα σωστά στοιχεία σύνδεσης. Μέσα από παραμετροποιήσιμα assets, ο connector λαμβάνει τις απαραίτητες πληροφορίες (όπως διεύθυνση server, όνομα βάσης, συλλογή, username και password) και μπορεί να συνδεθεί δυναμικά σε διαφορετικές βάσεις, τοπικές ή απομακρυσμένες. Η υπηρεσία αυτή προσφέρει endpoints για ανάγνωση δεδομένων από συλλογές MongoDB, εγγραφή/εισαγωγή δεδομένων είτε απευθείας είτε μέσω αρχείων (JSON ή CSV), και τέλος, διαχείριση μεγάλων datasets με batch insert και streaming, ώστε η διαδικασία να παραμένει αποδοτική χωρίς επιβάρυνση της μνήμης.

Στη διαδικασία του upload, ο MongoDB Connector ελέγχει πρώτα τον τύπο του αρχείου που ανέβηκε στην πλατφόρμα. Αν το αρχείο είναι σε μορφή JSON ή CSV, χρησιμοποιεί έναν cursor για να διαβάσει τα δεδομένα σε batches και τα ανεβάζει σταδιακά στη βάση μέσω της μεθόδου insertMany(), επιτυγχάνοντας αποδοτική μαζική εισαγωγή εγγράφων. Σε περίπτωση retrieve, ο connector επιστρέφει το περιεχόμενο της συλλογής σε μορφή αρχείου JSON. Αν το αρχείο που γίνεται upload δεν είναι σε μορφή JSON ή CSV, τότε ο connector αποθηκεύει κάθε αρχείο προσεκτικά στη MongoDB σε binary μορφή, ώστε κατά την ανάκτηση να μπορεί να μετατρέψει τα έγγραφα ξανά στην αρχική τους μορφή. Για παράδειγμα, αν αρχικά το αρχείο ήταν .txt, θα επιστραφεί ως .txt. Σε περίπτωση πολλαπλών αρχείων, αυτά συγκεντρώνονται και επιστρέφονται συμπιεσμένα σε μορφή .zip.

Ο MongoDB Connector που υλοποιήθηκε είναι πλήρως παραμετροποιήσιμος και δεν περιορίζεται σε μία μόνο βάση ή συλλογή. Μπορεί να χρησιμοποιηθεί σε διάφορα σενάρια, όπως data migration, ETL pipelines, real-time data flows ή απλή διασύνδεση εφαρμογών με βάσεις MongoDB. Αποτελεί ένα γενικό εργαλείο διασύνδεσης που προσαρμόζεται εύκολα σε κάθε ανάγκη, χωρίς να απαιτείται αλλαγή στον κώδικα για κάθε νέα βάση ή συλλογή. Με αυτόν τον τρόπο, ο MongoDB Connector διαχειρίζεται αποτελεσματικά δεδομένα ποικίλων μορφών, διασφαλίζοντας τη διατήρηση της αρχικής τους δομής και μορφής κατά την αποθήκευση και ανάκτηση.

Το REST API του MongoDB Connector αποσκοπεί κυρίως στο Data exchange. Αφορά την ανταλλαγή δεδομένων μεταξύ της πλατφόρμας Big Data και άλλων υποσυστημάτων βάσεων ονγοΔΒ. Οι σχετικές λειτουργίες παρουσιάζονται στον ακόλουθο πίνακα:

Λειτουργία	**Περιγραφή**
HTTP [GET] /api/data/retrieve	Ανάκτηση δεδομένων από υπάρχον asset.
HTTP [GET] /api/file/retrieve	Ανάκτηση αρχείων από υπάρχον asset.
HTTP [POST] /api/data/upload	Αποθήκευση δεδομένων σε νέο asset.
HTTP [POST] /api/file/upload	Αποθήκευση αρχείων σε νέο asset.

Πίνακας 4.1: *Λειτουργίες Data exchange του MongoDB Connector*

External API Connector

Στο πλαίσιο της πλατφόρμας, υλοποιήθηκε επίσης ένας External API Connector, ο οποίος επιτρέπει τη δυναμική σύνδεση με εξωτερικά συστήματα μέσω παραμέτρων που δηλώνονται στο Asset Cataloguing. Με αφορμή έναν πιλότο του έργου, σχεδιάτηκε ένας Connector που συνδέει την πλατφόρμα με την υπηρεσία Jotne Connect[1]. Η Jotne Connect είναι μια εξειδικευμένη λύση για την ασφαλή ανταλλαγή και διαχείριση τεχνικών δεδομένων και αρχείων, όπως σχέδια CAD[2], BOMs[3] και μοντέλα προϊόντων. Χρησιμοποιείται κυρίως σε βιομηχανίες όπως η άμυνα, η αεροναυπηγική και η ενέργεια, και προσφέρει λειτουργίες για έλεγχο ποιότητας, μακροχρόνια αρχειοθέτηση και τυποποιημένη ανταλλαγή δεδομένων μεταξύ συστημάτων.

Μέσω του Connector, η πλατφόρμα μπορεί να συνδέεται με ασφάλεια στο Jotne Connect API χρησιμοποιώντας authentication tokens, να στέλνει τεχνικά αρχεία (π.χ. 3Δ models, CAD files), να ανακτά δεδομένα ή αρχεία από το αποθετήριο και να προσαρμόζεται δυναμικά σε κάθε διαφορετικό Jotne instance χωρίς αλλαγή στον κώδικα, απλώς τροποποιώντας τις παραμέτρους στο asset.

Η υλοποίηση αυτή αποδεικνύει ότι η πλατφόρμα υποστηρίζει έναν ευέλικτο και επεκτάσιμο μηχανισμό διασύνδεσης με οποιοδήποτε εξωτερικό API. Ο μηχανισμός βασίζεται στην παραμετροποίηση μέσω του Asset Cataloguing, επιτρέποντας την εύκολη προσθήκη νέων εξωτερικών συστημάτων χωρίς να τροποποιείται ο πυρήνας του κώδικα. Με αυτόν τον τρόπο, η πλατφόρμα αποκτά μεγάλη προσαρμοστικότητα και μπορεί να ενσωματωθεί σε σύνθετα περιβάλλοντα όπου απαιτείται διασύνδεση με πολλαπλές πηγές τεχνικών ή επιχειρησιακών δεδομένων.

4.3 Υλοποίηση Βασικών Λειτουργιών της πλατφόρμας: Data Upload, Data Retrieve, Flows

Στην ενότητα αυτή παρουσιάζεται η υλοποίηση των βασικών λειτουργιών της πλατφόρμας, οι οποίες περιγράφηκαν θεωρητικά στο Κεφάλαιο 3 και οργανώνονται γύρω από την αποθήκευση, την ανάκτηση και τη ροή δεδομένων. Με την υλοποίηση των δύο τύπων Connectors, που αναλύθηκε στην προηγούμενη υποενότητα, δίνεται η δυνατότητα να παρουσιαστούν σενάρια λειτουργίας για καθέναν από αυτούς. Με αυτόν τον τρόπο, η αρχιτεκτονική των βασικών λειτουργιών καθίσταται πιο κατανοητή, καθώς αποτυπώνεται βήμα προς βήμα ο τρόπος με τον οποίο οι λειτουργίες Data Upload, Data Retrieve και Flows υλοποιούνται στην πράξη, αξιοποιώντας τη λογική των Connectors.

Data Upload - Asset Cataloguing, Mongodb Connector, External API Connector

Όταν ο χρήστης επιθυμεί να ανεβάσει δεδομένα (data upload) σε ένα συγκεκριμένο asset μέσω του UI της πλατφόρμας, ακολουθείται η παρακάτω διαδικασία. Ο χρήστης επιλέγει το asset και το αρχείο που θέλει να ανεβάσει (π.χ. JSON ή CSV) μέσω του frontend interface.

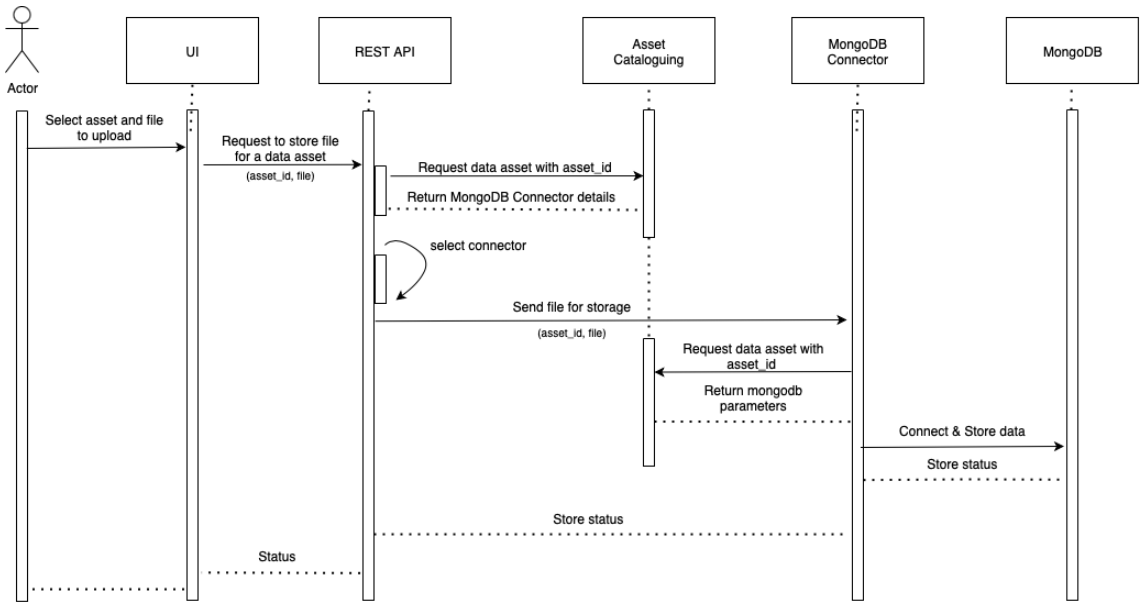
[1]Jotne Connect: https://jotneconnect.com
[2]CAD: https://www.autodesk.com/solutions/cad-design
[3]BOMs: https://www.solidworks.com/solution/what-is-bill-of-materials-bom-management

Το frontend συλλέγει το asset_id και το αρχείο, και με χρήση της βιβλιοθήκης axios αποστέλλει αίτημα POST στο backend API στο endpoint /data-upload/:asset_id, μεταφέροντας το αρχείο ως multipart/form-data.

Το backend ξεκινά την ταυτοποίηση του asset επικοινωνώντας με το Asset Cataloguing API στο endpoint /asset/$asset_id και αντλεί τον αντίστοιχο connector_id. Έπειτα ζητά από το Asset Cataloguing API πληροφορίες για τον connector μέσω endpoint /connector/$connector_id, ώστε να προσδιοριστεί ποιο είναι το κατάλληλο endpoint για το ανέβασμα των δεδομένων (π.χ. MongoDB, External API κλπ). Συνδυάζοντας τα δεδομένα του asset και του connector, κατασκευάζεται το πλήρες URL (url_data_upload) με όλες τις απαραίτητες παραμέτρους (όνομα βάσης database, συλλογή collection, credentials κά). Το backend στέλνει το αίτημα στον connector (πχ προς /api/data/upload) με axios, και ο connector με τη σειρά του επικοινωνεί ξανά με το Asset Cataloguing API για να αντλήσει τις παραμέτρους σύνδεσης στη βάση δεδομένων (host, port, database, credentials).

Για σύνδεση με MongoDB χρησιμοποιείται η βιβλιοθήκη mongoose, από την οποία κατασκευάζεται το connection string (mongoURI) με τα απαραίτητα στοιχεία, συμπεριλαμβανομένου του authSource για authentication. Αφού πραγματοποιηθεί επιτυχής σύνδεση, ο connector διαβάζει το περιεχόμενο του αρχείου και το εισάγει στο αντίστοιχο collection σύμφωνα με το schema του asset. Μετά την εισαγωγή, επιστρέφεται μήνυμα επιτυχίας ή αποτυχίας στο backend του Big Data Platform με κατάλληλο HTTP status code, το οποίο προωθεί το αποτέλεσμα στο frontend. Ο χρήστης ενημερώνεται με σχετικό μήνυμα (π.χ. «Τα δεδομένα ανέβηκαν επιτυχώς» ή «Προέκυψε σφάλμα κατά το ανέβασμα»).

Παρακάτω παρουσιάζεται σε UML Sequence Diagram η παραπάνω διαδικασία για αποθήκευση δεδομένων σε MongoDB με χρήση του MongoDB connector:



Σχήμα 4.1: *UML Sequence Diagram για Data Upload σε βάση τύπου MongoDB.*

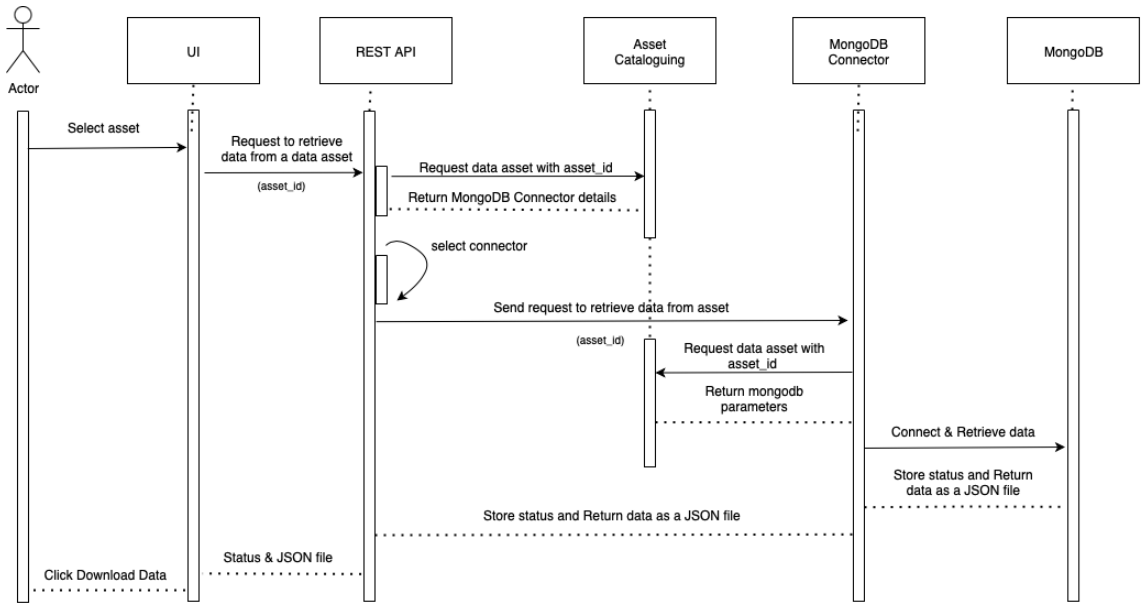
Το Σχήμα 4.1 απεικονίζει ένα UML Sequence Diagram για data upload σε βάση τύπου MongoDB.

Data Retrieve - Asset Cataloguing, Mongodb Connector, External API Connector

Όταν ο χρήστης επιθυμεί να ανακτήσει δεδομένα από ένα συγκεκριμένο asset, ακολουθείται η εξής διαδικασία. Το frontend συλλέγει το asset_id και καλεί το Big Data Platform API μέσω του endpoint /data-retrieve/:asset_id, συνήθως με AJAX/axios για ασύγχρονη επικοινωνία. Το backend ζητά λεπτομέρειες για το asset από το Asset Cataloguing API στο endpoint /asset/$asset_id, και παίρνει μεταξύ άλλων τον connector_id. Στη συνέχεια ζητά το αντίστοιχο connector στο endpoint /connector/$connector_id για να προσδιοριστεί το κατάλληλο endpoint ανάκτησης δεδομένων.

Συνδυάζοντας τα στοιχεία του asset και του connector, το backend κατασκευάζει το πλήρες URL (url_data_retrieve) με όλες τις απαιτούμενες παραμέτρους (όνομα βάσης database, συλλογή collection, credentials κ.λπ.) και στέλνει αίτημα στο endpoint του connector (π.χ. /api/data/retrieve) με axios, παρέχοντας το asset_id και τα σχετικά δεδομένα. Ο connector επικοινωνεί ξανά με το Asset Cataloguing API για να αντλήσει τις παραμέτρους σύνδεσης στη βάση και, αν πρόκειται για MongoDB, χρησιμοποιεί mongoose για να φτιάξει το mongoURI με authSource. Μετά την επιτυχή σύνδεση εκτελεί το κατάλληλο query στο collection και επιστρέφει τα δεδομένα, στη μορφή που είχαν αρχικά σταλθεί για αποθήκευση, με HTTP 200 OK στο backend, το οποίο τα προωθεί στο frontend. Εκεί ο χρήστης βλέπει κουμπί «Download Data» και μπορεί να κατεβάσει τα δεδομένα σε μορφή που τα είχε αποθηκεύσει (JSON, csv, .txt, .parquet)

Παρακάτω παρουσιάζεται σε UML Sequence Diagram η παραπάνω διαδικασία για ανάκτηση δεδομένων από MongoDB με χρήση του MongoDB connector:



Σχήμα 4.2: *UML Sequence Diagram για Data Retrieve σε βάση τύπου MongoDB.*

Το Σχήμα 4.2 απεικονίζει ένα UML Sequence Diagram για Data Retrieve σε βάση τύπου MongoDB.

Συνοψίζοντας, οι διαδικασίες data upload και Data Retrieve είναι πλήρως αυτοματοποιημένες και διασυνδεδεμένες, με το frontend να λειτουργεί ως αφετηρία, το backend να

διαχειρίζεται τη λογική και τις διασυνδέσεις, και τους connectors να αναλαμβάνουν την πραγματική ανάκτηση και αποθήκευση των δεδομένων από τις βάσεις ή τα συστήματα. Όλα τα βήματα γίνονται με ασφαλή και δομημένο τρόπο, με κατάλληλο error handling και χρήση σύγχρονων βιβλιοθηκών για HTTP επικοινωνία και πρόσβαση σε δεδομένα. Ο χρήστης απολαμβάνει απλό interface με άμεσα διαθέσιμα αποτελέσματα.

Flows - Flows API Management

Δημιουργία Flow (Create Flow): Ο χρήστης, μέσω του frontend, επιλέγει τα απαραίτητα στοιχεία για τη δημιουργία ενός νέου flow: το asset προέλευσης (source asset), το asset προορισμού (destination asset), το format των δεδομένων (π.χ. JSON/CSV ή Files) και το μέγεθος batch (batch size), αν απαιτείται. Το frontend συλλέγει τις παραμέτρους και αποστέλλει αίτημα POST στο backend API της πλατφόρμας, στο κατάλληλο endpoint για δημιουργία flow, μεταφέροντας όλες τις σχετικές πληροφορίες. Το backend λαμβάνει το αίτημα, επικυρώνει τις παραμέτρους (π.χ. ότι τα δύο assets είναι διαφορετικά, ότι το batch size είναι έγκυρο) και αντλεί επιπλέον πληροφορίες για τα assets και τους connectors μέσω του Asset Cataloguing API. Το backend δημιουργεί μια νέα εγγραφή flow στη βάση δεδομένων (συνήθως MongoDB), αποθηκεύοντας όλες τις σχετικές πληροφορίες (source, destination, format, batch size, timestamps, status κ.λπ.). Εάν απαιτείται, ξεκινάει και την αντίστοιχη διαδικασία μεταφοράς δεδομένων μεταξύ των assets, χρησιμοποιώντας τις κατάλληλες βιβλιοθήκες και connectors. Το backend επιστρέφει μήνυμα επιτυχίας ή αποτυχίας στο frontend, το οποίο ενημερώνει τον χρήστη και ανανεώνει τη λίστα των ενεργών flows.

Διακοπή Flow (Stop Flow): Ο χρήστης επιλέγει ένα ενεργό flow από το UI και πατάει το κουμπί "Stop". Το frontend αποστέλλει αίτημα POST στο backend API, στο endpoint διακοπής flow, παρέχοντας το μοναδικό αναγνωριστικό του flow (flowId). Το backend εντοπίζει το συγκεκριμένο flow στη βάση δεδομένων και αλλάζει την κατάστασή του (status) σε "inactive" ή "stopped". Εάν το flow εκτελείται ως background process ή job, το backend φροντίζει να τερματίσει τη σχετική διεργασία, χρησιμοποιώντας κατάλληλες βιβλιοθήκες διαχείρισης jobs/processes. Το backend επιστρέφει μήνυμα επιτυχίας ή αποτυχίας στο frontend, το οποίο ενημερώνει τον χρήστη και ανανεώνει τη λίστα των stopped flows.

Διαγραφή Flow (Delete Flow): Ο χρήστης επιλέγει ένα flow (ενεργό ή ανενεργό) από το UI και πατάει το κουμπί "Delete". Το frontend αποστέλλει αίτημα DELETE στο backend API, στο endpoint διαγραφής flow, παρέχοντας το μοναδικό αναγνωριστικό του flow (flowId). Το backend εντοπίζει το συγκεκριμένο flow στη βάση δεδομένων και το διαγράφει οριστικά. Εάν το flow είναι ενεργό, φροντίζει πρώτα να το διακόψει πριν το διαγράψει, ώστε να μην παραμείνουν ορφανές διεργασίες. Το backend επιστρέφει μήνυμα επιτυχίας ή αποτυχίας στο frontend, το οποίο ενημερώνει τον χρήστη και ανανεώνει τη λίστα flows.

Συνοψίζοντας, η διαχείριση των flows στο Big Data Platform είναι μια πλήρως αυτοματοποιημένη διαδικασία που προσφέρει στον χρήστη τη δυνατότητα να δημιουργεί, να διακόπτει και να διαγράφει flows με απλό και φιλικό τρόπο. Το frontend λειτουργεί ως διεπαφή, το backend διαχειρίζεται όλη τη λογική, τις επικοινωνίες και τις εγγραφές στη βάση, ενώ οι connectors αναλαμβάνουν την πραγματική μεταφορά των δεδομένων. Όλα τα βήματα γίνονται με χρήση σύγχρονων βιβλιοθηκών και τεχνολογιών, με κατάλληλο error handling και

ενημέρωση του χρήστη για κάθε ενέργεια.

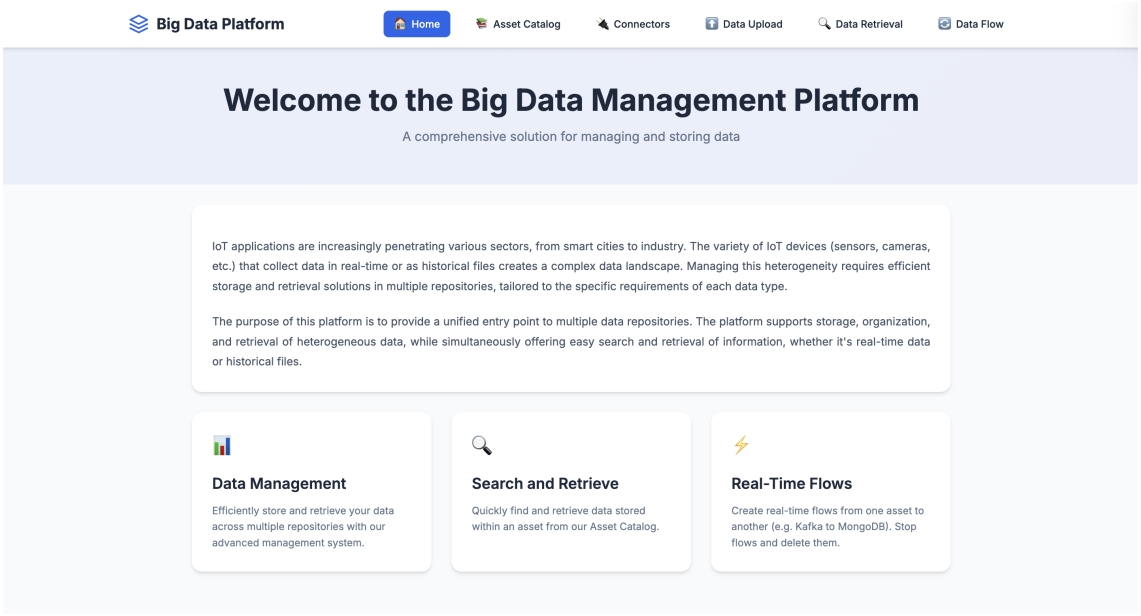
Πλευρά	**Τεχνολογίες / Βιβλιοθήκες**
Frontend	JavaScript (axios για HTTP requests), HTML/CSS (Bootstrap για UI), EJS (για dynamic rendering)
Backend (APIs)	Node.js/Express (για τα endpoints), axios (για επικοινωνία με external/internal APIs), mongoose (για διαχείριση της MongoDB), kafka-node (για σύνδεση με Kafka)

Πίνακας 4.2: *Βασικές τεχνολογίες και βιβλιοθήκες που χρησιμοποιούνται στην πλατφόρμα*

4.4 Υλοποίηση Frontend (UI)

Η διεπαφή του χρήστη αποτελεί το τελικό στάδιο του συστήματος και είναι υπεύθυνη για την παρουσίαση της πλατφόρμας με τρόπο κατανοητό και εύχρηστο. Μέσα από μία σύγχρονη και φιλική διεπαφή, οι χρήστες μπορούν να περιηγηθούν στα assets, να αναζητήσουν τον κατάλληλο connector και να στείλουν, απορροφήσουν ή να μεταφέρουν δεδομένα που θέλουν.

1. Αρχική Σελίδα (Home / Dashboard): Εμφανίζει το λογότυπο και το όνομα της πλατφόρμας, ένα μενού πλοήγησης για πρόσβαση στις υπόλοιπες σελίδες (Assets, Connectors, Data Flows, Data Retrieve, Data Upload) και μία σύντομη περιγραφή των λειτουργιών της πλατφόρμας.

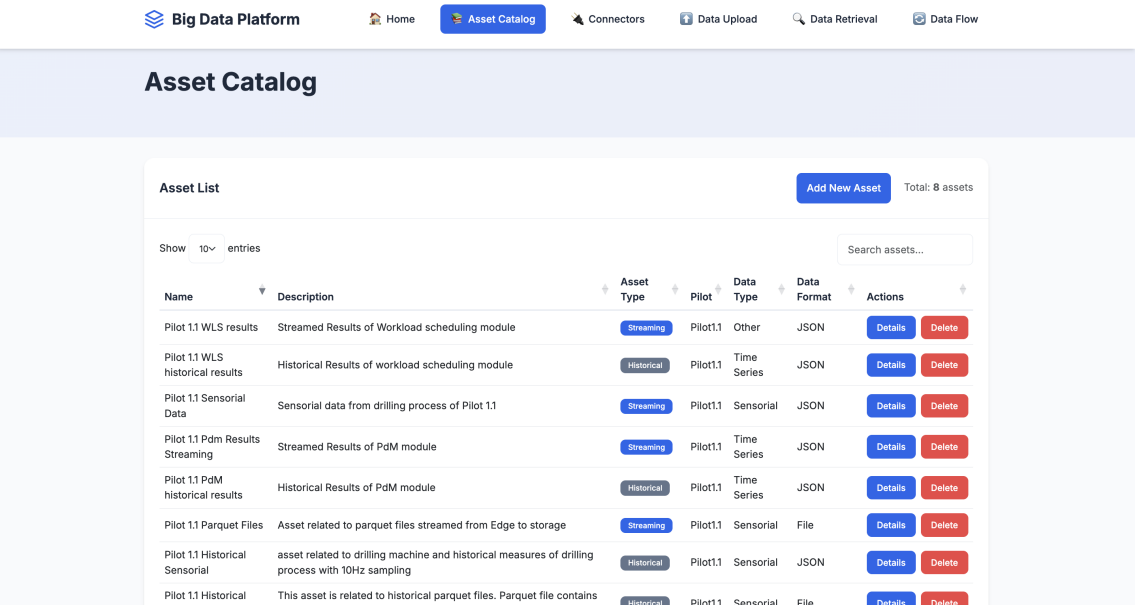


Εικόνα 4.1: *Αρχική Σελίδα*

Η εικόνα 4.1 απεικονίζει την Αρχική Σελίδα της πλατφόρμας.

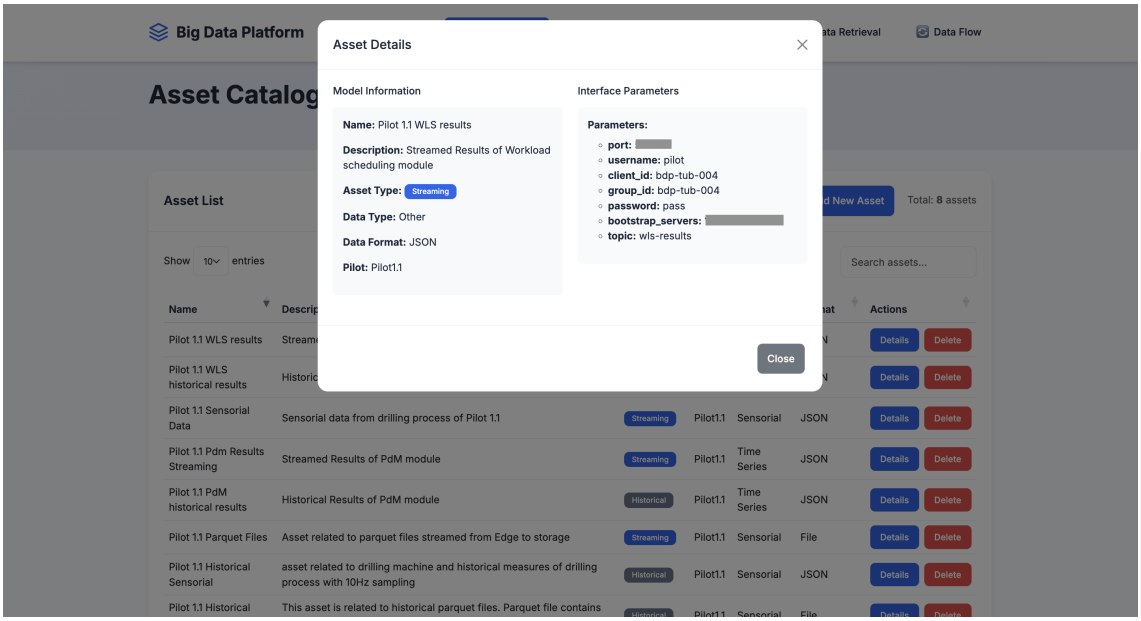
2. Σελίδα Διαχείρισης Assets: Στη σελίδα αυτή παρουσιάζεται ένας πίνακας με όλα τα assets που έχουν καταχωρηθεί στο σύστημα. Για κάθε asset φαίνεται το description,

asset type, data type, data format και pilot. Υπάρχει ένα κουμπί για διαγραφή ενός asset, ένα κουμπί για ανάγνωση περεταίρω χαρακτηριστικών του asset, ένα κουμπί για προσθήκη νέου asset και ένα πεδίο αναζήτησης για εντοπισμό ενός συγκεκριμένου asset.



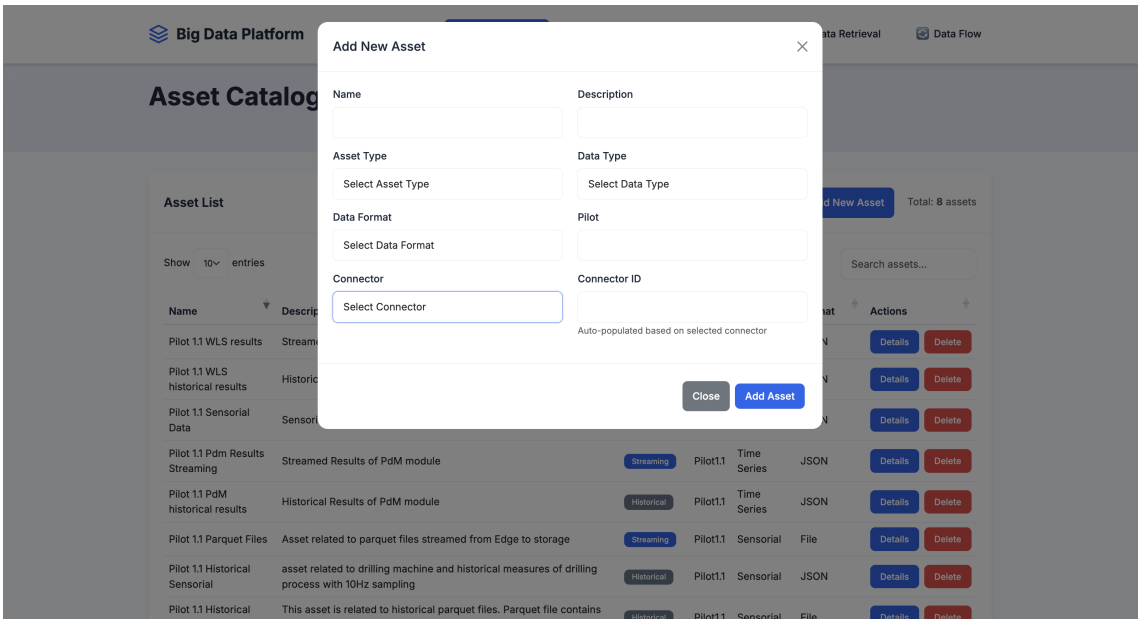
Εικόνα 4.2: *Σελίδα Διαχείρισης Assets*

Η εικόνα 4.2 απεικονίζει την σελίδα Διαχείρισης των Assets.



Εικόνα 4.3: *Σελίδα διαχείρισης Assets με προβολή των Asset Details.*

Η εικόνα 4.3 παρουσιάζει τα στοιχεία ενός Asset όπως αυτά ορίζονται στο μοντέλο του στην Ενότητα 3.6.1. Εμφανίζονται τα βασικά πεδία που περιγράφουν το αποθετήριο, όπως description, asset type, data type, data format, pilot, καθώς και οι παράμετροι σύνδεσης που εξαρτώνται από τον επιλεγμένο connector.



Εικόνα 4.4: *Σελίδα διαχείρισης Assets με φόρμα προσθήκης νέου Asset.*

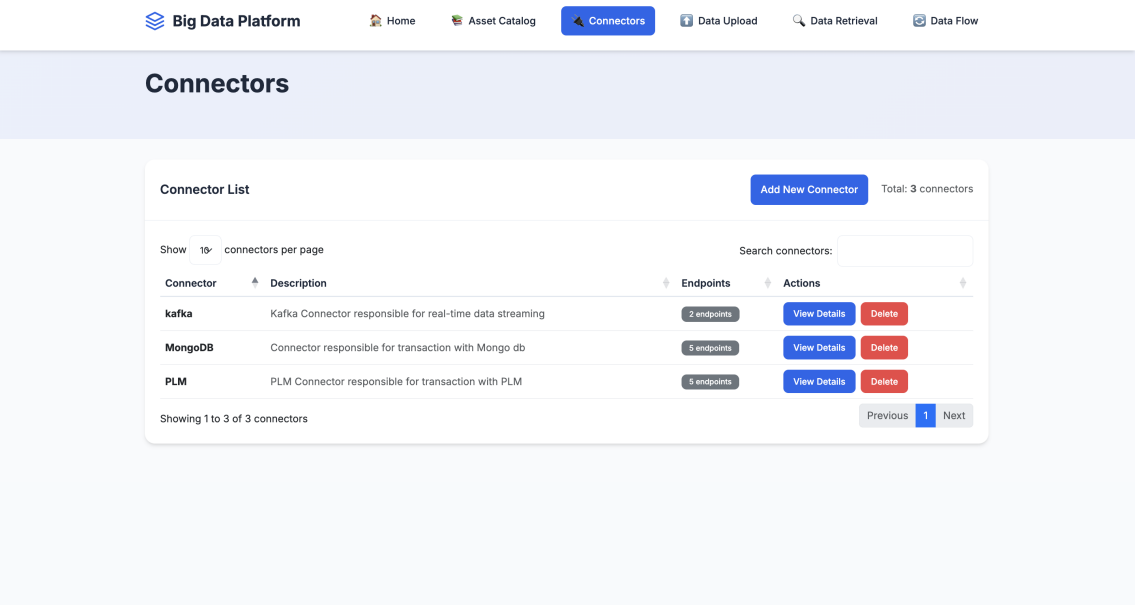
Κατά την προσθήκη ενός νέου Asset, ο χρήστης επιλέγει τον τύπο connector, ο οποίος καθορίζει και τις απαιτούμενες παραμέτρους σύνδεσης. Για παράδειγμα:

- Αν επιλεγεί **MongoDB connector**, απαιτούνται παράμετροι όπως server, port, mongoURI, database, collection, search.

- Αν επιλεγεί **Kafka connector**, απαιτούνται παράμετροι όπως bootstrap_servers, port, group_id, client_id, username, password, topic.

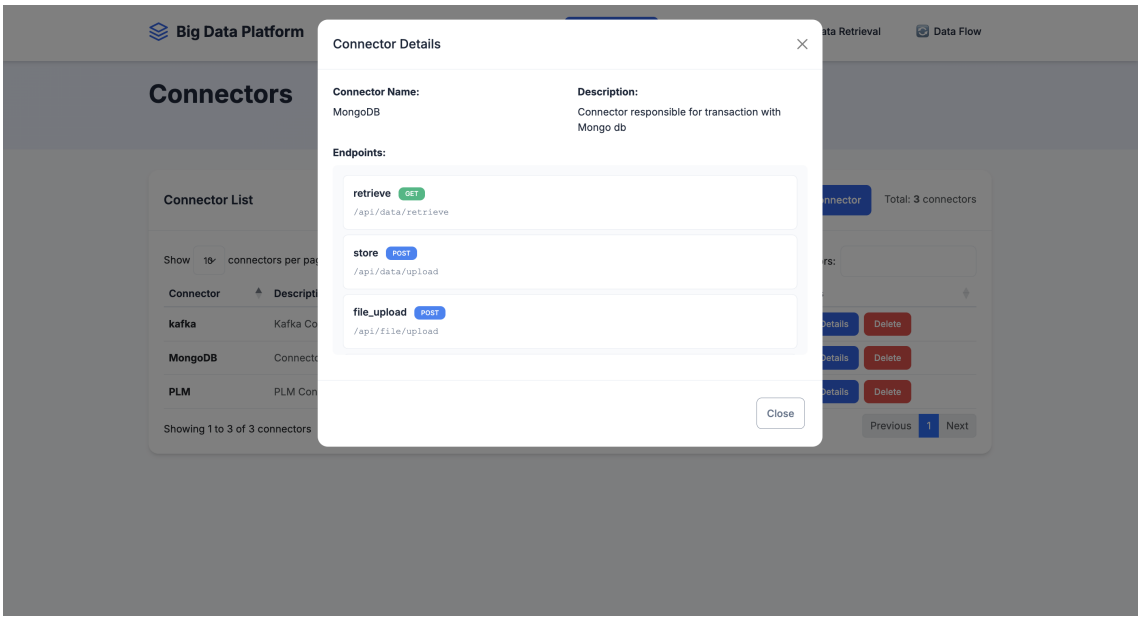
Με τον τρόπο αυτό, η ίδια φόρμα διατηρείται ενιαία και οι απαιτήσεις προσαρμόζονται δυναμικά ανάλογα με τον επιλεγμένο connector. Αυτό απλοποιεί σημαντικά τη διαχείριση ετερογενών αποθετηρίων και επιτρέπει εύκολη επέκταση για μελλοντικούς τύπους connectors.

2. Σελίδα Διαχείρισης Connectors: Εμφανίζει έναν πίνακα με όλους τους Connectors που έχουν καταχωρηθεί στο σύστημα. Για κάθε Connector φαίνεται το description, type και endpoints. Υπάρχει κουμπί για διαγραφή ενός Connector, κουμπί για ανάγνωση περαιτέρω χαρακτηριστικών του Connector, κουμπί για προσθήκη νέου Connector και πεδίο αναζήτησης για ένα συγκεκριμένο Connector.



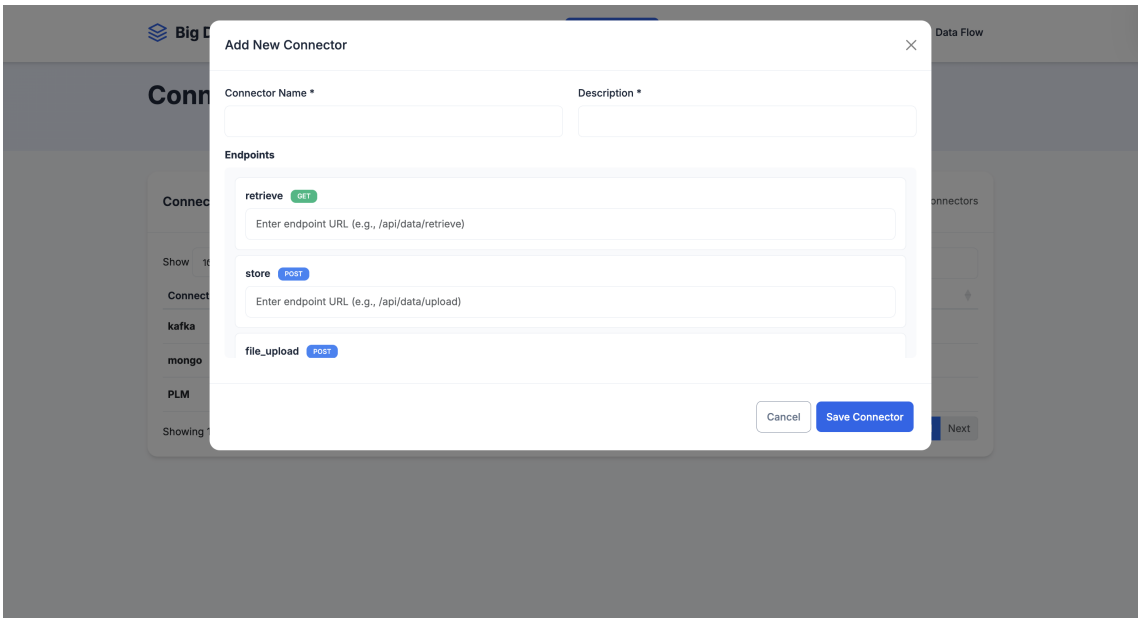
Εικόνα 4.5: *Σελίδα Διαχείρισης Connectors*

Η εικόνα 4.5 απεικονίζει Σελίδα Διαχείρισης Connectors.



Εικόνα 4.6: *Σελίδα διαχείρισης Connectors με προβολή των Connector Details.*

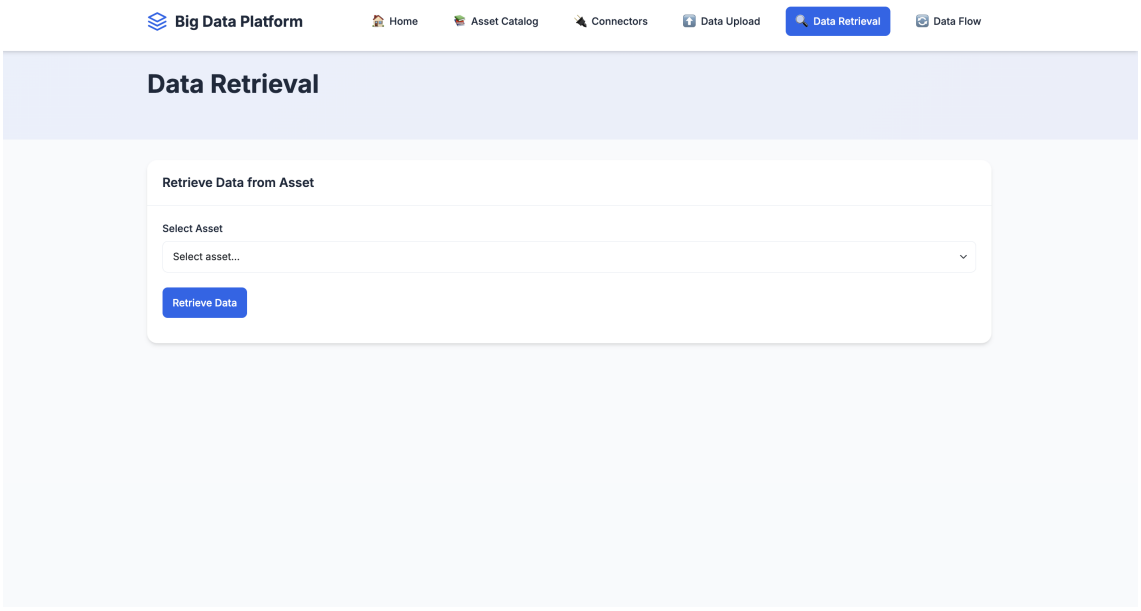
Η εικόνα 4.6 παρουσιάζει τα στοιχεία ενός Connector, όπως αυτά καταγράφονται στο σύστημα με βάση το μοντέλο δεδομένων στην Ενότητα 3.6.1. Εμφανίζονται το name, το description και τα διαθέσιμα endpoints: (α) GET [retrieve], (β) POST [store], (γ) POST [file_upload]. Στο συγκεκριμένο παράδειγμα απεικονίζεται ένας MongoDB Connector.



Εικόνα 4.7: *Σελίδα Διαχείρισης Connectors με άνοιγμα προσθήκης ενός νέου Connector.*

Η εικόνα 4.7 απεικονίζει την αρχιτεκτονική τη προσθήκη ενός νέου Connector.

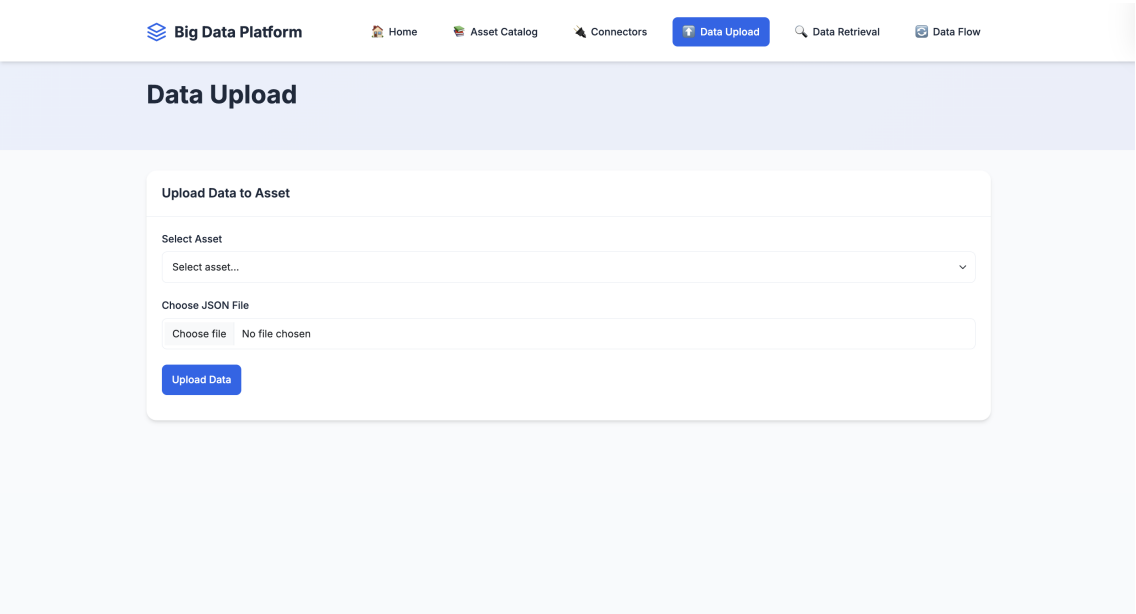
3. Σελίδα Data Retrieve: Εμφανίζει ένα Dropdown επιλογής source asset, ένα κουμπί Start Data Retrieve και τέλος αφού παρθούν τα δεδομένα ένα κουμπί Download Data και μηνύματα επιτυχίας/σφάλματος.



Εικόνα 4.8: *Σελίδα Data Retrieve*

Η εικόνα 4.8 απεικονίζει τη σελίδα Data Retrieve.

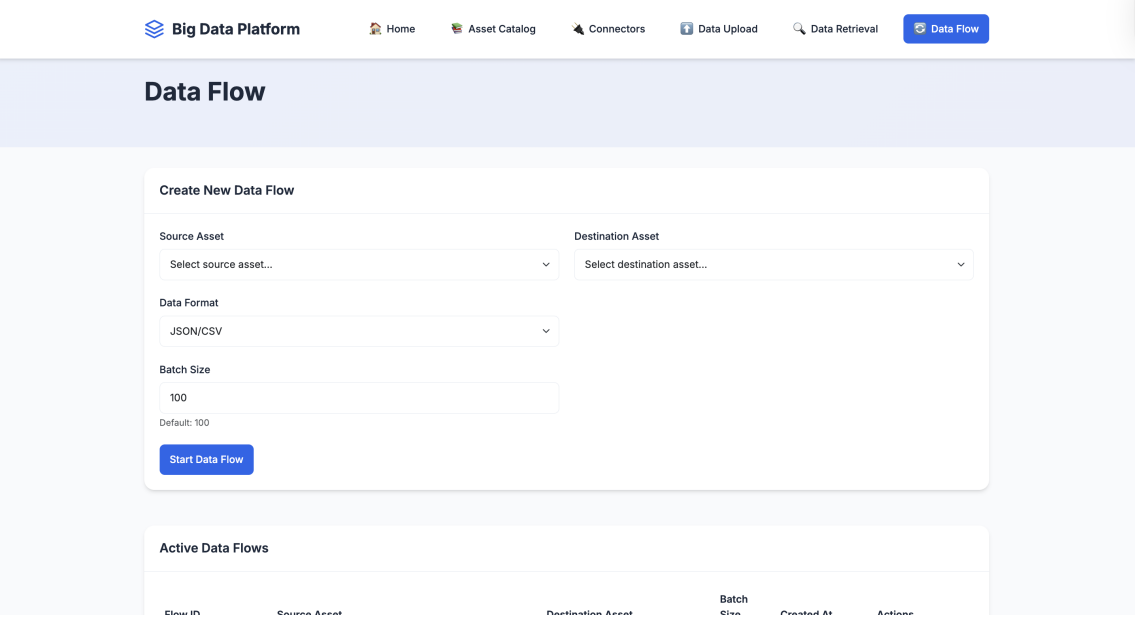
4. Σελίδα Data Upload: Εμφανίζει ένα Dropdown επιλογής destination asset, ένα πεδίο επιλογής αρχείου (file picker), ένα κουμπί Upload Data, μηνύματα επιτυχίας/σφάλματος και πληροφορίες για υποστηριζόμενα formats (π.χ. JSON, CSV).



Εικόνα 4.9: *Σελίδα Data Upload*

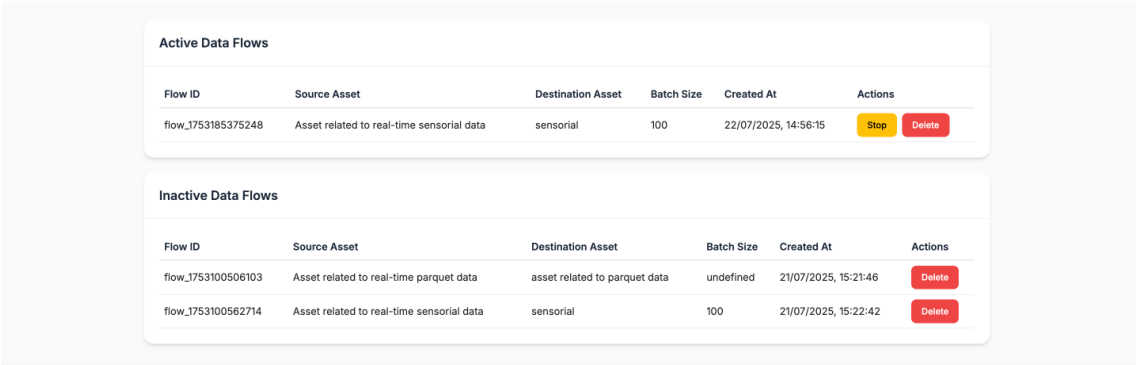
Η εικόνα 4.9 απεικονίζει τη σελίδα Data Upload.

5. Σελίδα Data Flows: Παρέχει φόρμα δημιουργίας νέου flow, με dropdowns επιλογής source/destination asset, format και batch size. Εμφανίζει επίσης δύο πίνακες: έναν με τα ενεργά flows, όπου παρουσιάζονται το Flow ID, το source, το destination, το batch size, το created at και διαθέσιμα κουμπιά Stop/Delete, και έναν δεύτερο με τα ανενεργά flows, όπου υπάρχει κουμπί Delete.



Εικόνα 4.10: *Σελίδα διαχείρισης Data Flows με φόρμα δημιουργίας νέου flow.*

Η εικόνα 4.10 παρουσιάζει τη σελίδα Data Flows, όπου ο χρήστης μπορεί να δημιουργήσει νέα flows και να παρακολουθήσει τα υπάρχοντα.



Εικόνα 4.11: *Πίνακας διαχείρισης Data Flows με ενεργά και ανενεργά flows.*

Η εικόνα 4.11 παρουσιάζει την προβολή των ενεργών και ανενεργών flows. Για τα ενεργά flows διατίθενται ενέργειες Stop/Delete, ενώ για τα ανενεργά μόνο Delete.

4.5 Deployment - Deployment diagram

Η ανάπτυξη (Deployment) της πλατφόρμας πραγματοποιήθηκε με χρήση του Docker, το οποίο πακετάρει κάθε υπηρεσία σε ένα αυτόνομο container μαζί με όλες τις απαραίτητες εξαρτήσεις της. Η αρχιτεκτονική ακολουθεί την προσέγγιση των microservices, γεγονός που σημαίνει ότι κάθε λειτουργικό τμήμα — Frontend, Management API, Data Flow API, Asset Cataloguing, Connectors — καθώς και οι βάσεις δεδομένων τους (MongoDB), εκτελούνται σε ξεχωριστά, απομονωμένα περιβάλλοντα. Αυτή η δομή προσφέρει ευελιξία, εύκολη συντήρηση και τη δυνατότητα ανεξάρτητης ανάπτυξης ή αναβάθμισης κάθε υπηρεσίας.

Ο συντονισμός όλων των containers γίνεται μέσω Docker Compose και ενός αρχείου διαμόρφωσης σε μορφή YAML, όπου ορίζονται όλες οι υπηρεσίες (π.χ. Kafka για ροές δεδομένων) και τα ports που εκθέτει η καθεμία. Η επικοινωνία τους γίνεται μέσα από ιδιωτικό δίκτυο Docker, ενώ η πρόσβαση από εξωτερικά συστήματα επιτρέπεται μόνο μέσω των δηλωμένων ports.

Η εγκατάσταση υλοποιήθηκε σε εικονικό υπολογιστή (Virtual Machine) που φιλοξενεί όλες τις υπηρεσίες. Αρχικά, κάθε υπηρεσία αναπτύχθηκε και δοκιμάστηκε ξεχωριστά με το δικό της Docker Compose αρχείο. Μετά την ολοκλήρωση των δοκιμών, οι εικόνες των υπηρεσιών (images) ανέβηκαν στο DockerHub[4]. Στη συνέχεια, δημιουργήθηκε ένα ενιαίο αρχείο docker-compose.yml, το οποίο περιέχει τον πλήρη ορισμό όλων των υπηρεσιών και των ρυθμίσεών τους, επιτρέποντας την εγκατάσταση και εκκίνηση ολόκληρης της πλατφόρμας με μία μόνο εντολή.

Αυτή η προσέγγιση εξασφαλίζει υψηλή επαναληψιμότητα και φορητότητα, καθώς η πλατφόρμα μπορεί να εγκατασταθεί σε οποιοδήποτε περιβάλλον χρησιμοποιώντας το ίδιο αρχείο διαμόρφωσης. Παράλληλα, διατηρείται η ανεξαρτησία και η επεκτασιμότητα του συστήματος, αφού κάθε υπηρεσία μπορεί να ενημερωθεί ή να αντικατασταθεί αυτόνομα, ενώ η προσθήκη νέων λειτουργικοτήτων γίνεται απλά με την προσθήκη νέων εγγραφών στο αρχείο YAML. Η μέθοδος αυτή ευθυγραμμίζεται με τις βέλτιστες πρακτικές ανάπτυξης πλατφορμών

[4]DockerHub: https://hub.docker.com

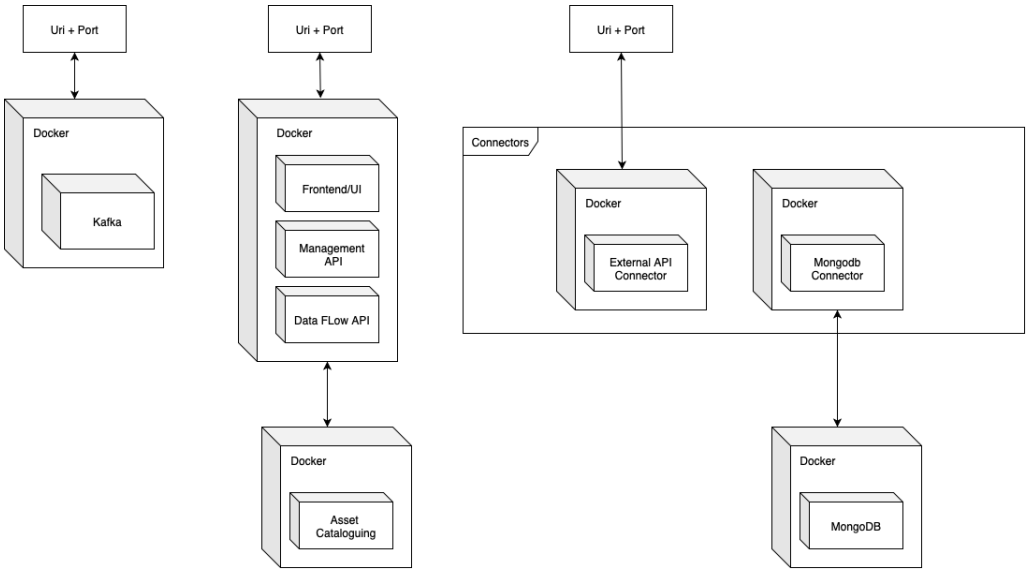
Big Data, οι οποίες απαιτούν ευελιξία, αξιοπιστία και δυνατότητα άμεσης κλιμάκωσης ή επανεκκίνησης των υπηρεσιών.

Το παρακάτω Σχήμα 4.3 παρουσιάζει με απλό τρόπο πώς αναπτύσσεται η πλατφόρμα σε επίπεδο containers. Στο κέντρο φαίνεται το κύριο Docker container, το οποίο περιλαμβάνει το Frontend/UI, το Management API και το Data Flow API. Αυτό το τμήμα είναι το σημείο αλληλεπίδρασης με τον χρήστη και ο κεντρικός διαχειριστής της πλατφόρμας, αφού οργανώνει όλες τις βασικές λειτουργίες.

Ακριβώς από κάτω βρίσκεται το Asset Cataloguing, το οποίο αποτελεί τη βάση καταγραφής όλων των assets και των αντίστοιχων connectors. Στα δεξιά βλέπουμε τα δύο connectors που υλοποιήθηκαν, τον External API Connector και τον MongoDB Connector. Ο πρώτος επιτρέπει τη σύνδεση με εξωτερικές υπηρεσίες μέσω APIs, ενώ ο δεύτερος αναλαμβάνει την επικοινωνία με βάσεις δεδομένων MongoDB. Ο MongoDB Connector συνδέεται απευθείας με τη βάση MongoDB, η οποία φαίνεται στο κάτω μέρος του σχήματος.

Στα αριστερά, απεικονίζεται το Kafka, το οποίο υλοποιείται επίσης σε ξεχωριστό container και είναι υπεύθυνο για τη ροή δεδομένων σε πραγματικό χρόνο. Λειτουργεί ανεξάρτητα, εκθέτοντας τα δικά του ports.

Όλες οι υπηρεσίες επικοινωνούν μεταξύ τους μέσω URIs και ports, ενώ οι connectors λειτουργούν ως γέφυρες ανάμεσα στην πλατφόρμα και τα ετερογενή συστήματα αποθήκευσης ή εξωτερικά APIs. Με αυτόν τον τρόπο το σχήμα δείχνει καθαρά πώς η λογική της αρχιτεκτονικής SOA μεταφέρεται στην πράξη, με κάθε υποσύστημα να λειτουργεί αυτόνομα σε δικό του container, αλλά ταυτόχρονα να συνεργάζεται με όλα τα υπόλοιπα.



Σχήμα 4.3: *Αρχιτεκτονική της Big Data Platform.*

Το Σχήμα 4.3 απεικονίζει την αρχιτεκτονική SOA σε ένα Deployment Diagram.

Κεφάλαιο 5

Αξιολόγηση Συστήματος

Στο κεφάλαιο αυτό περιγράφεται η μεθοδολογία που ακολουθήθηκε για την αξιολόγηση και τον έλεγχο της πλατφόρμας.

5.1 Μεθοδολογία Αξιολόγησης

Η αξιολόγηση της Big Data Platform είχε ως στόχο τη διερεύνηση της αντοχής και της αξιοπιστίας του συστήματος σε συνθήκες μεγάλης κλίμακας. Για να χαρακτηριστεί μια πλατφόρμα ως Big Data, πρέπει να είναι σε θέση να υποστηρίζει την αποθήκευση και επεξεργασία μεγάλων όγκων δεδομένων, εξασφαλίζοντας την απρόσκοπτη και σταθερή λειτουργία της, ακόμη και υπό αυξημένο φόρτο.

Η διαδικασία αξιολόγησης πραγματοποιήθηκε σε δύο διακριτά στάδια. Στο πρώτο στάδιο αξιοποιήθηκε το Swagger API για τον έλεγχο των βασικών λειτουργιών της πλατφόρμας σε χαμηλό επίπεδο. Στο δεύτερο στάδιο, η αξιολόγηση έγινε μέσω του γραφικού περιβάλλοντος χρήστη (UI), προσομοιώνοντας την εμπειρία και τις ενέργειες ενός τελικού χρήστη.

Για τον έλεγχο της πλατφόρμας σχεδιάστηκαν και εξετάστηκαν έξι βασικές απαιτήσεις, οι οποίες προέκυψαν από την ανάλυση των λειτουργικών αναγκών στο Κεφάλαιο 3:

- **1η Απαίτηση**: Διαχείριση μεγάλου όγκου δεδομένων (Heavy Streaming)
- **2η Απαίτηση**: Επεξεργασία ετερογενών δεδομένων
- **3η Απαίτηση**: Διαχείριση πολλαπλών αποθετηρίων
- **4η Απαίτηση**: Υποστήριξη ιστορικών δεδομένων (Batch Streaming)
- **5η Απαίτηση**: Διαχείριση πραγματικών ροών δεδομένων (Real-time Streaming)
- **6η Απαίτηση**: Ανάκτηση, προσθήκη και μεταφορά δεδομένων (Retrieve, Upload, Flow)

Με στόχο την αποτίμηση της απόδοσης της πλατφόρμας σε ρεαλιστικές συνθήκες λειτουργίας που καλύπτουν τις παραπάνω απαιτήσεις, διαμορφώθηκαν τρία σενάρια δοκιμών:

- **Σενάριο Α** (5.2.2): Ροή δεδομένων και αρχείων από Kafka προς αποθετήριο τύπου MongoDB.

- **Σενάριο Β** (5.2.3): Ανάκτηση αρχείων από αποθετήριο τύπου MongoDB.
- **Σενάριο Γ** (5.2.4): Προσθήκη δεδομένων JSON σε αποθετήριο τύπου MongoDB.

Η υλοποίηση των παραπάνω σεναρίων επέτρεψε τη διεξαγωγή δοκιμών που αντικατοπτρίζουν με ακρίβεια ρεαλιστικές συνθήκες λειτουργίας, παρέχοντας αξιόπιστα αποτελέσματα για την απόδοση και την ανθεκτικότητα της πλατφόρμας.

Προκειμένου να καλυφθεί η πρώτη απαίτηση, χρησιμοποιήθηκε ένα αρχείο μεγέθους 13.98 GB, το οποίο κατά τη φόρτωσή του αντιστοιχεί σε σενάριο Heavy Streaming.

Για τον έλεγχο της ετερογένειας των δεδομένων (2η Απαίτηση), στο πρώτο και το δεύτερο σενάριο χρησιμοποιήθηκαν δεδομένα σε μορφή αρχείων (file, parquet), ενώ στο τρίτο σενάριο αξιοποιήθηκαν δεδομένα σε μορφή JSON.

Η τρίτη απαίτηση καλύφθηκε μέσω της χρήσης δύο διαφορετικών αποθετηρίων: Kafka και MongoDB. Στο πρώτο σενάριο πραγματοποιείται πραγματική ροή δεδομένων, κατά την οποία τα δεδομένα μεταφέρονται σε πραγματικό χρόνο (real-time) από το Kafka σε μια βάση MongoDB. Στη συνέχεια, τα ίδια δεδομένα θεωρούνται ως ιστορικά και χρησιμοποιούνται για την εκτέλεση λειτουργιών ανάκτησης και προσθήκης, καλύπτοντας έτσι και την τέταρτη και έκτη απαίτηση.

5.2 Αναλυτική παρουσίαση ελέγχου

Στην ενότητα αυτή παρουσιάζουμε αναλυτικά τον έλεγχο του συστήματος σύμφωνα με τα σενάρια που περιγράφηκαν στην προηγούμενη ενότητα.

5.2.1 Προεργασία για την Εκτέλεση των Σεναρίων

Πριν από την εκτέλεση των σεναρίων πραγματοποιήθηκε η απαραίτητη προετοιμασία, η οποία περιλάμβανε δύο βασικά στάδια: (α) τη δημιουργία όλων των απαιτούμενων assets και connectors, και (β) την ανάπτυξη ενός μηχανισμού προσομοίωσης παραγωγής δεδομένων αισθητήρων, με στόχο την παροχή ρεαλιστικής ροής δεδομένων για τις ανάγκες της αξιολόγησης.

Για την εκτέλεση αξιοποιήθηκε αρχείο μεγέθους 13.98 GB, το οποίο περιείχε ιστορικά δεδομένα πραγματικών αισθητήρων από ορυχείο ενός εκ των πιλότων του έργου. Το αρχείο περιελάμβανε συνολικά 137,200,000 γραμμές δεδομένων. Οι αισθητήρες διέφεραν ως προς τη συχνότητα δειγματοληψίας, η οποία κυμαινόταν μεταξύ 1ms, 10ms και 100ms, δηλαδή αντίστοιχα 10.000 Hz, 100 Hz και 10 Hz.

Για τη διαχείριση και μετάδοση αυτών των δεδομένων αναπτύχθηκε το `parser.js`, ένα Node.js script που διαβάζει το αρχείο εισόδου και το μετατρέπει σε ροή δεδομένων.

Η μετάδοση των δεδομένων πραγματοποιήθηκε μέσω UDP packets. Τα UDP (User Datagram Protocol) packets είναι ελαφριά πακέτα δεδομένων, τα οποία μεταφέρονται χωρίς μηχανισμούς επιβεβαίωσης ή διόρθωσης σφαλμάτων. Έτσι επιτυγχάνεται υψηλή ταχύτητα μετάδοσης, αλλά υπάρχει πιθανότητα απώλειας πακέτων. Στην παρούσα υλοποίηση, το `parser.js` διάβαζε κάθε 1000 γραμμές (δηλαδή ανά 100 ms) και τις έστελνε ως ένα UDP packet στο Edge.

Το Edge υλοποιήθηκε μέσω Node-RED, όπου αναπτύχθηκαν συγκεκριμένα flows με στόχο (α) την αξιόπιστη προώθηση δεδομένων προς τον Kafka, (β) την αποφυγή απωλειών πληροφορίας, και (γ) την αποτροπή υπερφόρτωσης του συστήματος. Εκεί δημιουργήθηκαν επίσης μετρητές (counters) που κατέγραφαν τον όγκο των δεδομένων που προωθούσαν στα αντίστοιχα Kafka topics. Στο Edge διαμορφώθηκαν δύο κύριες ροές επεξεργασίας:

1. **Ροή αρχείων Parquet:** Τα UDP packets εισάγονταν αρχικά σε έναν buffer, όπου παρέμεναν μέχρι να συγκεντρωθούν 600 πακέτα (δηλαδή περίπου 60.000 ms ή 1 λεπτό δεδομένων). Σε περίπτωση που το χρονικό όριο έληγε χωρίς να συμπληρωθεί ο αριθμός, τα πακέτα προωθούνταν κανονικά. Στη συνέχεια, ένα script τα ενοποιούσε σε μορφή `CSV`, το οποίο μετατρεπόταν σε αρχείο Parquet και αποστέλλονταν στο Kafka topic με την ονομασία `Parquet-test`.

2. **Ροή αντικειμένων JSON:** Σε αυτή τη ροή κάθε UDP packet μετατρεπόταν σε JSON object. Υλοποιήθηκαν δύο υπο-ροές:

   (α') *Πλήρης αποθήκευση:* Για κάθε μεταβλητή δημιουργούνταν πίνακας με όλες τις τιμές των 1000 γραμμών του πακέτου. Το αντικείμενο αποστέλλονταν στο Kafka topic `sensorial-test`.

   (β') *Συνοπτική αποθήκευση:* Για κάθε μεταβλητή υπολογιζόταν η μέση τιμή των 1000 γραμμών και αποστέλλονταν μόνο αυτή. Το αντικείμενο αποθηκευόταν στο Kafka topic `short-test`.

Με τον τρόπο αυτό, η πλατφόρμα υποστήριζε τρεις διαφορετικές στρατηγικές ροής δεδομένων προς τον Kafka: (α) ως λεπτομερή JSON objects, (β) ως συνοπτικά JSON objects, και (γ) ως μαζικά αρχεία Parquet. Έτσι διασφαλίστηκε ότι το σύστημα μπορούσε να απορροφήσει και να αποθηκεύσει αποτελεσματικά τα **ετερογενή** δεδομένα αισθητήρων, χωρίς απώλειες και χωρίς υπερφόρτωση.

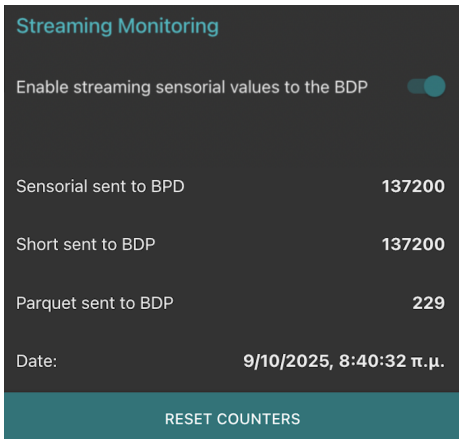
Ο `parser.js` διάβασε αρχείο μεγέθους 13.98 GB με 137,200,000 γραμμές. Για τη ροή αντικειμένων JSON τα μυνήματα στέλνονταν ανά 100 ms οπότε το αναμενόμενο πλήθος είναι:

$$\frac{137,200,000 \text{ γραμμές}}{1000 \text{ γραμμές/μήνυμα}} = 137,200 \text{ μηνύματα}$$

Για τη ροή αρχείων Parquet τα μηνύματα στέλνονταν ανά 60,000 ms (1 λεπτό), οπότε το αναμενόμενο πλήθος μηνυμάτων είναι:

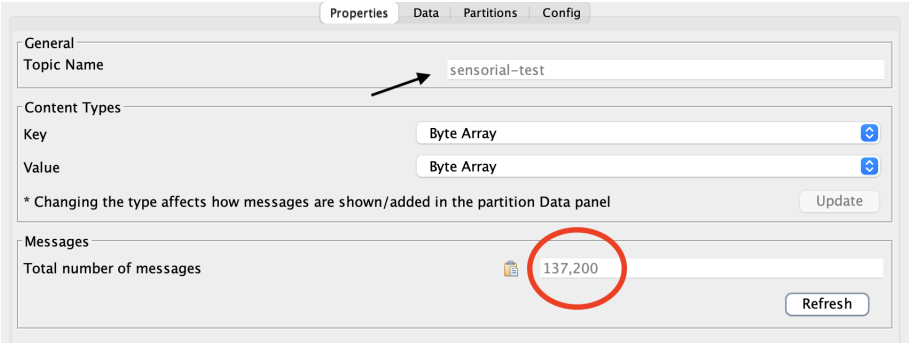
$$\frac{137,200,000 \text{ γραμμές}}{600,000 \text{ γραμμές/μήνυμα}} \approx 229 \text{ μηνύματα}$$

Οι αριθμοί αυτοί επιβεβαιώνονται από την Εικόνα 5.1. Παρακάτω βλέπουμε τους μετρητές του Edge όπου μέτρησαν πόσα πακέτα στάλθηκαν στα τρία topics — `sensorial-test`, `short-test` και `Parquet-test`.



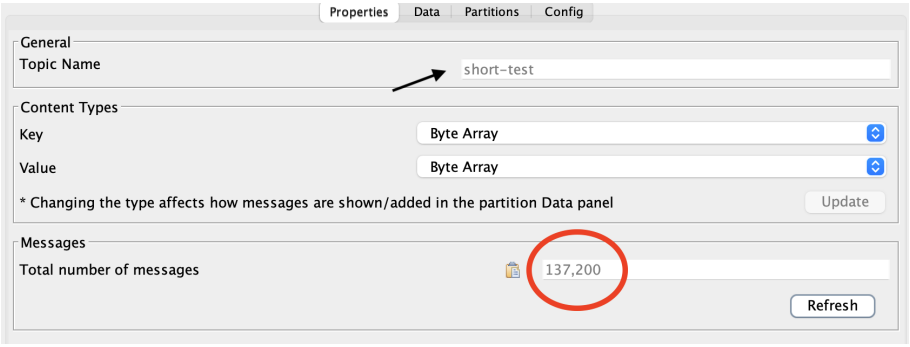
Εικόνα 5.1: *Edge Counters*

Στη συνέχεια, στις Εικόνες 5.2, Εικόνα 5.3 και Εικόνα 5.4 παρουσιάζεται το τελικό αποτέλεσμα του `parser.js`. Τα τρία topics — `sensorial-test`, `short-test` και `Parquet-test` — του Kafka έχουν φορτωθεί επιτυχώς με όλα τα μηνύματα που προήλθαν από το Edge, χωρίς απώλειες, και είναι πλέον έτοιμα, μέσω των ροών, να μετατραπούν σε ιστορικά δεδομένα και να αποθηκευτούν στα αντίστοιχα αποθετήρια. Η παρακολούθηση της ροής και της κατάστασης των μηνυμάτων του Kafka πραγματοποιήθηκε με τη βοήθεια του Offset Explorer[1], ενός εργαλείου παρακολούθησης και διαχείρισης Kafka clusters, το οποίο παρέχει άμεση εποπτεία των topics, των partitions και των μηνυμάτων σε πραγματικό χρόνο, διευκολύνοντας την επαλήθευση της ορθής λειτουργίας του συστήματος.

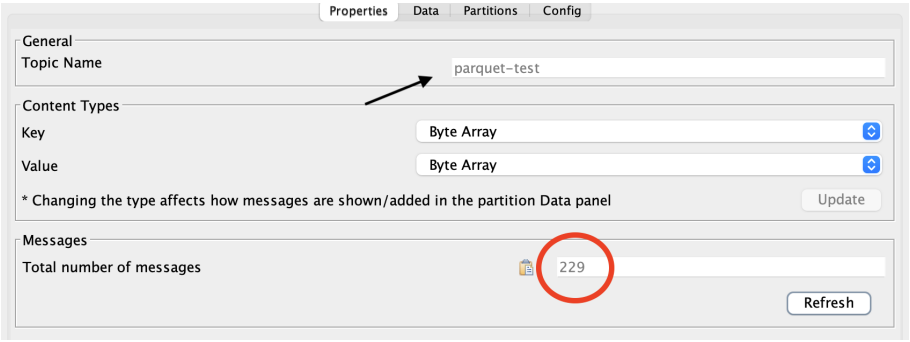


Εικόνα 5.2: *Kafka topic sensorial-test*

[1]Offset Explorer: `https://www.kafkatool.com`

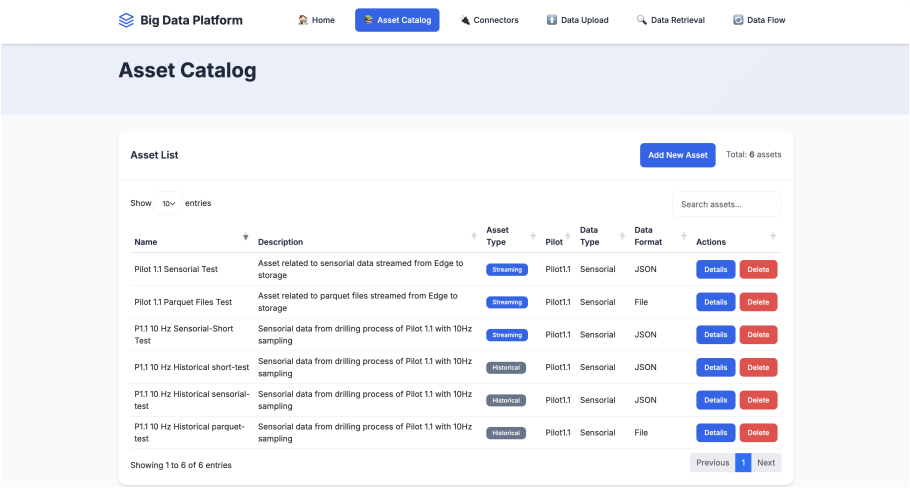


Εικόνα 5.3: *Kafka topic short-test*



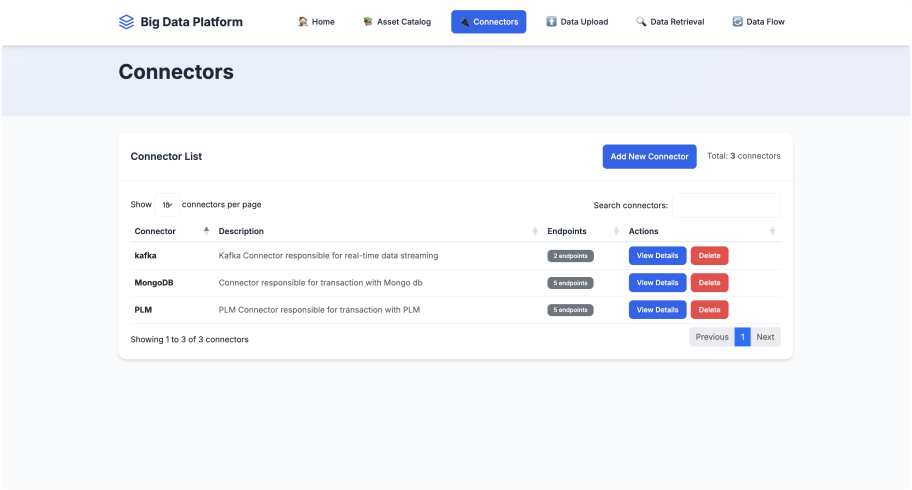
Εικόνα 5.4: *Kafka topic parquet-test*

Στη συνέχεια, όπως φαίνεται και στην Εικόνα 5.5, πραγματοποιήθηκε η εγγραφή όλων των απαραίτητων assets στο υποσύστημα Asset Catalog. Παρατηρείται ότι δημιουργήθηκαν τρία Streaming assets και τρία Historical assets. Τα πρώτα αφορούν την πηγή (source) της ροής δεδομένων και αντιστοιχούν σε δύο Kafka assets για τα topics sensorial-test, short-test και parquet-test. Τα δεύτερα αφορούν τον προορισμό (destination) των ροών δεδομένων και αντιστοιχούν σε δύο αποθετήρια: δύο MongoDB assets για την αποθήκευση δεδομένων από το topic sensorial-test και topic short-test αντίστοιχα και ένα PLM asset για την αποθήκευση αρχείων από το topic parquet-test.



Εικόνα 5.5: *Asset Catalog*

Αντίστοιχα, όπως φαίνονται στην επόμενη Εικόνα 5.6, δηλώθηκαν οι connectors, ένας Kafka connector, ένας MongoDB connector και ένας PLM connector, με τα αντίστοιχα endpoints τους για την ανάκτηση και αποθήκευση δεδομένων και αρχείων.



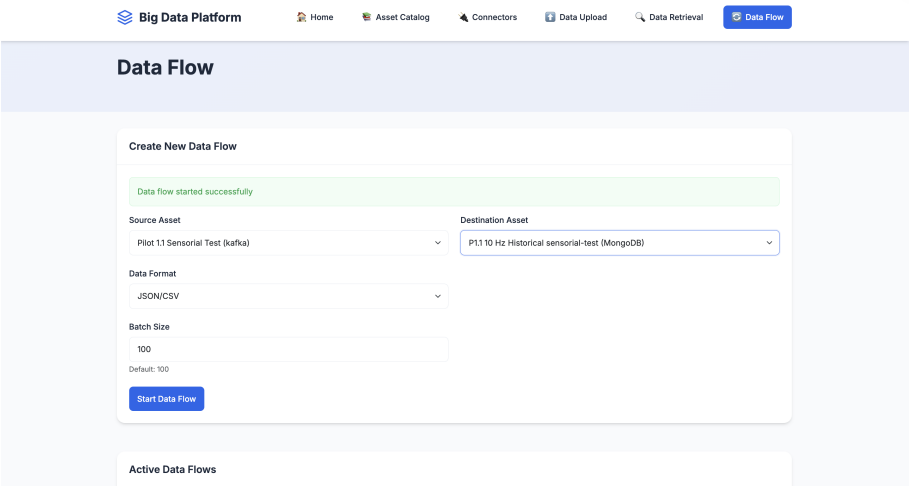
Εικόνα 5.6: *Connector Catalog*

Αφού ολοκληρώθηκε η δημιουργία των assets και connectors, υλοποιήθηκαν τα ενεργά flows για τα δύο σενάρια αξιολόγησης που παρουσιάζονται στην επόμενη ενότητα.

5.2.2 Σενάριο Α: Ροή δεδομένων και αρχείων από Kafka προς αποθετήριο τύπου MongoDB

Για τη δοκιμή και αξιολόγηση της πλατφόρμας υλοποιήθηκε ένα σενάριο ροής δεδομένων, το οποίο εκκινεί από το Kafka topic sensorial-test και καταλήγει σε ένα collection με όνομα sensorial-test στη βάση MongoDB.

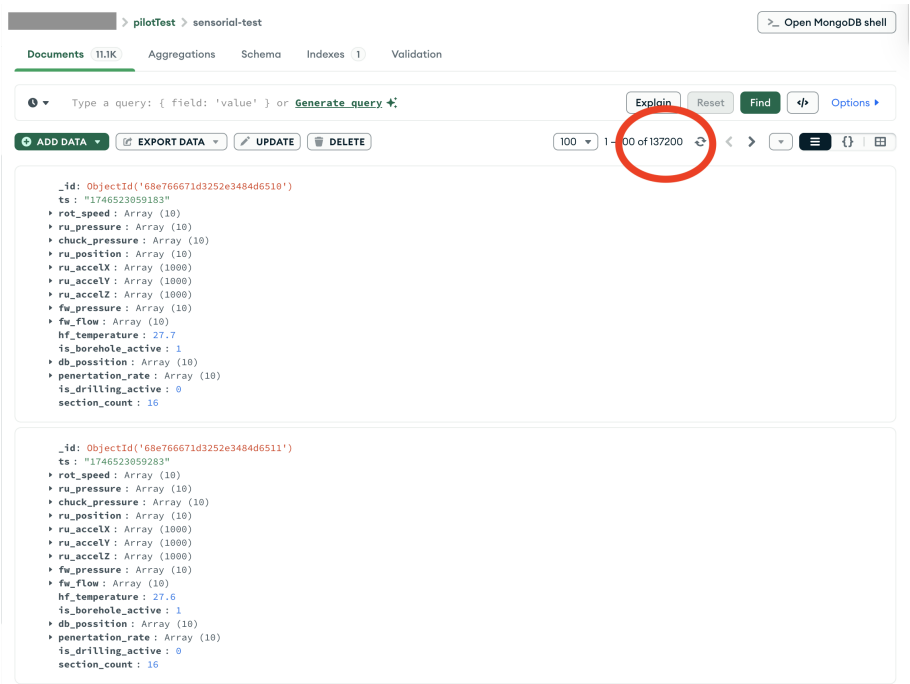
Αρχικά, αφού είχε προηγηθεί η εγγραφή όλων των απαραίτητων assets, δημιουργήθηκε ένα νέο flow με source asset το Kafka topic sensorial-test και destination asset το MongoDB collection sensorial-test. Για τη ροή δεδομένων επιλέχθηκε μορφή JSON/CSV και batch size ίσο με 100. Στην Εικόνα 5.7 παρουσιάζεται η επιτυχής δημιουργία της ροής.



Εικόνα 5.7: *Creation of Data Flow: sensorial-test (Kafka) -> sensorial-test (MongoDB)*

Η ορθότητα της διαδικασίας επαληθεύεται συγκρίνοντας τον αριθμό μηνυμάτων που είχαν φορτωθεί στο Kafka topic με τον αριθμό εγγραφών που καταχωρήθηκαν στο αντίστοιχο collection της MongoDB. Όπως φαίνεται στην Εικόνα 5.8, η μεταφορά των δεδομένων ολοκληρώθηκε χωρίς απώλειες ή σφάλματα.

Η παρακολούθηση και επιβεβαίωση της επιτυχούς αποθήκευσης των δεδομένων πραγματοποιήθηκε με τη χρήση του εργαλείου MongoDB Compass[2], το οποίο παρέχει δυνατότητες γραφικής απεικόνισης και άμεσου ελέγχου των συλλογών δεδομένων.

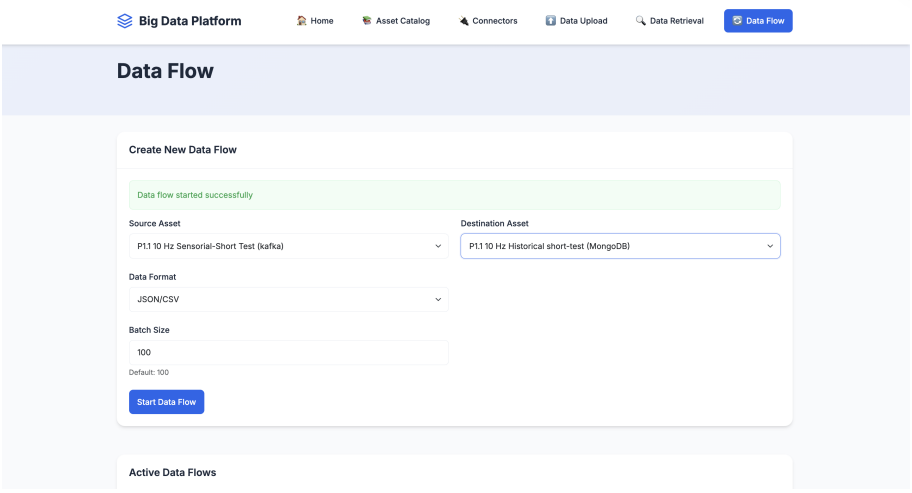


Εικόνα 5.8: *Collection Sensorial-test at MongoDB*

Αντίστοιχα, στην Εικόνα 5.9 παρουσιάζεται η επιτυχής δημιουργία της ροής flow με

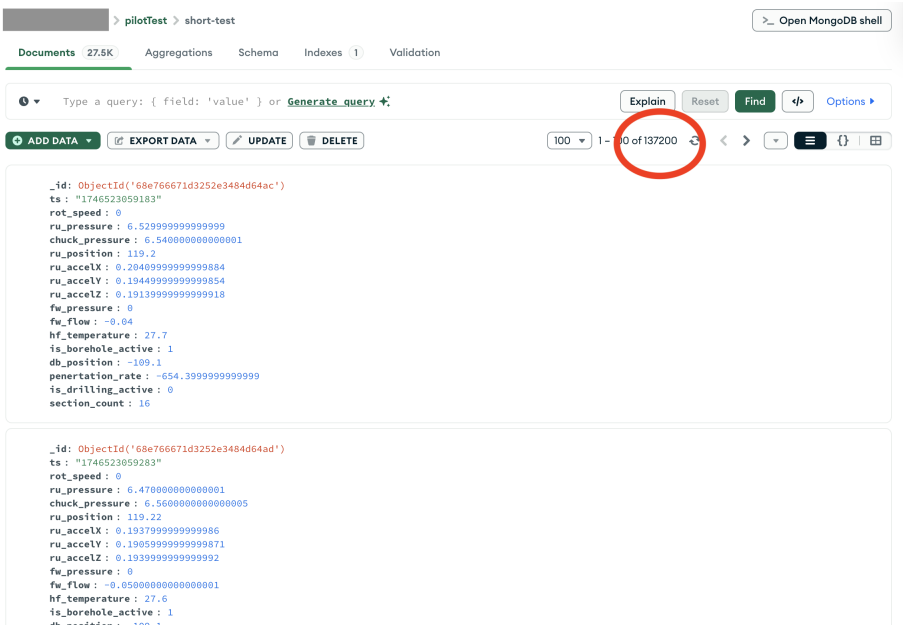
[2]MongoDB Compass: https://www.mongodb.com/products/tools/compass

source asset το Kafka topic short-test και destination asset το MongoDB collection short-test.



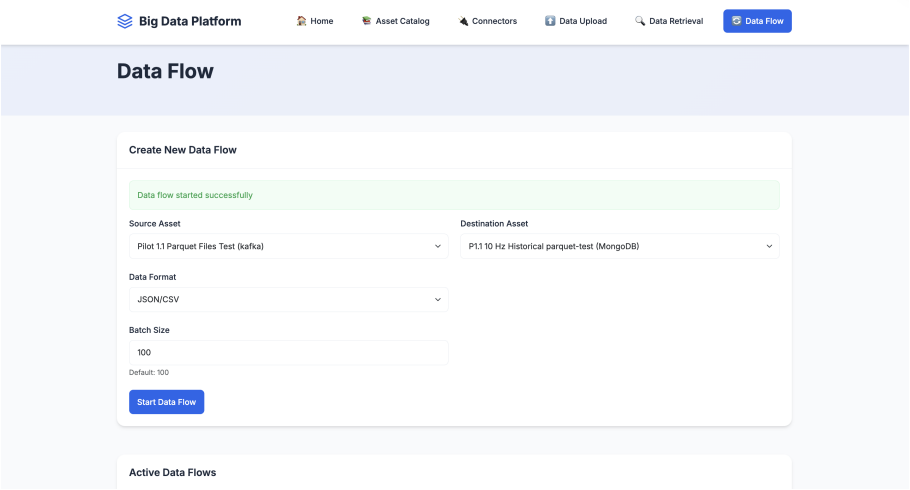
Εικόνα 5.9: *Creation of Data Flow: short-test (Kafka) -> short-test (MongoDB)*

Η ορθότητα της διαδικασίας επαληθεύεται συγκρίνοντας τον αριθμό μηνυμάτων που είχαν φορτωθεί στο Kafka topic με τον αριθμό εγγραφών που καταχωρήθηκαν στο αντίστοιχο collection της MongoDB. Όπως φαίνεται στην Εικόνα 5.10, η μεταφορά των δεδομένων ολοκληρώθηκε χωρίς απώλειες ή σφάλματα.



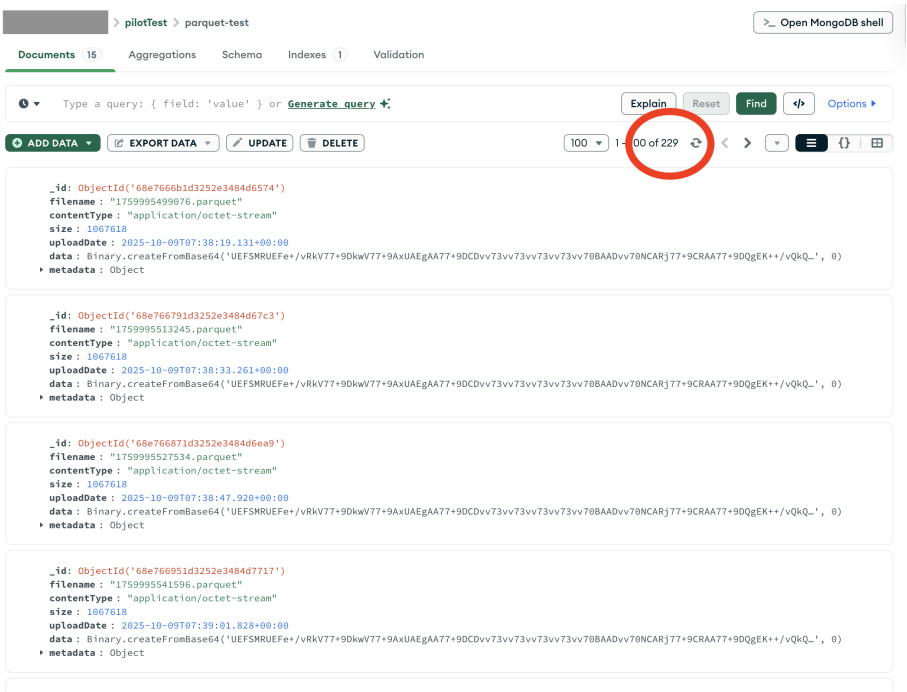
Εικόνα 5.10: *Collection Short-test at MongoDB*

Τέλος, στην Εικόνα 5.11 παρουσιάζεται η επιτυχής δημιουργία της ροής flow αρχείων με source asset το Kafka topic parquet-test, destination asset το MongoDB collection parquet-test και επιλογή Data Format = Files.



Εικόνα 5.11: *Creation of Data Flow: parquet-test (Kafka) -> parquet-test (MongoDB)*

Η ροή επαληθεύεται συγκρίνοντας τον αριθμό μηνυμάτων που είχαν φορτωθεί στο Kafka topic με τον αριθμό εγγραφών που καταχωρήθηκαν στο αντίστοιχο collection της MongoDB. Όπως φαίνεται στην Εικόνα 5.12, η μεταφορά των δεδομένων ολοκληρώθηκε χωρίς απώλειες ή σφάλματα.



Εικόνα 5.12: *Collection Parquet-test at MongoDB*

Τέλος, η ορθότητα τεκμηριώνεται και από τη μορφή των εγγραφών στη MongoDB. Συγκεκριμένα: (i) στο sensorial-test κάθε μεταβλητή αποθηκεύεται ως πίνακας μετρήσεων (ii) στο short-test κάθε μεταβλητή έχει μοναδική τιμή τη μέση τιμή του αντίστοιχου πίνακα, (iii) στο parquet-test κάθε μήνυμα αποθηκεύεται ως Binary (Base64) μαζί με μεταδεδομένα (`filename`, `contentType`, `size`, `uploadDate`, `metadata`), ώστε το αρχικό αρχείο Parquet να μπορεί

να ανασυντεθεί αυτούσιο και να παραχθεί εκ νέου ως αρχείο όταν απαιτηθεί.

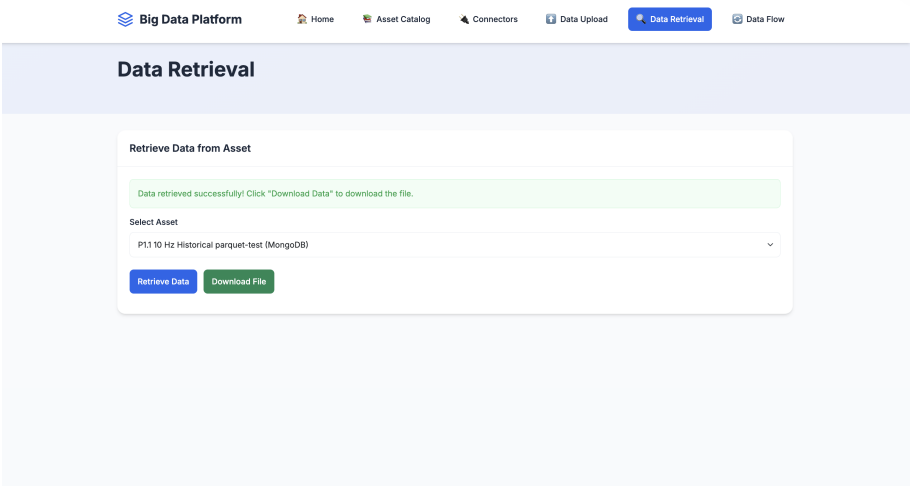
	sensorial-test	**short-test**	**parquet-test**
Τύπος Δεδομένων	JSON	JSON	File
Αριθμός Μηνυμάτων στον Kafka	137.200	137.200	229
Αριθμός Εγγραφών στη Mongodb	137.200	137.200	229
Ακρίβεια	**100%**	**100%**	**100%**

Πίνακας 5.1: *Σύνοψη Αξιολόγησης για το Σενάριο Α*

5.2.3 Σενάριο Β: Ανάκτηση αρχείων από αποθετήριο τύπου MongoDB

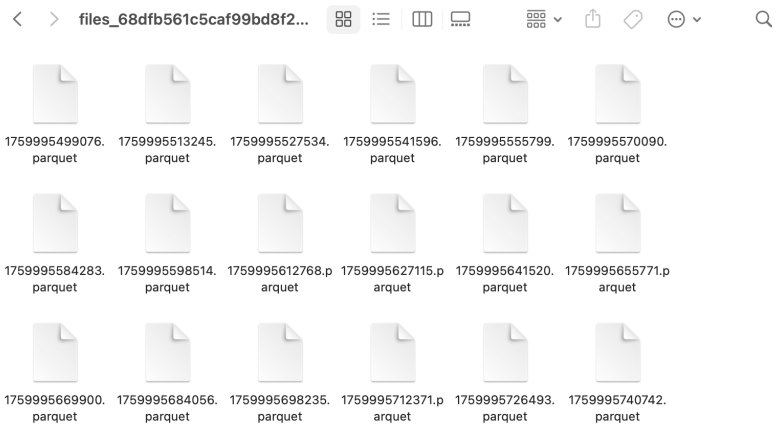
Για τη δοκιμή και αξιολόγηση της πλατφόρμας ως προς το Data Retrieval, δηλαδή την ανάκτηση δεδομένων ή αρχείων από αποθετήρια, συνεχίστηκε το προηγούμενο σενάριο με επόμενο βήμα την ανάκτηση των αρχείων που είχαν μεταφερθεί με ζωντανή ροή από το Kafka topic parquet-test. Στόχος ήταν να εξεταστούν τα όρια και ο μέγιστος αριθμός αρχείων που μπορούν να ανακτηθούν και να συμπιεστούν σε μορφή zip.

Στο UI, μέσω του ανώτερου navigation menu, επιλέχθηκε η λειτουργία Data Retrieval, ορίστηκε ως asset το Historical parquet-test (MongoDB) και εκτελέστηκε η εντολή Data Retrieve. Όπως παρουσιάζεται στην Εικόνα 5.13, η ανάκτηση ολοκληρώθηκε επιτυχώς και τα αρχεία συμπιέστηκαν σε ένα zip αρχείο, διαθέσιμο προς λήψη μέσω της επιλογής Download File.



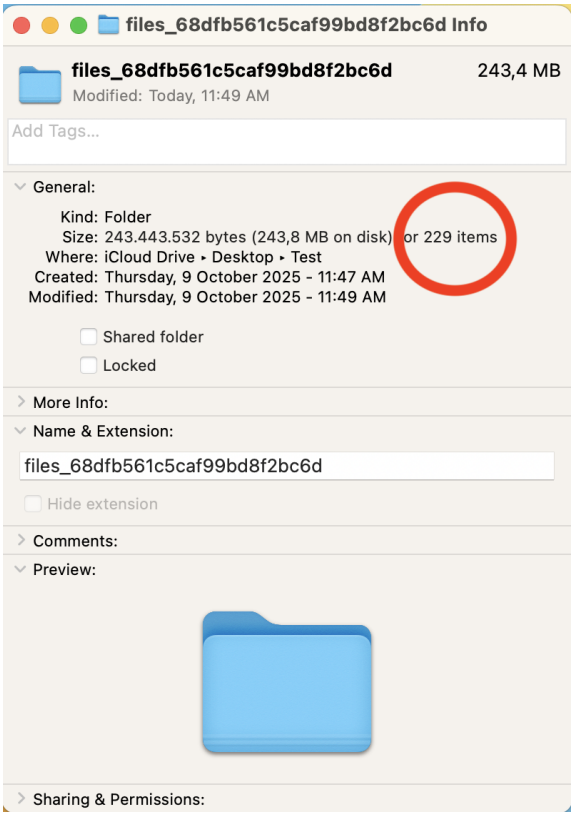
Εικόνα 5.13: *Parquet Files Retrieval from MongoDB as .zip*

Πατώντας Download File το αρχείο κατέβηκε επιτυχώς και στην παρακάτω Εικόνα 5.14 βλέπουμε το περιεχόμενο του φακέλου αφού αποσυμπιέσαμε το zip.



Εικόνα 5.14: *Parquet Files Directory*

Η ορθότητα της διαδικασίας επαληθεύτηκε συγκρίνοντας τον αριθμό αρχείων που είχαν καταχωρηθεί στον collection της MongoDB με τον αριθμό αρχείων που περιλαμβάνονταν στο zip. Το αρχείο εισόδου περιείχε 137,200,000 γραμμές δεδομένων και δημιουργήθηκαν συνολικά 229 αρχεία Parquet. Ο ίδιος αριθμός επιβεβαιώθηκε μετά την αποσυμπίεση (unzip) του zip αρχείου, όπως φαίνεται στην Εικόνα 5.15, γεγονός που καταδεικνύει ότι η διαδικασία ανάκτησης ολοκληρώθηκε χωρίς απώλειες ή σφάλματα.



Εικόνα 5.15: *File Count in Parquet Directory after unzip*

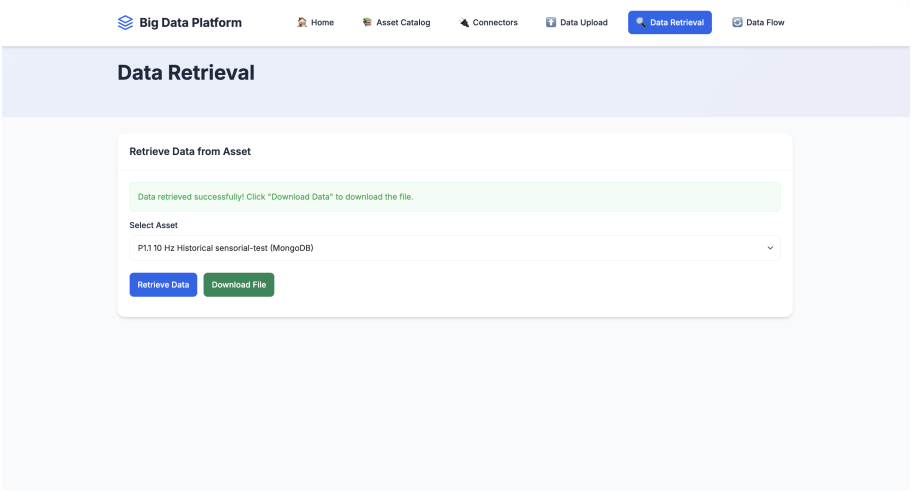
	Πριν από Ανάκτηση	**Μετά από Ανάκτηση**
Τύπος Δεδομένων	File	.zip
Αποθετήριο	MongoDB Database	Local Filesystem
Αριθμός Αρχείων στη Mongodb	229	.zip (229 files unzipped)
Ακρίβεια	**100%**	**100%**

Πίνακας 5.2: *Σύνοψη Αξιολόγησης για το Σενάριο Β*

5.2.4 Σενάριο Γ: Προσθήκη ιστορικών δεδομένων σε αποθετήριο τύπου MongoDB

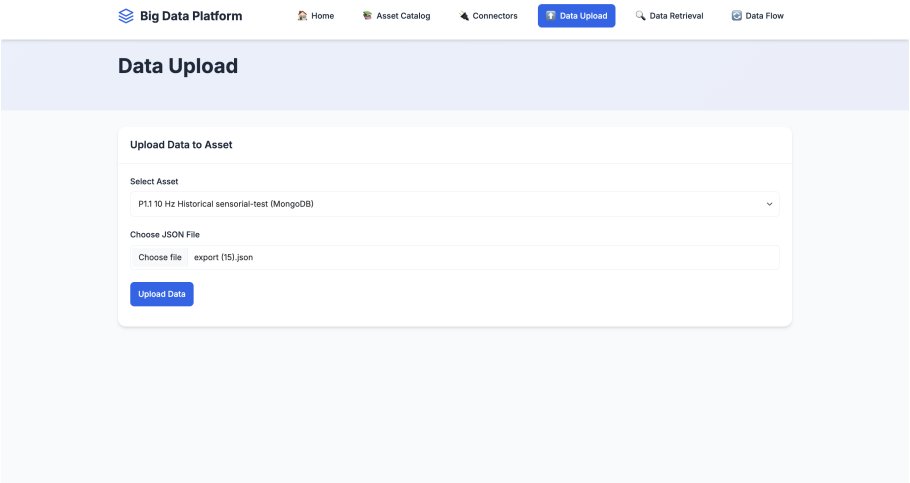
Για τη δοκιμή και αξιολόγηση της πλατφόρμας ως προς το Data Upload, δηλαδή την προσθήκη ιστορικών δεδομένων σε αποθετήριο, συνεχίστηκε το πρώτο σενάριο. Συγκεκριμένα, αφού πραγματοποιήθηκε η ανάκτηση των δεδομένων μέσω Data Retrieve, τα δεδομένα αυτά επανεισάχθηκαν στο αντίστοιχο collection με όνομα sensorial-test της βάσης MongoDB. Με τον τρόπο αυτό ελέγχθηκαν τόσο η διαδικασία ανάκτησης όσο και η διαδικασία προσθήκης δεδομένων.

Στο UI, από το ανώτερο navigation menu, επιλέχθηκε η λειτουργία Data Retrieval, με asset το Historical Sensorial Test (MongoDB) και εκτελέστηκε η εντολή Data Retrieve. Η διαδικασία ολοκληρώθηκε επιτυχώς και το αρχείο δεδομένων κατέστη διαθέσιμο προς λήψη, όπως φαίνεται στην Εικόνα 5.16.



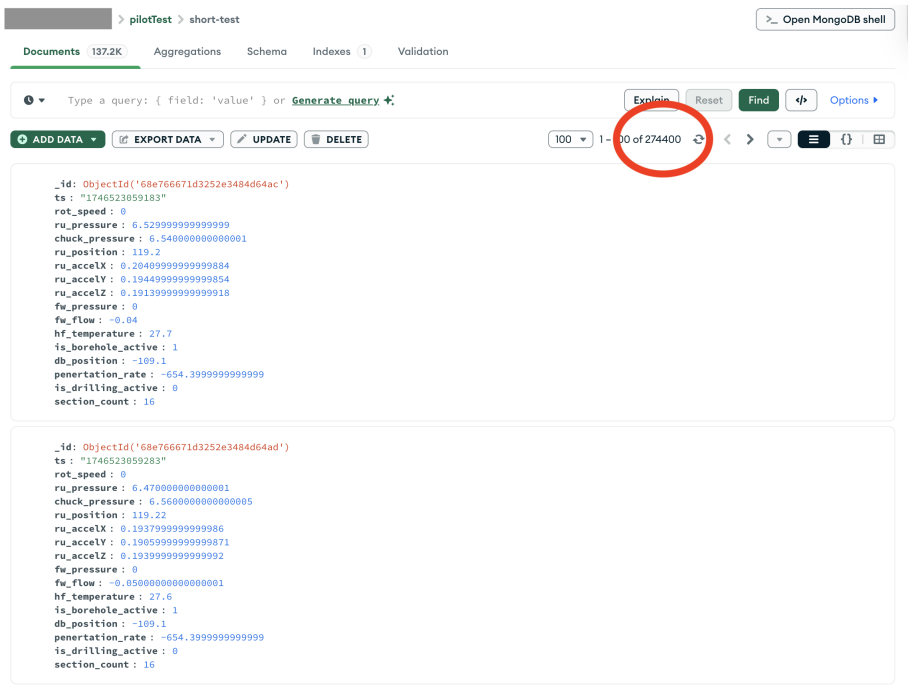
Εικόνα 5.16: *Data Retrieve from collection sensorial-test (MongoDB)*

Στο επόμενο βήμα πραγματοποιήθηκε η προσθήκη των δεδομένων. Από το UI, μέσω της επιλογής Data Upload, επιλέχθηκε ως asset το Historical Sensorial Test (MongoDB) και στη συνέχεια έγινε η μεταφόρτωση του αρχείου που είχε προηγουμένως ανακτηθεί. Η διαδικασία ολοκληρώθηκε επιτυχώς, όπως φαίνεται στην Εικόνα 5.17.



Εικόνα 5.17: *Data Upload to collection sensorial-test (MongoDB)*

Η ορθότητα της διαδικασίας επαληθεύτηκε συγκρίνοντας τον αριθμό των εγγραφών στο collection sensorial-test πριν και μετά την προσθήκη. Αρχικά το collection περιείχε 137,200 documents. Μετά την προσθήκη, ο αριθμός διπλασιάστηκε σε 274,400 documents, όπως φαίνεται και στο MongoDB Compass (Εικόνα 5.18), γεγονός που επιβεβαιώνει την ορθή εκτέλεση της διαδικασίας.



Εικόνα 5.18: *Data Count in Sensorial-test Collection at MongoDB*

Παρακάτω στον Πίνακα 5.3 παρουσιάζονται συνοπτικά τα αποτελέσματα του Σεναρίου Γ. Φαίνονται οι εγγραφές στη βάση MongoDB πριν και μετά τη προσθήκη των νέων δεδομένων που έγινε με ακρίβεια **100%**.

	Πριν από Προσθήκη	Μετά από Προσθήκη
Τύπος Δεδομένων	JSON	JSON
Αποθετήριο	MongoDB Database	MongoDB Database
Αριθμός Εγγραφών	137.200	274.400
Ακρίβεια	**100%**	

Πίνακας 5.3: *Σύνοψη Αξιολόγησης για το Σενάριο Γ*

5.2.5 Σύνοψη Αξιολόγησης

Παρακάτω συνοψίζονται τα αποτελέσματα των τριών σεναρίων αξιολόγησης (Α–Γ), τα οποία χαρτογραφούν τις έξι απαιτήσεις που ορίστηκαν στην Ενότητα 5.1. Τα ευρήματα επιβεβαιώνουν την ορθή και αξιόπιστη λειτουργία της πλατφόρμας σε συνθήκες μεγάλης κλίμακας: επιτεύχθηκε **100% ακρίβεια** χωρίς απώλειες δεδομένων, με πλήρη ταύτιση του αναμενόμενου και παρατηρηθέντος πλήθους μηνυμάτων/αρχείων. Συνολικά, η πλατφόρμα απέδειξε ότι μπορεί να διαχειριστεί ετερογενή δεδομένα, πολλαπλά αποθετήρια τόσο σε ροές real-time όσο και batch, διασφαλίζοντας συνεπή αποθήκευση και ανάκτηση.

	Σενάριο Α	Σενάριο Β	Σενάριο Γ
Τύπος Δεδομένων	sensorial-test: JSON short-test: JSON parquet-test: Parquet	Files | Parquet	JSON
Αποθετήριο Πηγής	Kafka Producer	MongoDB Database	Local File System
Αριθμός Εγγραφών Πηγής	sensorial-test: 137.200 short-test: 137.200 parquet-test: 229	229	137.200
Αποθετήριο Προορισμού	MongoDB Database	Local File System	MongoDB Database
Αριθμός Εγγραφών Προορισμού	sensorial-test: 137.200 short-test: 137.200 parquet-test: 229	229	Αρχικά: 137.200 ، Τελικά: 274.400
Ακρίβεια	**100%**	**100%**	**100%**

Πίνακας 5.4: *Σύνοψη Αξιολόγησης για όλα τα Σενάρια*

5.2.6 Σχόλια και Βελτιώσεις

Κατά τη διάρκεια του ελέγχου της πλατφόρμας σε χαμηλό επίπεδο μέσω Swagger API, εντοπίστηκαν σημαντικά προβλήματα στη ροή δεδομένων. Η αρχική υλοποίηση βασιζόταν στην άμεση αποστολή κάθε μηνύματος με αναλογία ένα-προς-ένα από τους Kafka consumers προς τους connectors. Η προσέγγιση αυτή οδήγησε σε υπερφόρτωση του συστήματος, με αποτέλεσμα απώλειες δεδομένων, καθυστερήσεις και σφάλματα όπως 500 HTTP errors, timeouts και ECONNREFUSED. Η τελευταία κατηγορία σφαλμάτων οφειλόταν στην

έλλειψη διαθέσιμων συνδέσεων προς τις βάσεις δεδομένων, καθώς ο μεγάλος όγκος συνεχόμενων αιτημάτων εξάντλησε τις δυνατότητες του συστήματος.

Η ανάλυση πραγματοποιήθηκε με εργαλεία όπως το MongoDB Compass και το Offset Explorer, τα οποία ανέδειξαν ότι η κύρια αιτία ήταν ο υπερβολικά υψηλός ρυθμός αιτημάτων προς τις βάσεις. Για την αντιμετώπιση του προβλήματος εφαρμόστηκαν οι εξής βελτιώσεις:

- Υιοθέτηση batch processing, ώστε τα δεδομένα να αποστέλλονται σε ομάδες και όχι μεμονωμένα.
- Εισαγωγή μηχανισμού buffering, που ρυθμίζει την ταχύτητα αποστολής δεδομένων.
- Χρήση connection pooling, ώστε οι ενεργές συνδέσεις με τις βάσεις να επαναχρησιμοποιούνται και να αποφεύγεται η δημιουργία νέων σε κάθε αίτημα.
- Αξιοποίηση των μεθόδων `consumer.pause()` και `consumer.resume()`, για προσωρινή παύση της κατανάλωσης σε περιπτώσεις υπερφόρτωσης.

Με τις παρεμβάσεις αυτές, η πλατφόρμα σταθεροποιήθηκε και οι ροές δεδομένων εκτελέστηκαν χωρίς απώλειες ακόμη και σε συνθήκες heavy streaming. Οι δοκιμές φόρτου (stress tests) επιβεβαίωσαν ότι το σύστημα μπορεί πλέον να υποστηρίξει ρυθμό μηνυμάτων έως και ένα κάθε 1ms, διατηρώντας την ακεραιότητα και την αξιοπιστία των δεδομένων.

Κεφάλαιο 6

Επίλογος

Στην τελευταία αυτή ενότητα συνοψίζουμε παρουσιάζοντας συμπεράσματα σχετικά με το σύνολο της εργασίας καθώς και κάποια από τα βασικά προβλήματα που αντιμετωπίσαμε. Τέλος προτείνουμε πιθανές μελλοντικές επεκτάσεις.

6.1 Σύνοψη και Γενικά Συμπεράσματα

Η παρούσα διπλωματική εργασία είχε ως βασικό στόχο τον σχεδιασμό και την υλοποίηση μιας ενιαίας, επεκτάσιμης πλατφόρμας αποθήκευσης και ανάκτησης ετερογενών δεδομένων από πολλαπλά αποθετήρια, ικανής να διαχειρίζεται τόσο ροές δεδομένων σε πραγματικό χρόνο όσο και ιστορικά δεδομένα. Η ανάγκη για μια τέτοια πλατφόρμα προκύπτει από την ετερογένεια και τον κατακερματισμό των δεδομένων στη μεταλλευτική βιομηχανία, όπου απουσιάζουν κοινά πρότυπα που να διασφαλίζουν διαλειτουργικότητα και αποδοτική αξιοποίηση της πληροφορίας.

Η ανάπτυξη της λύσης πραγματοποιήθηκε στο πλαίσιο του ευρωπαϊκού έργου MINE.IO, το οποίο στοχεύει στη βελτιστοποίηση της διαχείρισης δεδομένων στον χώρο της εξορυκτικής δραστηριότητας μέσω της ψηφιοποίησης των υποδομών και των διαδικασιών. Η προτεινόμενη Big Data Platform λειτουργεί ως ενιαίο σημείο πρόσβασης για την αποθήκευση και αναζήτηση δεδομένων διαφορετικών τύπων, υποστηρίζοντας τη σύνδεση πολλαπλών αποθετηρίων (π.χ. MongoDB, Kafka, εξωτερικά APIs) μέσω δυναμικών connectors. Το Asset Cataloguing αποτελεί τον κεντρικό μηχανισμό οργάνωσης και παρακολούθησης των αποθετηρίων, ενώ η χρήση της MongoDB ως asset registry προσφέρει ευελιξία στην αποθήκευση και επέκταση μεταδεδομένων.

Η ανάπτυξη της πλατφόρμας πραγματοποιήθηκε σε στενή συνεργασία με τους πιλότους του έργου, ώστε η υλοποίηση να ανταποκρίνεται σε πραγματικές επιχειρησιακές ανάγκες. Από τα πρώτα στάδια, οι πιλότοι συνέβαλαν στον καθορισμό απαιτήσεων και προδιαγραφών, παρέχοντας λεπτομερείς περιγραφές των διαδικασιών εξόρυξης και αντιπροσωπευτικά datasets με δεδομένα αισθητήρων και γεωχωρικές πληροφορίες. Κατά τη διάρκεια της υλοποίησης, η ανατροφοδότησή τους οδήγησε σε κρίσιμες βελτιστοποιήσεις, όπως η μετάβαση από JSON σε CSV και στη συνεχεία σε Parquet με τεχνικές συμπίεσης, ώστε να καταστεί δυνατή η αποδοτική διαχείριση τεράστιων όγκων δεδομένων υψηλής συχνότητας (1ms).

Η στενή αυτή συνεργασία διασφάλισε ότι η πλατφόρμα δεν περιορίζεται σε θεωρητικό επίπεδο αλλά υποστηρίζει πραγματικά σενάρια λειτουργίας, αντέχοντας σε συνθήκες υψηλού

φόρτου και σύνθετων απαιτήσεων. Μετά την ολοκλήρωση των δοκιμών φόρτου (stress testing), η πλατφόρμα αξιοποιείται ήδη επιχειρησιακά από δύο πιλότους, οι οποίοι αποστέλλουν live stream δεδομένα σε εξατομικευμένα αποθετήρια μέσω ενεργών ροών.

Συμπερασματικά, η εργασία αυτή αποδεικνύει ότι είναι εφικτή η ανάπτυξη μιας επεκτάσιμης και ευέλικτης Big Data Platform για τη μεταλλευτική βιομηχανία, η οποία γεφυρώνει την απόσταση μεταξύ ετερογενών πηγών και εφαρμογών. Επιπλέον, θέτει βάσεις για μελλοντική ενσωμάτωση τεχνικών advanced analytics και machine learning, καθώς και για τη δημιουργία επιπλέον connectors που θα επεκτείνουν ακόμη περισσότερο τις δυνατότητές της. Η πλατφόρμα μπορεί έτσι να αποτελέσει σημείο αναφοράς για αντίστοιχες υλοποιήσεις σε άλλους επιστημονικούς και βιομηχανικούς τομείς.

6.2 Μελλοντικές Επεκτάσεις

Η παρούσα εργασία μπορεί να αποτελέσει αφετηρία για περαιτέρω ανάπτυξη και εμπλουτισμό της πλατφόρμας, ώστε να ενισχυθεί η χρηστικότητα, η απόδοση και η επεκτασιμότητά της. Μερικές ενδεικτικές κατευθύνσεις για μελλοντικές επεκτάσεις είναι οι εξής:

- **Ενσωμάτωση εργαλείων οπτικοποίησης δεδομένων:** Τα δεδομένα που ανακτά ο χρήστης θα μπορούσαν να παρουσιάζονται απευθείας στην πλατφόρμα μέσω δυναμικών γραφημάτων και διαδραστικών πινάκων. Με αυτόν τον τρόπο, ο χρήστης θα έχει τη δυνατότητα τόσο να εξετάζει τα δεδομένα οπτικά, όσο και να τα επιλέξει και να κατεβάσει σε μορφή αρχείου της επιλογής του (JSON, CSV, Excel).

- **Χρήση Elasticsearch για προηγμένη αναζήτηση:** Η ενσωμάτωση του Elasticsearch θα επιτρέψει την ταχύτερη και πιο αποδοτική αναζήτηση μέσα σε μεγάλους όγκους δεδομένων, προσφέροντας δυνατότητες φιλτραρίσματος, ταξινόμησης και εύρεσης συσχετίσεων.

- **Ενσωμάτωση με Hadoop Distributed File System (HDFS):** Η αξιοποίηση του Hadoop για αποθήκευση αρχείων και πολυμεσικού υλικού (π.χ. εικόνες, μεγάλα datasets) θα προσφέρει μεγαλύτερη επεκτασιμότητα και ανθεκτικότητα, επιτρέποντας την υποστήριξη ακόμη μεγαλύτερου όγκου πληροφορίας.

- **Αυτοματοποιημένη εγκατάσταση αποθετηρίων:** Κατά την προσθήκη ενός νέου αποθετηρίου, η πλατφόρμα θα μπορούσε να προσφέρει τη δυνατότητα αυτόματης εγκατάστασης της αντίστοιχης βάσης δεδομένων ή υπηρεσίας. Για παράδειγμα, εάν ο χρήστης επιλέξει να αποθηκεύσει τα δεδομένα του σε μια βάση MongoDB, θα μπορεί να ορίσει τις απαραίτητες παραμέτρους (π.χ. server, port, credentials) και η πλατφόρμα θα αναλαμβάνει την αυτόματη εγκατάσταση και προετοιμασία της βάσης, πριν προχωρήσει η μεταφορά των δεδομένων. Με αυτόν τον τρόπο, μειώνεται η ανάγκη χειροκίνητης παρέμβασης και αυξάνεται ο βαθμός αυτοματοποίησης και ευχρηστίας του συστήματος.

- **Προσθήκη μηχανισμών μηχανικής μάθησης:** Η αξιοποίηση αλγορίθμων machine learning για την αυτόματη κατηγοριοποίηση, πρόβλεψη ή ανακάλυψη προτύπων στα

δεδομένα θα ενίσχυε την αξία της πλατφόρμας, καθιστώντας την εργαλείο όχι μόνο αποθήκευσης αλλά και εξαγωγής γνώσης.

- **Βελτίωση της ασφάλειας:** Επέκταση των μηχανισμών αυθεντικοποίησης και εξουσιοδότησης, π.χ. με χρήση OAuth 2.0 και JWT, καθώς και ενσωμάτωση συστημάτων ελέγχου πρόσβασης με βάση ρόλους (Role-Based Access Control – RBAC) για ακριβέστερη διαχείριση δικαιωμάτων χρηστών.

Με αυτόν τον τρόπο, η πλατφόρμα μπορεί να εξελιχθεί σε ένα πιο ολοκληρωμένο οικοσύστημα διαχείρισης και ανάλυσης δεδομένων, ικανό να καλύψει πιο εξειδικευμένες ανάγκες χρηστών.

Βιβλιογραφία

[1] Chong Qi. *Big data management in the mining industry*. *Journal of Central South University*, 27(9):2480–2492, 2020.

[2] Jonatan Adam Fekete. *Big Data in Mining Operations: Change Management and the Impact of Industrial Analytics*. Master's thesis, Copenhagen Business School, 2015.

[3] Jianqing Fan, Fang Han και Han Liu. *Challenges of Big Data Analysis*. *National Science Review*, 1(2):293–314, 2014.

[4] Jiafu Wang, Yan Ma, Lin Zhang, Robert X. Gao και Di Wu. *Industrial Big Data Analytics: Challenges, Methodologies, and Applications*. *Journal of Manufacturing Systems*, 48:144–156, 2018.

[5] Aditi Rajesh Nimodiya και Shruti Sunil Ajankar. *A Review on Internet of Things*. *International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)*, 2(1):135–144, 2022.

[6] Sachin Kumar, Pragya, Shane Fatima και Rajeev Nigam. *IoT Assisted Optimization of Resources for Healthcare, Mining and Transportation Industry*. *Asian Journal of Research in Computer Science*, 18(4):505–514, 2025.

[7] Mahshad Mahmoudian, S. M. Zanjani, Hossein Shahinzadeh, Yasin Kabalci, Ersan Kabalci και others. *An Overview of Big Data Concepts, Methods, and Analytics: Challenges, Issues, and Opportunities*. *Proceedings of the 2023 IEEE Global Power, Energy and Communication Conference (GPECOM)*, σελίδες 554–559, 2023.

[8] Swati Gupta, Meenu Vijarania, Anjali Gautam και Aarti Yadav. *IoT and Big Data Security Issues and Challenges: A Technological Perspective*. *Intelligent Engineering Applications and Applied Sciences for Sustainability*, σελίδες 59–76. IGI Global, 2023.

[9] Ashish Thusoo, Raghotham Borthakur, Namit Jain, Zheng Shao, Hao Liu, Suresh Anthony και Joydeep Sen Sarma. *Data Warehousing and Analytics Infrastructure at Facebook*. *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*, σελίδες 1013–1020, 2010.

[10] Abhinandan Banik και Samir Kumar Bandyopadhyay. *Big Data - A Review on Analysing 3Vs*. *Journal of Scientific and Engineering Research*, 3(1), 2016.

[11] M. Aliud din Khan, M. Fahim Uddin και Navarun Gupta. *Seven V's of Big Data: Understanding Big Data to Extract Value*. *Proceedings of the 2014 Zone 1 Confere-*

nce of the American Society for Engineering Education (ASEE Zone 1), σελίδες 1–5, Bridgeport, CT, USA, 2014. IEEE.

[12] Michael Stonebraker. *SQL databases v. NoSQL databases*. *Communications of the ACM*, 53(4):10–11, 2010.

[13] Søren Kejser Jensen, Torben Bach Pedersen και Christian Thomsen. *Time Series Management Systems: A Survey*. *IEEE Transactions on Knowledge and Data Engineering*, ΠΠ(99):1–1, 2017.

[14] Maninder Pal Singh, Mohammad A. Hoque και Sasu Tarkoma. *A survey of systems for massive stream analytics*. *arXiv preprint arXiv:1605.09021*, 2016.

[15] Min Chen, Shiwen Mao και Yunhao Liu. *Big data: A survey*. *Information Sciences*, 275:314–347, 2014.

[16] Martin Kleppmann. *A critique of the CAP theorem*. *arXiv preprint arXiv:1509.05393*, 2015.

[17] Fotios Nikolaidis. *Tromos: a software development kit for virtual storage systems*. Διδακτορική Διατριβή, Université Paris-Saclay (ComUE), 2019.

[18] John Erickson και Keng L Siau. *Service Oriented Architecture: A Research Review from the Software and Applications Perspective*. *Innovations in Information Systems Modeling: Methods and Best Practices*, σελίδες 190–203. IGI Global, 2009.

[19] Neda Niknejad, Waqas Ismail, Irshad Ghani, Behzad Nazari, Maziar Bahari και A. R. B. C. Hussin. *Understanding service-oriented architecture (SOA): A systematic literature review and directions for further investigation*. *Information Systems*, 91:101491, 2020.

[20] Mike P. Papazoglou και Willem-Janvan den Heuvel. *Service-Oriented Architectures: Approaches, Technologies and Research Issues*. *VLDB Journal*, 2007.

[21] Marc-Thomas Schmidt, B Hutchison και P Lambros. *The Enterprise Service Bus: Making Service-Oriented Architecture Real*. *IBM Systems Journal*, 44(4):781–797, 2005.

[22] Omer Aziz, Muhammad Shoaib Farooq, Adnan Abid, Rubab Saher και Naeem Aslam. *Research Trends in Enterprise Service Bus (ESB) Applications: A Systematic Mapping Study*. *IEEE Access*, 8:31180–31197, 2020.

[23] Chellammal Surianarayanan. *Microservices architecture for edge computing environments*. *Recent Advances in Computing, Communication and Information Technology*, τόμος 1537 στο *Advances in Intelligent Systems and Computing*, σελίδες 00–00. Springer, 2022.

[24] Wei Tek Tsai, Xiaoying Bai και Yu Huang. *Software-as-a-service (SaaS): perspectives and challenges*. *Science China Information Sciences*, 57(5):1–15, 2014.

[25] G. Marimuthu. *Research in Service-Oriented Architecture with Cloud and SaaS Solutions*. *International Journal of Computer Science, Engineering and Applications (IJCSEA)*, 3(5), 2019.

[26] Wei Tek Tsai, Qihong Shao, Yu Huang και Xiaoying Bai. *Towards a Scalable and Robust Multi-tenancy SaaS*. *Proceedings of the 2nd Asia-Pacific Symposium on Internetware (Internetware 2010)*, 2010.

[27] Ian Sommerville. *Software Engineering*. Addison-Wesley, 9η έκδοση, 2011.

[28] Adetokunbo A. A. Adenowo και Basirat A. Adenowo. *Software Engineering Methodologies: A Review of the Waterfall Model and Object-Oriented Approach*. *International Journal of Scientific & Engineering Research (IJSER)*, 4(7):427–434, 2013.

[29] Sardar Mudassar Ali Khan. *Iterative Model Used in Software Development*. *Software Requirements Engineering Report (SRE-009)*, 2023.

[30] Ganesh B. Regulwar, Pradip Jawandhiya, Vijay S. Gulhane και R. M. Tugnayat. *Variations in V Model for Software Development*. *International Journal of Advanced Research in Computer Science*, 1(2):133–;, 2010.

[31] Eman A. Altameem. *Impact of Agile Methodology on Software Development*. *Computer and Information Science*, 8(2):9–14, 2015.

[32] Rajendra Ganpatrao Sabale και A. R. Dani. *Comparative Study of Prototype Model For Software Engineering With System Development Life Cycle*. *IOSR Journal of Engineering*, 2(7):21–24, 2012.

[33] Dhruv Doshi, Labdhi Jain και Kunj Gala. *Review of the Spiral Model and Its Applications*. *International Journal of Engineering Applied Sciences and Technology*, 5(12):311–316, 2021.

[34] Farhan Nadim Iqbal. *A Brief Introduction to Application Programming Interface (API)*. ResearchGate / Zenodo, 2023.

[35] Carlos Rodríguez, Marcos Baez, Florian Daniel, Fabio Casati, Juan Carlos Trabucco, Luigi Canali και Gianraffaele Percannella. *REST APIs: A Large-Scale Analysis of Compliance with Principles and Best Practices*. *Web Engineering — 16th International Conference, ICWE 2016, Lugano, Switzerland, June 6-9, 2016, Proceedings*Alessandro Bozzon, Philippe Cudré-Maroux και Cesare Pautasso, επιμελητές, τόμος 9671 στο *Lecture urls in Computer Science*, σελίδες 21–39. Springer International Publishing, 2016.

[36] Srinivas Adilapuram. *Unveiling Modern Authentication Strategies for Java APIs: Exploring OAuth 2.0, JWT, API Keys, and Basic Authentication*. *Journal of Scientific and Engineering Research*, 9(9):119–125, 2022.

[37] Harika Sanugommula. *A Comprehensive Review and Practical Guide to Postman: Revolutionizing API Development, Testing and Collaboration in Modern Software Engineering*. *International Journal of Scientific Research in Engineering and Management*, 2024.

[38] Sardar Mudassar Ali Khan. *How to Implement Swagger in ASP.NET Core Web API*, 2021.

[39] Gregor Hohpe και Bobby Woolf. *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. Addison-Wesley, 2003.

[40] Douglas Vavilapalli, Murthy και Agarwal.. *Apache Hadoop YARN: Yet Another Resource Negotiator*. *Proceedings of the 4th Symposium on Cloud Computing (SOCC)*, 2013.

[41] Matei Zaharia, Reynold S. Xin, Patrick Wendell, Tathagata Das, Michael Armbrust, Ankur Dave, Xiangrui Meng, Josh Rosen, Shivaram Venkataraman, Michael J. Franklin, Ali Ghodsi, Joseph Gonzalez, Scott Shenker και Ion Stoica. *Apache Spark: A Unified Engine for Big Data Processing*. *Communications of the ACM*, 59(11):56–65, 2016.

[42] Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker και Ion Stoica. *Spark: Cluster Computing with Working Sets*. *Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing (HotCloud)*, τόμος 10, σελίδα 95, 2010.

[43] Paris Carbone, Asterios Katsifodimos, Stephan Ewen, Volker Markl, Seif Haridi και Kostas Tzoumas. *Apache Flink: Stream and Batch Processing in a Single Engine*. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering (TCDE)*, 36(4), 2015.

[44] Alexander Alexandrov, Rico Bergmann, Stephan Ewen, Johann Christoph Freytag, Fabian Hueske, Arvid Heise, Odej Kao, Marcus Leich, Ulf Leser, Volker Markl και others. *The Stratosphere Platform for Big Data Analytics*. *The VLDB Journal*, 23(6):939–964, 2014.

[45] F. Gurcan και M. Berigel. *Real-time Processing of Big Data Streams: Lifecycle, Tools, Tasks, and Challenges*. *2018 2nd International Symposium on Multidisciplinary Studies and Innovative Technologies (ISMSIT)*, σελίδες 1–6. IEEE, 2018.

[46] A. Matacuta και C. Popa. *Big Data Analytics: Analysis of Features and Performance of Big Data Ingestion Tools*. *Informatica Economica*, 22(2), 2018.

[47] Ticiana L.Coelho da Silva, Regis Pires Magalhaes, Igo Ramalho Brilhante, Jose A. F. de Macedo, David Araújo, Paulo Rego και Aloisio Vieira Lira Neto. *Big Data Analytics Technologies and Platforms: a Brief Review*. *CEUR Workshop Proceedings*, 2170:4, 2018.