ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΕΠΙΚΟΙΝΩΝΙΩΝ, ΗΛΕΚΤΡΟΝΙΚΗΣ ΚΑΙ ΣΥΣΤΗΜΑΤΩΝ
ΠΛΗΡΟΦΟΡΙΚΗΣ

Έξυπνη διαχείριση μεθόδων API σε περιβάλλον IoT

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Κυριάκος Γ. Παπαβασιλείου

Επιβλέπων: Θεοδώρα Βαρβαρίγου
Καθηγήτρια Ε.Μ.Π.

Αθήνα, Ιούλιος 2019

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΕΠΙΚΟΙΝΩΝΙΩΝ, ΗΛΕΚΤΡΟΝΙΚΗΣ ΚΑΙ ΣΥΣΤΗΜΑΤΩΝ
ΠΛΗΡΟΦΟΡΙΚΗΣ

Έξυπνη διαχείριση μεθόδων API σε περιβάλλον IoT

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Κυριάκος Γ. Παπαβασιλείου

Επιβλέπων: Θεοδώρα Βαρβαρίγου
Καθηγήτρια Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή τον Ιούλιο του 2019.

……..	……..	……..
Θεοδώρα Βαρβαρίγου	Συμεών Παπαβασιλείου	Εμμανουήλ Βαρβαρίγος
Καθηγήτρια Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.

Αθήνα, Ιούλιος 2019

……

Κυριάκος Γ. Παπαβασιλείου
Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Περίληψη

Ο σκοπός και το αντικείμενο της παρούσας διπλωματικής εργασίας είναι ο σχεδιασμός και η ανάπτυξη μιας διαδικτυακής πλατφόρμας στο πλαίσιο του DITAS Project, η οποία να παρέχει στον χρήστη τη δυνατότητα σύνθεσης μεθόδων (Method Composition). Η ιδέα για αυτό το θέμα προέκυψε από την ανάγκη για διαχείριση του τεράστιου όγκου πληροφορίας που είναι πλέον διαθέσιμη στο ευρύ κοινό, εξαιτίας της εμφάνισης, ανάπτυξης και διάδοσης του διαδικτύου και του Διαδικτύου των Πραγμάτων.

Η πλατφόρμα που απασχολεί την εργασία αποτελεί μέρος του DITAS Project, που υλοποιεί την τεχνολογία Δεδομένα ως Υπηρεσία. Ταυτόχρονα, η ίδια η πλατφόρμα υλοποιεί την υπηρεσία Σύνθεσις Μεθόδων.

Για τη δημιουργία της πλατφόρμας χρησιμοποιήθηκαν εργαλεία, βιβλιοθήκες και πλαίσια ανάπτυξης που επιλέγηκαν με βάση τις ανάγκες και τις απαιτήσεις της εργασίας. Πιο συγκεκριμένα, έγινε χρήση της ReactJS, του Node.js και του Node-RED. Οι κανόνες που διέπουν της Σύνθεση Μεθόδων προέκυψαν μέσα από τον τρόπο υλοποίησης της πλατφόρμας, καθώς η βιβλιογραφική ανασκόπηση δεν απέδωσε αποτελέσματα στον τομέα αυτό.

Με την υλοποίηση της πλατφόρμας, ο χρήστης κατέχει πλέον τη δυνατότητα σύνθεσης μεθόδων, ενώ αφήνοντας ανοικτά περιθώρια ανάπτυξης και εξέλιξης, παρέχονται ακόμα περισσότερες δυνατότητες στον τομέα αυτό.

Λέξεις Κλειδιά:
Σύνθεση Μεθόδων, DITAS Project, IoT, API, VDC, React, Node.js, Node-RED

Abstract

The object of this diploma thesis is the design and development of an online platform within the DITAS Project, which gives the user the ability to synthesize methods (Method Composition). The idea for this issue has emerged from the need to manage the huge amount of information that is now available to the general public due to the emergence, development and spread of the Internet and the Internet of Things.

The work platform is part of the DITAS Project, which provides Data as a Service (DaaS). At the same time, the platform itself provides the ability for method composition.

The platform was developed using tools, libraries and frameworks that were selected on the basis of the needs and requirements of the project. More specifically, ReactJS, Node.js and Node-RED were used. The rules governing the Method Composition have emerged through the implementation of the platform, as the bibliographic review has not yielded any results in this area.

With the implementation of the platform, the user now has the ability to synthesize methods, while leaving room for growth and development, also even more possibilities in this area are provided.

Keywords:

Method Composition, DITAS Project, IoT, API, VDC, React, Node.js, Node-RED

Ευχαριστίες

Με την παρούσα διπλωματική εργασία, ολοκληρώνεται η ακαδημαϊκή μου πορεία στη Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών του Εθνικού Μετσόβιου Πολυτεχνείου, η οποία δε θα μπορούσε να διεκπεραιωθεί χωρίς τη βοήθεια των ανθρώπων με τους οποίους συνεργάστηκα και οι οποίοι με στήριξαν σε αυτό το έργο.

Αρχικά θα ήθελα να ευχαριστήσω την κα. Θεοδώρα Βαρβαρίγου, καθηγήτρια Ε.Μ.Π για την εμπιστοσύνη που μου έδειξε και την ανάθεση μίας πολύ ενδιαφέρουσας διπλωματικής εργασίας, καθώς και τον κ. Αχιλλέα Μαρινάκη, υποψήφιο Διδάκτορα Ε.Μ.Π για τη βοήθεια και τον χρόνο που διέθεσαν καθοδηγώντας με σε όλα τα στάδια της εργασίας.

Τέλος, θα ήθελα να ευχαριστήσω την οικογένεια μου για την όλη στήριξη κατά τη διάρκεια της πενταετούς μου φοίτησης στο πανεπιστήμιο, καθώς και τους φίλους και τους συμφοιτητές μου, που ήταν παρόντες τόσο στα ωραία όσο και στα δύσκολα αυτής της πορείας.

Πίνακας Περιεχομένων

Πίνακας Εικόνων

Κεφάλαιο 1 Εισαγωγή

Η σημερινή εποχή χαρακτηρίζεται ως η εποχή της πληροφορίας. Κι αυτό γιατί οι πληροφορίες μεταφέρονται πλέον με τεράστια ευκολία και ταχύτητα και σε απίστευτα μεγάλο εύρος. Η εξέλιξη αυτή είναι αδιαμφισβήτητα απόρροια της ανακάλυψης, εξέλιξης και διάδοσης του Internet. Η «δημιουργία» του στα τέλη της δεκαετίας του 1980 και η ευρεία υιοθέτησή του κατά τη δεκαετία του 1990 οδήγησε τον κόσμο σε μία εποχή όπου η οικονομία του πλανήτη βασίζεται στην Τεχνολογία της Πληροφορίας (Information Technology – IT), τομέας ο οποίος έχει ως κύριο αντικείμενο ενασχόλησης την αποθήκευση, τη μεταφορά και την επεξεργασία της πληροφορίας.

Τεράστια ανάπτυξη έχει επίσης παρατηρηθεί τα τελευταία χρόνια στον τομέα του Διαδικτύου των Πραγμάτων (Internet of Things – IoT), το οποίο ορίζεται ως μια υποδομή που επεκτείνει το Internet διασυνδέοντας συσκευές είτε φυσικές είτε εικονικές με μόνη προϋπόθεση τη δυνατότητα να ταυτοποιήσουν τον εαυτό τους στο δίκτυο. Στον σύγχρονο κόσμο, τέτοιες συσκευές μπορεί να αποτελούν οποιαδήποτε αντικείμενα καθημερινής χρήσης, όπως κινητά τηλέφωνα και έξυπνες οικιακές συσκευές, αλλά ακόμα και πιο σύνθετα μηχανήματα, όπως εξειδικευμένοι αισθητήρες με εφαρμογές στον τομέα της βιομηχανίας ή της υγείας. Αυτός ο νέος τρόπος αξιοποίησης του διαδικτύου προκάλεσε μια ακόμα πιο τεράστια αύξηση της διαθέσιμης πληροφορίας. Αυτό ακριβώς το φαινόμενο καλούμαστε να δαμάσουμε. Κάτι τέτοιο βέβαια δε σημαίνει να το αναχαιτίσουμε ή να το περιορίσουμε, αλλά να το ελέγξουμε και να το διαχειριστούμε με ωφέλιμο προς την κοινωνία τρόπο.

Ένα μέσο με το οποίο μπορεί να τύχει διαχείρισης αυτή η πληροφορία είναι το DITAS Project[1]. Το DITAS Project είναι μια Cloud και Edge Computing πλατφόρμα που σκοπό έχει να αντιμετωπίσει την πολυπλοκότητα ανάπτυξης εφαρμογών με μεγάλη ανάγκη δεδομένων (Data Intensive Applications)[2]. Η τακτική που ακολουθείται για την επίτευξη αυτού του στόχου είναι η εξής: η διαχείριση δεδομένων απομακρύνεται από τον τελικό τους χρήστη, δηλαδή τον προγραμματιστή, προσφέροντάς του τη δυνατότητα να καθορίσει απλώς τις απαιτήσεις ως προς τα δεδομένα. Την ίδια στιγμή, το DITAS αναλαμβάνει την ευθύνη για την παροχή τους, αποκρύπτοντας τη διαδικασία συλλογής και αρχικής επεξεργασίας τους[2].

1.1 Αντικείμενο και Σκοπός Διπλωματικής Εργασίας

Ο σκοπός και το αντικείμενο της παρούσας διπλωματικής εργασίας είναι ο σχεδιασμός και η ανάπτυξη μιας διαδικτυακής πλατφόρμας στο πλαίσιο του DITAS Project, η οποία να παρέχει στον χρήστη τη δυνατότητα σύνθεσης μεθόδων (Method Composition). Αυτό σημαίνει ότι μέσω της πλατφόρμας, ο προγραμματιστής/σχεδιαστής της εκάστοτε εφαρμογής θα είναι σε θέση να προσαρμόσει το Virtual Data Container (VDC) που έχει λάβει από το DITAS Platform στις ακριβείς ανάγκες της εφαρμογής του, επιτυγχάνοντας μείωση της άχρηστης πληροφορίας που μεταφέρεται καθώς και μείωση της επεξεργασίας που θα χρειάζονται τα δεδομένα μετά τη λήψη τους. Όσον αφορά στο VDC, αυτό θα οριστεί και θα επεξηγηθεί αναλυτικά σε επόμενο κεφάλαιο.

1.2 Οργάνωση Κειμένου

Η παρούσα διπλωματική εργασία αποτελείται από 6 κεφάλαια, συμπεριλαμβανομένου του παρόντος και πρώτου κεφαλαίου που παρουσιάζει το αντικείμενο, τον σκοπό και την οργάνωση της εργασίας. Το δεύτερο κεφάλαιο αναφέρεται στο θεωρητικό πλαίσιο στο οποίο βασίζεται η διαδικτυακή πλατφόρμα, καθώς και στις τεχνολογίες που χρησιμοποιήθηκαν. Στο τρίτο κεφάλαιο σκιαγραφείται η αρχιτεκτονική του συστήματος, ενώ αναλύονται και συγκρίνονται τα διαθέσιμα εργαλεία που μπορούν να χρησιμοποιηθούν για την ανάπτυξη βασικών τμημάτων του συστήματος. Το τέταρτο κεφάλαιο εμπεριέχει την υλοποίηση βασικών κομματιών του συστήματος και παραθέτει ένα παράδειγμα λειτουργίας του. Στο πέμπτο κεφάλαιο γίνεται μια σύνοψη και συναγωγή των συμπερασμάτων της εργασίας, ενώ προτείνονται μελλοντικές επεκτάσεις. Στο έκτο και τελευταίο κεφάλαιο της διπλωματικής εργασίας αναγράφεται η βιβλιογραφία.

Κεφάλαιο 2 Θεωρητικό Πλαίσιο – Τεχνολογίες

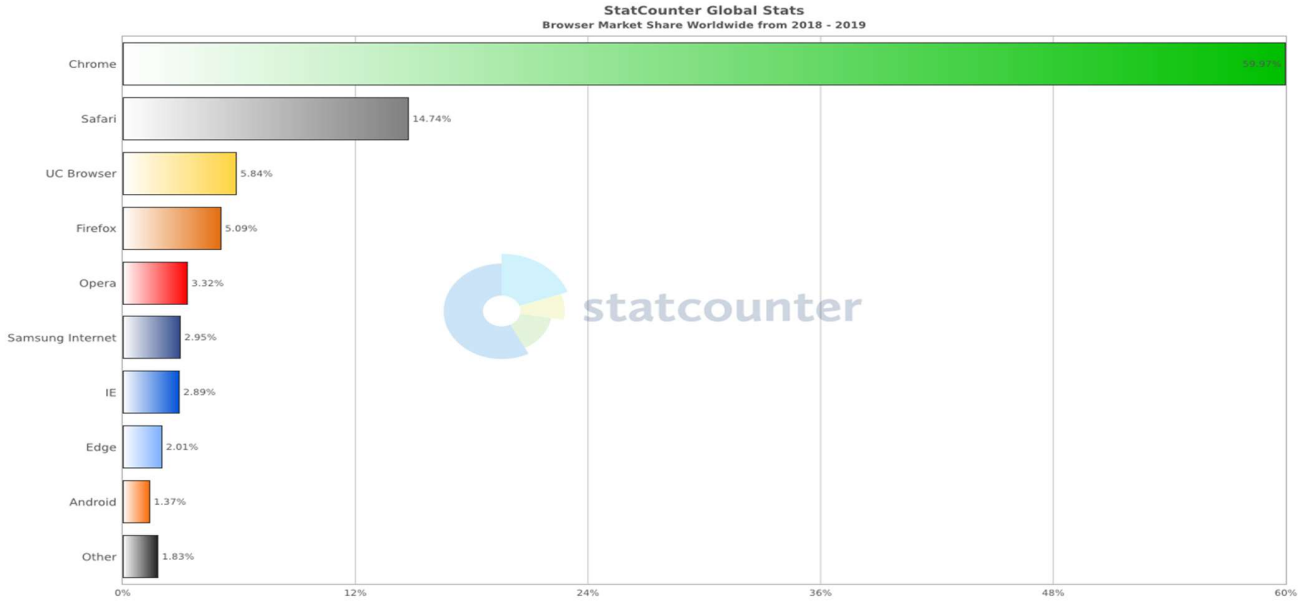
Στο δεύτερο κεφάλαιο της διπλωματικής εργασίας παρουσιάζεται το θεωρητικό πλαίσιο, πάνω στο οποίο βασίζεται η διαδικτυακή πλατφόρμα που αποτελεί το αντικείμενό μας, όπως αυτή εξηγήθηκε στο προηγούμενο κεφάλαιο. Γίνεται επίσης αναφορά στις τεχνολογίες που χρησιμοποιήθηκαν και πρόκειται είτε για υπάρχουσες τεχνολογίες είτε για τεχνολογίες που αναπτύχθηκαν στο πλαίσιο του DITAS Project. Αρχικά, θα παρουσιαστούν βασικές τεχνολογίες πάνω στις οποίες βασίζεται η ομαλή λειτουργία του διαδικτύου στις μέρες μας και στη συνέχεια θα γίνει αναφορά σε πιο εξειδικευμένες τεχνολογίες, όπως το REST API και το API specification, οι οποίες όμως ακόμα βρίσκουν ευρεία εφαρμογή σε όλους τους τομείς του διαδικτύου. Τέλος, θα παρουσιαστούν τεχνολογίες εξειδικευμένες στο DITAS Project και στη συγκεκριμένη διπλωματική εργασία, όπως το VDC και το Method Composition.

2.1 Τεχνολογίες Διαδικτύου

Με τον όρο τεχνολογίες διαδικτύου αναφερόμαστε στις τεχνολογίες οι οποίες καταστούν δυνατή την καθημερινή χρήση του διαδικτύου και τεχνολογίες οι οποίες βρίσκονται πίσω από όλους τους ιστότοπους, όπως η γλώσσα σήμανσης HTML.

2.1.1 Περιηγητής Ιστού

Ο περιηγητής ιστού (Web Browser) αποτελεί ένα από τα πιο χαρακτηριστικά παραδείγματα τεχνολογίας διαδικτύου. Πιο συγκεκριμένα, είναι μια εφαρμογή η οποία επιτρέπει στον χρήστη να έχει πρόσβαση και να βλέπει ιστοσελίδες και ιστότοπους στον Παγκόσμιο Ιστό. Μερικά παραδείγματα περιηγητών ιστού είναι ο Microsoft Edge, το Google Chrome, το Mozzila Firefox και το Safari. Στην Εικόνα 1 φαίνεται το μερίδιο αγοράς για τους πιο δημοφιλείς περιηγητές ιστού τον τελευταίο χρόνο.



Εικόνα 1: Μερίδιο αγοράς περιηγητών ιστού παγκόσμια 2018-19[1]

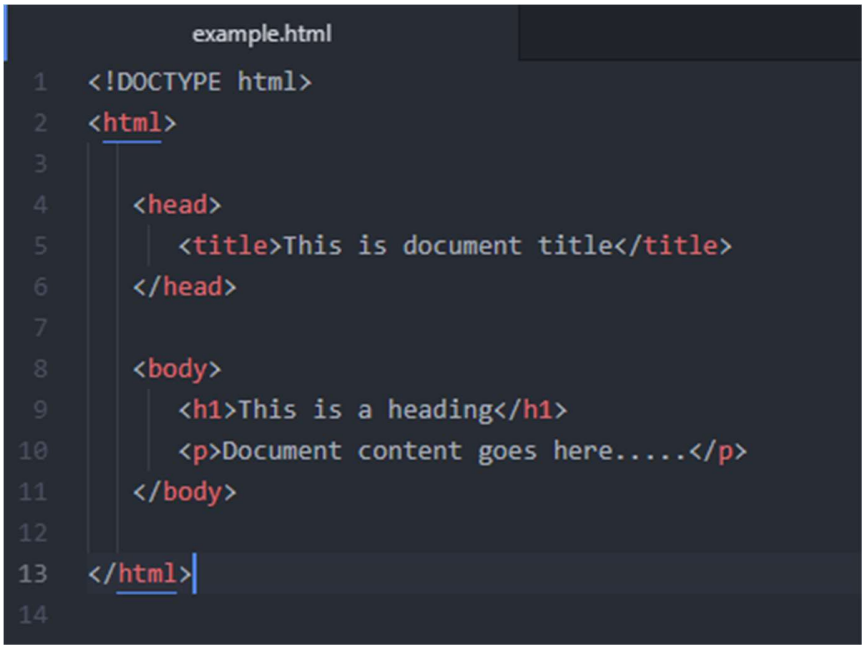
Η κύρια λειτουργία ενός περιηγητή ιστού είναι η απόδοση κώδικα σήμανσης HTML. Κάθε φορά που φορτώνει μια ιστοσελίδα, ο περιηγητής ιστού επεξεργάζεται τον κώδικα HTML και όλα τα στοιχεία που περιέχει (τα οποία περιγράφονται αναλυτικά στη συνέχεια) και αποδίδει το αποτέλεσμα στην οθόνη του χρήστη[3].

2.1.2 HTML

Το HTML είναι το αρκτικόλεξο του HyperText Markup Language, που στα ελληνικά σημαίνει Γλώσσα Σήμανσης Υπερκειμένου. Η HTML είναι η κύρια γλώσσα σήμανσης που χρησιμοποιείται στις μέρες μας για την απόδοση ιστοσελίδων σε περιηγητές ιστού. Πιο συγκεκριμένα, οι ιστοσελίδες αποτελούνται από HTML, η οποία χρησιμοποιείται σε συνδυασμό με έναν περιηγητή ιστού για να εμφανίσει κείμενο, εικόνες, βίντεο και άλλους πόρους[4]. Η πιο πρόσφατη έκδοση της HTML είναι η HTML5.

[1] http://gs.statcounter.com/browser-market-share#yearly-2018-2019-bar

Τα αρχεία της HTML είναι αρχεία απλού κειμένου σε μορφή κατανοητή από τον άνθρωπο και δε μεταγλωττίζονται σε κώδικα μηχανής. Τα αρχεία αυτά έχουν κατάληξη .html ή .htm[4] και η δομή τους αποτελείται από στοιχεία HTML, όπως φαίνεται στην Εικόνα 2.



Εικόνα 2: Παράδειγμα Αρχείου HTML

Με τον όρο «στοιχεία HTML» αναφερόμαστε στα δομικά στοιχεία ενός εγγράφου HTML. Τα στοιχεία αυτά χαρακτηρίζονται από μια ετικέτα έναρξης και μία ετικέτα τέλους (αν και υπάρχουν στοιχεία τα οποία έχουν μόνο ετικέτα έναρξης). Τα βασικά στοιχεία HTML είναι καταρχάς το ίδιο το αρχείο HTML και υποδηλώνεται με τις ετικέτες <html> </html>, μέσα στις οποίες περιέχονται όλα τα υπόλοιπα στοιχεία. Στη συνέχεια έχουμε το στοιχείο <head> </head>, μέσα στο οποίο περιέχονται διάφορα στοιχεία για την ιστοσελίδα όπως ο τίτλος, τα στυλ μορφοποίησης, ο κώδικας σεναρίων και οι μετα-πληροφορίες. Τέλος, έχουμε το στοιχείο <body> </body>, το οποίο περιέχει το ορατό στον χρήστη τμήμα της ιστοσελίδας. Αυτά είναι μερικά μόνο παραδείγματα των στοιχείων HTML. Στην πραγματικότητα, αυτά είναι πολύ περισσότερα και κάθε ένα έχει τον δικό του ρόλο και τη δική του ερμηνεία[5].

2.1.3 CSS

Το CSS (Cascading Style Sheets), αντιστοιχεί στον ελληνικό όρο Διαδοχικά Φύλλα Ύφους και πρόκειται για μια γλώσσα η οποία χρησιμοποιείται για τον έλεγχο και την περιγραφή της παρουσίασης ιστοσελίδων. Χρησιμοποιείται σε συνδυασμό με γλώσσες σήμανσης όπως η HTML, αλλά είναι ανεξάρτητη από κάποια συγκεκριμένη γλώσσα και μπορεί να χρησιμοποιηθεί με οποιαδήποτε γλώσσα σήμανσης βασισμένη σε Extensible Markup Language (XML)[6]. Τα αρχεία CSS έχουν δομή όπως φαίνεται στην Εικόνα 3 και έχουν κατάληξη .css.

example.css

```
1  body {
2    background-color: powderblue;
3  }
4  h1 {
5    color: blue;
6  }
7  p {
8    color: red;
9  }
10
```

Εικόνα 3: Παράδειγμα αρχείου CSS

Η CSS μπορεί να χρησιμοποιηθεί για να περιγράψει τα στιλιστικά στοιχεία μιας ιστοσελίδας όπως τα χρώματα, τις γραμματοσειρές και τη διαμόρφωση της σελίδας, αλλά και για να προσθέσει δυναμικά στοιχεία. Χρησιμοποιείται επίσης για να εφαρμόσει κινούμενες εικόνες (animations) σε μία ιστοσελίδα[7]. Ακόμα, μπορεί να αλλάξει τον τρόπο που εμφανίζεται μια σελίδα ανάλογα με το μέγεθος της οθόνης και το είδος της συσκευής[6].

2.1.4 JavaScript

Η JavaScript είναι μια διερμηνευμένη γλώσσα προγραμματισμού, η οποία συμμορφώνεται με την προδιαγραφή ECMAScript. Είναι ευρέως γνωστή ως γλώσσα σεναρίων (Scripting Language) και χρησιμοποιείται σε ιστοσελίδες σε συνδυασμό με γλώσσες σήμανσης, ώστε να προσφέρει δυναμικό περιεχόμενο στους χρήστες[8]. Μαζί με την HTML και τη CSS αποτελούν τον βασικό πυρήνα του Παγκόσμιου Ιστού. Έχει όμως χρήση και εκτός των περιηγητών ιστού, όπως για παράδειγμα τα έγραφα PDF, και ανάπτυξη εφαρμογών ιστού από την πλευρά του διακομιστή (server-side programming).

Η JavaScript είναι μια γλώσσα σεναρίων βασισμένη σε πρωτότυπα (prototype-based) με ασθενείς τύπους. Χαρακτηρίζεται ως γλώσσα πολλαπλών παραδειγμάτων (multi-paradigm) καθώς υποστηρίζει διάφορα στιλ προγραμματισμού (αντικειμενοστραφές, προστακτικό και συναρτησιακό). Η βασική σύνταξη της JavaScript είναι παρόμοια με τις γλώσσες Java και C++, έχει όμως σημαντικές διαφορές στη λειτουργία της[9].

2.1.5 JSON

Το JSON (JavaScript Object Notation) είναι ένας ελαφρύς μορφότυπος, ο οποίος χρησιμοποιείται για την ανταλλαγή δεδομένων. Έχει μορφή η οποία είναι κατανοητή στον άνθρωπο και είναι βασισμένο στο ECMAScript. Το JSON είναι ανεξάρτητο από γλώσσα προγραμματισμού, αλλά χρησιμοποιεί συμβάσεις οι οποίες είναι κοινές σε πολλές γλώσσες προγραμματισμού καθιστώντας το ως ένα από τα ιδανικότερα μέσα ανταλλαγής δεδομένων[10].

Το JSON αποτελείται από δύο βασικές δομές δεδομένων: τα αντικείμενα (Objects) και τους πίνακες (Arrays). Τα αντικείμενα είναι μη ταξινομημένα σύνολα αποτελούμενα από ζεύγη ονομάτων και τιμών, ενώ οι πίνακες είναι σύνολα από τιμές. Μια τιμή μπορεί να είναι είτε κείμενο (σειρά από χαρακτήρες) είτε αριθμός είτε τιμή Boolean. Μπορεί ακόμα να είναι με τη σειρά της αντικείμενο ή πίνακας[10].

2.2 REST API

Όπως υποδεικνύει και το όνομά του, το REST API (Representational State Transfer Application Program Interface) είναι ένα API το οποίο συμμορφώνεται με το μοντέλο REST. Το μοντέλο REST ορίστηκε από τον Roy Thomas Fielding και καθορίζει μια σειρά από κανόνες για την ανάπτυξη υπηρεσιών ιστού (Web Services) [11]. Οι κανόνες αυτοί αναφέρονται συνοπτικά πιο κάτω:

1. Μοντέλο πελάτη-Διακομιστή: η υπηρεσία REST τρέχει σε έναν κεντρικό διακομιστή και οι πελάτες/χρήστες αιτούνται πόρους από αυτή.
2. Stateless: ο διακομιστής δε διατηρεί κάποια πληροφορία ως προς την κατάσταση της σύνδεσης, αλλά επεξεργάζεται κάθε αίτηση το ίδιο.
3. Cacheable responses: τα δεδομένα πρέπει να δηλώνονται είτε άμεσα είτε έμμεσα ως προς τη δυνατότητά τους να αποθηκευτούν. Στην περίπτωση που διαθέτουν τη δυνατότητα αυτή, τότε ο χρήστης μπορεί να τα ξαναχρησιμοποιήσει μέσα σε κάποιο ορισμένο χρονικό διάστημα χωρίς να ξανακάνει αίτημα στον διακομιστή.
4. Ομοιόμορφη διασύνδεση: όλες οι υπηρεσίες REST καλούνται να είναι να ομοιόμορφες ως προς τις μεθόδους τους και τον τρόπο χρήσης τους αποκρύπτοντας έτσι τον τρόπο υλοποίησής τους από τον πελάτη/χρήστη.
5. Layered system: ο χρήστης της υπηρεσίας δεν πρέπει να έχει την ικανότητα να καταλάβει αν είναι συνδεδεμένος στον κεντρικό διακομιστή ή σε κάποιον ενδιάμεσο.
6. Code-on-demand (Προαιρετικό): Ο διακομιστής μπορεί να αλλάξει ή να επεκτείνει τη λειτουργικότητα του πελάτη μεταφέροντας σε αυτόν έναν εκτελέσιμο κώδικα.

Τα REST APIs, όπως και όλα τα συστήματα που βασίζονται στο μοντέλο REST, χαρακτηρίζονται από υψηλή απόδοση, αξιοπιστία και δυνατότητα κλιμάκωσης ακόμα και κατά τη διάρκεια τις εκτέλεσης. Τα REST APIs, ως πρωτόκολλο επικοινωνίας, χρησιμοποιούν το πρωτόκολλο HTTP και τις μεθόδους που αυτό υποστηρίζει (GET, POST, PUT, PATCH, DELETE).

2.3 API Specification

Η ευρεία χρήση των APIs είτε REST είτε άλλου είδους έχει οδηγήσει στην ανάγκη για έναν επίσημο και τυποποιημένο τρόπο περιγραφής τους, τόσο σε μορφή κατανοητή από τον άνθρωπο όσο και σε μορφή κατανοητή από τον υπολογιστή. Παρακάτω παρουσιάζονται οι τρεις πιο διαδομένοι τρόποι περιγραφής, καθώς και μια σύντομη επεξήγηση για τον κάθε ένα.

2.3.1 OpenApi Specification

Το OpenApi Specification (OAS), προηγουμένως γνωστό και ως Swagger Specification, είναι μια ανοικτή προδιαγραφή η οποία δημιουργήθηκε στα πλαίσια του OpenAPI Initiative του Linux Foundation.

Η OAS καθορίζει μια προδιαγραφή για την τυπική περιγραφή REST APIs, αγνωστική τόσο ως προς την γλώσσα προγραμματισμού που χρησιμοποιήθηκε για την ανάπτυξη του API (το οποίο είναι επόμενο του μοντέλου REST), αλλά και ως προς τη γλώσσα προγραμματισμού που θα χρησιμοποιήσει την περιγραφή. Έχει μορφή η οποία είναι κατανοητή από τον άνθρωπο αλλά και από τον υπολογιστή.

Το γεγονός ότι η περιγραφή είναι φτιαγμένη σε μορφή που να μπορεί εύκολα να κατανοηθεί από τον υπολογιστή μας δίνει νέες δυνατότητες ως προς τη χρήση της. Μπορούμε για παράδειγμα να προβούμε σε αυτοματοποιημένη δημιουργία κώδικα για έλεγχο της υπηρεσίας. Τα έγραφα OAS τα οποία περιγράφουν REST APIs μπορεί να είναι σε μορφή JSON ή σε μορφή YAML. Ένα ενδεικτικό παράδειγμα για την κάθε μορφή φαίνεται στην Εικόνα 4[12].

OpenApiExample.yaml

```yaml
openapi: "3.0.2"
info:
  version: 1.0.0
  title: Petstore
servers:
  - url: server url
paths:
  /pets:
    get:
      summary: List all pets
      operationId: listPets
      responses:
        '200':
          description: A paged array of pets
          content:
            application/json:
              schema:
                type: array
                items:
                  required:
                  - id
                  - name
                properties:
                  id:
                    type: integer
                    format: int64
                  name:
                    type: string
```

OpenApiExample.json

```json
{
  "openapi": "3.0.2",
  "info": {
    "version": "1.0.0",
    "title": "Petstore"
  },
  "servers": [
    {
      "url": "server url"
    }
  ],
  "paths": {
    "/pets": {
      "get": {
        "summary": "List all pets",
        "operationId": "listPets",
        "responses": {
          "200": {
            "description": "A paged array of pets",
            "content": {
              "application/json": {
                "schema": {
                  "type": "array",
                  "items": {
                    "required": [
                      "id",
                      "name"
                    ]
```

Εικόνα 4: Παράδειγμα OpenApi Specification

2.3.2 RAML

Το RAML (RESTful API Modelling Language) είναι μια γλώσσα βασισμένη στο YAML για την περιγραφή API βασισμένων στο πρωτόκολλο HTTP και υπακούει είτε στις περισσότερες είτε σε όλες τις αρχές του μοντέλου REST.

Το RAML όπως και η OAS έχει μορφή η οποία είναι κατανοητή τόσο από τον άνθρωπο όσο και από τον υπολογιστή με αποτέλεσμα και αυτή η προδιαγραφή να μας προσφέρει τα πλεονεκτήματα της OAS. Το RAML προσφέρει τρόπους μείωσης της επανάληψης στις περιγραφές API υπό τη μορφή Resource Types και Traits. Τα αρχεία RAML έχουν μορφή YAML με κατάληξη .raml. Ένα τυπικό αρχείο RAML φαίνεται στην Εικόνα 5[13].

ramlExample.raml

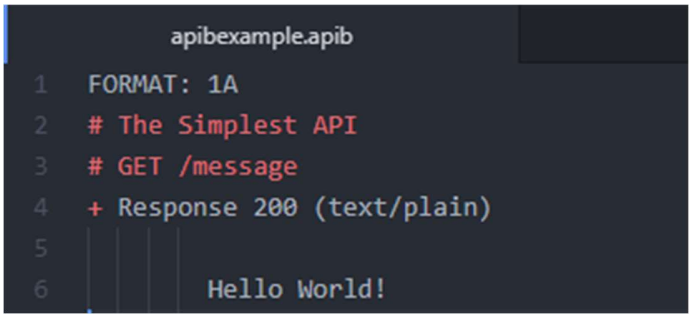
```
#%RAML 1.0
title: Hello world

/greeting:
  get:
    responses:
      200:
        body:
          application/json:
            type: object
            properties:
              message: string
            example:
              message: "Hello world"
```

Εικόνα 5: Παράδειγμα RAML

2.3.3 API Blueprint

Η API Blueprint είναι μια υψηλού επιπέδου γλώσσα βασισμένη στη Markdown για την περιγραφή Web APIs. Έχει φτιαχτεί με τη φιλοσοφία design-first, δηλαδή έχει φτιαχτεί με σκοπό να λειτουργεί ως ένα συμβόλαιο μεταξύ του δημιουργού και του χρήστη του API.

Όπως και οι δύο προηγούμενες περιγραφές, έτσι και το API Blueprint, είναι γραμμένο σε μορφή κατανοητή από τον άνθρωπο και τον υπολογιστή. Σε αντίθεση με τις άλλες δύο, δε χρησιμοποιεί αρχεία τύπου JSON ή YAML, άλλα χρησιμοποίει αρχεία βασισμένα στη γλώσσα Markdown με κατάληξη .apib, όπως φαίνεται στην Εικόνα 6[14].



Εικόνα 6: Παράδειγμα API Blueprint

2.4 DaaS

Το DaaS αποτελεί το αρκτικόλεξο για το Data as a Service, που αποδίδεται στα ελληνικά με τον όρο Δεδομένα ως Υπηρεσία. Το DaaS ανήκει στην οικογένεια τεχνολογιών aaS (as a Service), δηλαδή των τεχνολογιών εκείνων που σκοπό έχουν να προσφέρουν στον πελάτη ένα προϊόν υπό τη μορφή υπηρεσίας. Πιο αναλυτικά, πρόκειται για τις τεχνολογίες που προσφέρουν ένα προϊόν στον πελάτη χωρίς αυτός να χρειάζεται να ασχοληθεί με τον τρόπο υλοποίησής του.

Τα δεδομένα αποδεικνύονται ως ένα από τα ισχυρότερα εργαλεία στη σύγχρονη οικονομία, καθώς χρησιμοποιούνται για σκοπούς όπως η εκτίμηση ρίσκου, η βελτίωση απόδοσης του συστήματος και η περικοπή εξόδων[15]. Για να μπορεί να γίνει όμως αυτό, τα δεδομένα πρέπει να περάσουν από κάποια στάδια, τις περισσότερες φορές κοινά μεταξύ τους. Αυτά είναι η συλλογή των δεδομένων, ο καθαρισμός τους και η εξαγωγή των πληροφοριών[15]. Η ανάπτυξη όμως διαφορετικών συστημάτων για τη διαχείριση των δεδομένων για κάθε οργανισμό είναι δαπανηρή και χρονοβόρα. Το πρόβλημα αυτό λύνει το DaaS αναλαμβάνοντας την όλη διαδικασία και παραδίδοντας στον τελικό χρήστη μόνο τα δεδομένα χωρίς να χρειάζεται ο ίδιος να ξέρει ούτε από πού ακριβώς προέρχονται ούτε τη διαδικασία που ακολουθήθηκε για να φτάσουν σε αυτόν.

Το DITAS Project υλοποιεί μια υπηρεσία DaaS. Παρέχει δηλαδή δεδομένα υπό τη μορφή υπηρεσίας δίνοντας συγκεκριμένες διαβεβαιώσεις ως προς τη διαθεσιμότητα και την ποιότητά τους. Για να το επιτύχει αυτό χρησιμοποιεί εικονικά κιβώτια δεδομένων (VDC – Virtual Data Container). Οι βασικές ορολογίες περιγράφονται στη συνέχεια.

2.4.1 VDC Blueprint

Το VDC Blueprint είναι ένα αρχείο τύπου JSON το οποίο δημιουργεί ο διαχειριστής των δεδομένων και δημοσιοποιεί στο DITAS. Το αρχείο αυτό περιέχει μια πλήρη περιγραφή των δεδομένων χωρισμένη σε πέντε ενότητες:

1. Internal Structure: Στην ενότητα αυτή δίνεται μια περιγραφή των πηγών των δεδομένων τις οποίες εκπροσωπεί το VDC Blueprint (όνομα, τύπος, σχήμα) καθώς και πληροφορίες για το πώς μπορεί να γίνει ανάκτηση αυτών των πληροφοριών.
2. Data Management: Αυτή η ενότητα περιέχει διάφορες μετρικές, οι οποίες χαρακτηρίζουν τα δεδομένα (reliability, completeness) κάτω από το πεδίο Data Utility. Επιπρόσθετα, κάτω από το πεδίο Security υπάρχουν μέτρα ασφαλείας για τα συγκεκριμένα δεδομένα, όπως το ποιοι έχουν πρόσβαση σε αυτά ή ποια πρωτόκολλα κρυπτογράφησης πρέπει να χρησιμοποιηθούν.
3. Abstract Properties: Στη συγκεκριμένη ενότητα βρίσκονται όλες οι επιχειρηματικές και τεχνικές ιδιότητες του VDC, οι οποίες θα χρησιμοποιηθούν για τη διαφήμιση του.
4. CookBook Appendix: Η ενότητα αυτή περιέχει όλες της απαραίτητες πληροφορίες για να τρέξει το συγκεκριμένο VDC, όπως για παράδειγμα τις απαραίτητες βιβλιοθήκες ή τα πακέτα που πρέπει να εγκατασταθούν στο σύστημα.
5. Exposed API: Αυτή η ενότητα δίνει μια πλήρη περιγραφή σε OpenApi Specification του API που παρουσιάζει το συγκεκριμένο VDC[16][17].

Στο Παράρτημα Γ παρατίθεται το VDC Blueprint Schema το οποίο περιγράφει πλήρως τη δομή του VDC Blueprint.

2.4.2 VDC

Όταν ο χρήστης του DITAS έχει επιλέξει (ή έχει επεξεργαστεί) το VDC Blueprint που του ταιριάζει και συμφωνηθούν οι απαιτήσεις, τότε υλοποιείται το VDC. Το VDC είναι η υλοποίηση του VDC Blueprint, είναι δηλαδή η εφαρμογή που έχει την αρμοδιότητα να μαζέψει τα δεδομένα από τις διάφορες πηγές που περιγράφονται στο VDC Blueprint και να τα παρουσιάσει στον χρήστη μέσω του API που παρουσιάζεται στην τελευταία ενότητα του VDC Blueprint[17].

Το VDC μπορεί να μετακινείται από το DITAS ανάλογα με τις ανάγκες του χρήστη. Για παράδειγμα, σε περίπτωση που βρίσκεται σε διακομιστή εκτός του Cloud και δεν επιτυγχάνονται τα συμφωνημένα επίπεδα διαθεσιμότητας, τότε το VDC μπορεί να μετακινηθεί στο Cloud. Το VDC έχει την ευθύνη αυτή η αλλαγή να μην είναι φανερή στον τελικό χρήστη, αλλά αυτός να μπορεί να δει μόνο την αύξηση της διαθεσιμότητας[17].

2.5 Method Composition

Με τον όρο Method Composition αναφερόμαστε στην αναδιάρθρωση των μεθόδων ενός API, δηλαδή αναφερόμαστε είτε στην επιλογή συγκεκριμένων μεθόδων από το αρχικό API και δημιουργία ενός νέου με μόνο την επιλεγμένες μεθόδους, είτε την εξολοκλήρου δημιουργία νέων μεθόδων επιλέγοντας απαντήσεις από τις υπάρχουσες μεθόδους ακολουθώντας ένα σύνολο κανόνων και δημιουργία ενός νέου API το οποίο υποστηρίζει μόνο τις καινούργιες μεθόδους.

Στα πλαίσια του DITAS project στην παρούσα διπλωματική εργασία υλοποιείται Method Composition το οποίο υποστηρίζει πλήρως και τις δύο μεθόδους οι οποίες προαναφέρθηκαν. Οι δύο μέθοδοι θα υποστηρίζονται και συνδυαστικά, δηλαδή ο χρήστης θα έχει την δυνατότητα να επιλέξει συγκεκριμένες μεθόδους από το αρχικό API αλλά θα έχει και την δυνατότητα να δημιουργήσει και δικές του.

Στην συγκεκριμένη υλοποίηση το σύστημα λαμβάνει την περιγραφή ενός υπάρχοντος API η οποία πρέπει να ακολουθεί το OpenAPI specification v3.0.2 και παρουσιάζει γραφικά στον χρήστη της εφαρμογής της υπάρχουσες μεθόδους. Στο χρήστη δίνεται η επιλογή να επιλέξει είτε ολόκληρες μεθόδους είτε να φτιάξει δικές του επιλέγοντας απαντήσεις από της υπάρχουσες. Στην συνέχεια το σύστημα δημιουργεί ένα νέο API το οποίο είναι άμεσα λειτουργικό καθώς και μια περιγραφή για το νέο API η οποία ακολουθεί το OpenAPI specification. Αναλυτική περιγραφή τις υλοποίησης του συστήματος αλλά και παράδειγμα χρήσης παρουσιάζονται στα επόμενα κεφάλαια.

Κεφάλαιο 3 Επιλογή Εργαλείων

Στο παρόν κεφάλαιο γίνεται μία αναφορά αρχικά στην αρχιτεκτονική του συστήματος και στη συνέχεια στα εργαλεία που υπάρχουν διαθέσιμα στην αγορά και μπορούν να χρησιμοποιηθούν στην ανάπτυξη των βασικών τμημάτων του συστήματος. Τέλος, γίνεται μία σύγκριση ανάμεσα στα υπάρχοντα εργαλεία για κάθε τμήμα και επιλέγεται αυτό που ανταποκρίνεται καλύτερα στις ανάγκες του συστήματος.

Το σύστημα χωρίζεται σε τρία βασικά τμήματα: το Front-End, ένα REST API το οποίο αναλαμβάνει τον ρόλο του Back-End και έναν διακομιστή στον οποίο θα τρέχουν τα νέα API που δημιουργεί το σύστημα. Το Front-End είναι το γραφικό περιβάλλον το οποίο παίρνει την περιγραφή ενός API και παρουσιάζει τις μεθόδους γραφικά στον χρήστη, ώστε να επιλέξει ή να δημιουργήσει τις μεθόδους του νέου API. Αυτό το περιβάλλον θα έχει τη μορφή ιστοσελίδας. Το Back-End θα έχει την μορφή ενός REST API και θα είναι υπεύθυνο να λαμβάνει τις επιλογές του χρήστη από το Front-End και να δημιουργεί την περιγραφή για το νέο API καθώς επίσης και τον κώδικα για αυτό. Ο διακομιστής ανάπτυξης των νέων API θα έχει την ευθύνη να λαμβάνει τον κώδικα των νέων API από το Back-End και να τα αναπτύσσει (deploy) σε πραγματικό χρόνο.

Ως γλώσσα ανάπτυξης του συστήματος επιλέχθηκε η JavaScript καθώς δίνει δυνατότητες ανάπτυξης τόσο για τον πελάτη (client-side) όσο και για τον διακομιστή (server-side), προσφέροντας έτσι ομοιομορφία στο σύστημα. Την ίδια ώρα, η JavaScript θέτει στη διάθεση του χρήστη μια μεγάλη ποικιλία από Frameworks και βιβλιοθήκες, τα οποία μπορούν να βελτιστοποιήσουν και να επισπεύσουν τη διαδικασία ανάπτυξης.

3.1 Σύγκριση και Επιλογή Εργαλείων Ανάπτυξης Front-End

Η ευρεία υιοθέτηση της JavaScript ως της κύριας γλώσσας για τη δημιουργία δυναμικών και διαδραστικών ιστοσελίδων οδήγησε στην ανάγκη για εξορθολογισμό των βασικών διαδικασιών ανάπτυξης. Η ανάγκη αυτή οδήγησε με τη σειρά της στη δημιουργία πλαισίων ανάπτυξης (frameworks). Τα frameworks είναι περιβάλλοντα προγραμματισμού που σκοπό έχουν να παρέχουν στον προγραμματιστή όλα τα απαραίτητα εργαλεία τα οποία χρειάζεται για την ανάπτυξη ενός συστήματος, όπως βιβλιοθήκες, σετ εργαλείων, APIs και μεταγλωττιστές. Παρακάτω παρουσιάζονται τα τρία δημοφιλέστερα JavaScript Frameworks για Front-End development και στη συνέχεια γίνεται μια σύγκρισή τους για να επιλεγεί τελικά το Framework που θα χρησιμοποιηθεί στην ανάπτυξη του συστήματος[18].

3.1.1 Angular

Η Angular είναι ένα ανοικτό framework ανάπτυξης διαδικτυακών εφαρμογών βασισμένο στην TypeScript (συντακτικό υπερσύνολο της JavaScript). Η Angular κυκλοφόρησε για πρώτη φορά το 2010 από την Google με την ονομασία AngularJS, η οποία είναι βασισμένη στην JavaScript και συντηρείται μέχρι και σήμερα. Λίγα χρόνια αργότερα, το 2016, η Google κυκλοφόρησε την Angular 2 ή απλά Angular, η οποία πλέον χρησιμοποιεί TypeScript και επιφέρει σημαντικές αλλαγές τόσο στον τρόπο λειτουργίας και ανάπτυξης των εφαρμογών όσο και στην αρχιτεκτονική τους, εξ ου και η αλλαγή ονόματος. Η τελευταία σταθερή έκδοσή της είναι η Angular 8.

Μερικά χαρακτηριστικά της Angular τα οποία τη διαφοροποιούν από τα υπόλοιπα frameworks που παρουσιάζονται είναι:

- Επιτρέπει τη δημιουργία εφαρμογών μια σελίδας (Single Page Application) με υψηλή αποδοτικότητα και ασφάλεια.

- Υποστηρίζει την TypeScript η οποία παρέχει όλες τις δυνατότητες της JavaScript, αλλά προσθέτει επίσης τη δυνατότητα για στατικούς τύπους, καλύτερη πλοήγηση, αυτόματη συμπλήρωση και δυνατότητα εύρεσης λαθών κατά τη συγγραφή του κώδικα.
- Προσφέρει τρείς τρόπους Data Binding: 2-way binding, one-way property binding, event binding και interpolation.
- Υποστηρίζει το μοντέλο Model-View-ViewModel (MVVM), το οποίο επιτρέπει στους προγραμματιστές να δουλεύουν ξεχωριστά και ταυτόχρονα στα διάφορα κομμάτια μιας εφαρμογής.
- Λόγω του μοντέλου MVVM και τη χρήση Component, ο κώδικας που παράγεται είναι επαναχρησιμοποιήσιμος και κατανοητός, ενώ επιτρέπει ευκολότερη δοκιμή μονάδων (Unit Testing).
- Υπάρχει πληθώρα βοηθητικού υλικού τόσο από επίσημες πηγές (Google) αλλά και από την κοινότητα.
- Κάθε κύρια έκδοση της Angular υποστηρίζεται για 18 μήνες. Από αυτούς οι 6 αφορούν ενεργή υποστήριξη και οι 12 μήνες μακροχρόνια υποστήριξη. Αξίζει δε να σημειωθεί ότι σε περιπτώσεις που μια κύρια έκδοση της Angular σταματά τη χρήση κάποιων APIs δίνεται μια περίοδος 6 μηνών μέχρι να σταματήσει τελείως η υποστήριξή τους, δίνοντας έτσι αρκετό χρόνο στους προγραμματιστές να κάνουν τις απαραίτητες αλλαγές.

Θα ήταν όμως παράλειψη να μη γίνει αναφορά και στα μειονεκτήματα τις Angular, τα πιο σημαντικά από τα οποία συνοψίζονται στα πιο κάτω σημεία:

- Λόγω του ότι η Angular βασίζεται στην TypeScript και λόγω των σχετικά δύσκολων στην κατανόηση αρχιτεκτονικών που χρησιμοποιεί, είναι αρκετά δύσκολη στην εκμάθηση.
- Η μετάβαση από την AngularJS στην Angular είναι αρκετά δύσκολη, λόγω διαφορετικής αρχιτεκτονικής των δύο frameworks.
- Το γεγονός ότι η Angular ενημερώνεται σε κύρια έκδοση κάθε 6 μήνες αυξάνει το κόστος συντήρησης του κώδικα, με αποτέλεσμα να απομακρύνει πολλούς προγραμματιστές από το να τη χρησιμοποιούν.

Με τα θετικά και τα αρνητικά της χαρακτηριστικά χρησιμοποιείται από αρκετούς γνωστούς ιστότοπους, με τους σημαντικότερους από αυτούς να είναι: Google, Microsoft, Adobe και AWS.

3.1.2 ReactJS

Η React (ReactJS), σε αντίθεση με τα άλλα δύο frameworks στη λίστα μας, δεν είναι framework, αλλά μια ανοικτή βιβλιοθήκη της JavaScript, η οποία ειδικεύεται στη δημιουργία διαδραστικών διεπαφών χρήστη (UIs). Η React κυκλοφόρησε για πρώτη φορά το 2013 από τη Facebook και συντηρείται από την ίδια, καθώς και από μια κοινότητα εταιριών και χρηστών ως έργο ανοικτού κώδικα. Η React μπορεί επίσης να χρησιμοποιηθεί και για την ανάπτυξη εφαρμογών Android και iOS με τη χρήση React Native. Η πιο πρόσφατη έκδοση είναι η React 16.8.6.

Μερικά χαρακτηριστικά της React τα οποία τη διαφοροποιούν από τα υπόλοιπα frameworks είναι τα ακόλουθα:

- Είναι πολύ εύκολη στην εκμάθηση, αφού χρησιμοποιεί καθαρή JavaScript και έχει απλό σχεδιασμό.
- Υποστηρίζει την JSX. Αυτό σημαίνει ότι τα components της React μπορούν να γραφούν με τη χρήση JSX, το οποίο είναι μια επέκταση της JavaScript που επιτρέπει σύνταξη component, χρησιμοποιώντας σύνταξη παρόμοια με την HTML.
- Μπορεί να χρησιμοποιηθεί σε συνδυασμό με την TypeScript και τη Flow, προσφέροντας έτσι στατικούς τύπους.
- Προσφέρει one-way data binding από τον component πατέρα στο component παιδί, αποφεύγοντας έτσι ανεπιθύμητες και απρόβλεπτες ενέργειες. Ταυτόχρονα, διευκολύνει τον έλεγχο μονάδων.
- Είναι βασισμένη σε component, διευκολύνοντας έτσι την επαναχρησιμοποίηση και τη συντήρηση κώδικα.
- Το Virtual DOM της React την κάνει εξαιρετικά γρήγορη.

- Η μετάβαση στις νεότερες εκδόσεις της React είναι εξαιρετικά εύκολη με τη χρήση του εργαλείου CodeMod.
- Υπάρχει πληθώρα βοηθητικού υλικού τόσο από επίσημες πηγές (Facebook), αλλά και από την κοινότητα.

Όπως και τα υπόλοιπα εργαλεία όμως, συναντούμε κι εδώ κάποια μειονεκτήματα που θέτουν περιορισμούς στη χρήση του. Μερικά από τα σημαντικότερα της React είναι:

- Δεν επιβάλλει κάποιο στιλ προγραμματισμού, αλλά αφήνεται στον προγραμματιστή να αποφασίσει, επιτρέποντας έτσι κακό κώδικα και πρακτικές.
- Η χρήση του Virtual Dom επιφέρει αυξημένες ανάγκες μνήμης (RAM), αφού κρατιέται διαρκώς ένα αντίγραφο του DOM στη μνήμη.
- Η μικτή σύνταξη τις React (JavaScript – JSX) μπορεί να αποθαρρύνει ή να συγχίσει τους προγραμματιστές, τουλάχιστο στην αρχή.

Η React χρησιμοποιείται επίσης από γνωστούς ιστότοπους, με πιο σημαντικούς να είναι οι εξής: Facebook, Instagram, Netflix, Airbnb, Uber.

3.1.3 VueJS

Το Vue (Vue.js) είναι ένα ανοικτό προοδευτικό framework ανάπτυξης διαδικτυακών εφαρμογών μιας σελίδας (Single Page Applications), καθώς και διαδραστικών διεπαφών χρήστη (UIs). Το Vue κυκλοφόρησε για πρώτη φορά το 2014 από τον Evan You και συντηρείται από την ομάδα του Vue.js καθώς και μια κοινότητα χρηστών. Το Vue αναπτύχθηκε βασισμένο στις καλύτερες πρακτικές άλλων frameworks που προαναφέρθηκαν, όπως η Angular και η React. Η πιο πρόσφατη έκδοση είναι το Vue 2.6.10.

Όπως και τα προαναφερθέντα frameworks, έτσι και το Vue έχει κάποια χαρακτηριστικά που το διαφοροποιούν από τα υπόλοιπα frameworks, μερικά από τα οποία είναι τα εξής:

- Είναι ένα προοδευτικό framework που σημαίνει ότι μπορεί να εφαρμοστεί σταδιακά ή επιλεκτικά σε συγκεκριμένα κομμάτια ενός συστήματος χωρίς αρνητικές επιπτώσεις.
- Είναι εύκολο για νέους χρήστες, αφού χρειάζεται μόνο γνώση της JavaScript.
- Έχει το μικρότερο μέγεθος από όλα τα frameworks που παρουσιάζονται.
- Μπορεί να χρησιμοποιηθεί είτε σε ένα απλό UI είτε σε ένα Single Page Application μέχρι και σε μεγάλα και περίπλοκα συστήματα.
- Υπάρχει λεπτομερής τεκμηρίωση στη επίσημη σελίδα του Vue, η οποία διευκολύνει ακόμα περισσότερο την εκμάθηση.

Το σημαντικότερο τώρα μειονέκτημα του Vue είναι ότι λόγω της μικρότερης υιοθέτησής του δεν υπάρχει διαθέσιμο βοηθητικό υλικό στην κλίμακα που υπάρχει για τα δύο προαναφερθέντα πλαίσια ανάπτυξης, Angular και React.

Το Vue χρησιμοποιείται μεταξύ άλλων από τους ιστότοπους Alibaba, Gitlab, Xiaomi και Grammarly.

3.1.4 Σύνοψη και επιλογή

Έχοντας μελετήσει τα τρία εργαλεία ανάπτυξης Front-End, το επόμενο βήμα είναι η επιλογή ενός από αυτά για την πραγματοποίηση ενός συγκεκριμένου έργου. Είναι γεγονός ότι δεν μπορεί να υπάρξει καθαρή επιλογή με την έννοια του ποιο framework είναι καλύτερο, αφού κάθε ένα έχει τα θετικά και τα αρνητικά του χαρακτηριστικά, που το καθιστούν σε κάποιες περιπτώσεις, ενώ θέτουν περιορισμούς σε κάποιες άλλες. Αν και φαινομενικά και τα τρία έχουν τον ίδιο σκοπό, το κάθε ένα από αυτά διαπρέπει σε διαφορετικά σενάρια είτε λόγω του συστήματος το οποίο αναπτύσσεται είτε λόγω των προγραμματιστών, ανάλογα με τις γνώσεις και την εμπειρία τους. Ως εκ τούτου, επιλογή μπορεί να γίνει μόνο βασισμένη στο εκάστοτε έργο καθώς και στην εκάστοτε ομάδα ανάπτυξης.

Στο πλαίσιο της παρούσας διπλωματικής εργασίας έχει επιλεγεί η βιβλιοθήκη React για τις ανάγκες του Front-End. Η επιλογή αυτή είναι βασισμένη σε διάφορες αιτίες. Αρχικά, το μεγαλύτερο υπολογιστικό κομμάτι θα γίνεται από το Rest API. Το Frontend θέλουμε να χειρίζεται μόνο το UI, κάτι στο οποίο εξειδικεύεται η React. Επιπλέον, θεωρείται απαραίτητο η εφαρμογή να είναι ελαφριά και γρήγορη, κάτι το οποίο προσφέρει η React. Τελευταίο αλλά όχι έσχατο κίνητρο αποτέλεσε η προηγούμενη εμπειρία του συγγραφέα με το συγκεκριμένο εργαλείο.

3.2 Εργαλεία Ανάπτυξης Back-End (Rest API)

Για την ανάπτυξη του Rest API επιλέχθηκε το περιβάλλον Node.js ως το πιο διαδεδομένο JavaScript runtime έξω από τους περιηγητές ιστού, καθώς επίσης και για επιπλέον λόγους, οι οποίοι αναφέρονται στην ανάλυση του εργαλείου, όπως παρουσιάζεται πιο κάτω.

3.2.1 Node.js

Το Node.js είναι μια ανοικτή (Open-Source) πλατφόρμα ανάπτυξης λογισμικού βασισμένη στη JavaScript. Παρέχει ένα περιβάλλον εκτέλεσης JavaScript έξω από τον περιηγητή ιστού, βασισμένο στο Google V8 JavaScript engine. Το V8 JavaScript engine είναι μια μηχανή εκτέλεσης JavaScript ανοιχτού κώδικα, η οποία βρίσκεται πίσω από το Chrome και όλους τους περιηγητές ιστού βασισμένους στο Chromium. Το Node.js κυκλοφόρησε για πρώτη φορά το 2009 από τον Ryan Dahl και η πιο πρόσφατη έκδοσή του είναι το Node.js 12.4.0.

Η κύρια χρήση του Node.js είναι η δημιουργία διακομιστών γραμμένων σε JavaScript, δημιουργώντας έτσι ένα ενοποιημένο περιβάλλον ανάπτυξης διαδικτυακών εφαρμογών, όπου τόσο η μεριά του πελάτη όσο και η μεριά του διακομιστή είναι γραμμένα σε JavaScript. Έτσι μειώνονται δραματικά οι τεχνολογίες που πρέπει να γνωρίζει ένας προγραμματιστής ώστε να αναπτύξει μια εφαρμογή.

Η αρχιτεκτονική του Node.js, σε αντίθεση με άλλα περιβάλλοντα ανάπτυξης εφαρμογών δικτύου, δε βασίζεται στην πολυνηματικότητα, αλλά στην ασύγχρονη είσοδο/έξοδο (asynchronous I/O). Δηλαδή, λειτουργεί σε ένα μόνο νήμα χρησιμοποιώντας κλήσεις εισόδου/εξόδου οι οποίες δε σταματούν την εκτέλεση του κώδικα (non-blocking I/O). Αυτό του επιτρέπει να χειρίζεται πολλαπλές συνδέσεις χωρίς να χρειάζεται πολλαπλά νήματα. Για να επιτευχθεί αυτό, κάθε διαδικασία η οποία διενεργεί κάποια είσοδο/έξοδο πρέπει να δηλώσει μια συνάρτηση επανάκλησης (callback), δηλαδή μια συνάρτηση η οποία θα εκτελεστεί μόλις τελειώσει η είσοδος/έξοδος.

Το 2010 παρουσιάστηκε το npm (Node Package Manager), ένας διαχειριστής πακέτων (package manager) για το Node.js. Το npm αποτελείται από ένα εργαλείο γραμμής εντολών και μια online βάση δεδομένων. Το εργαλείο γραμμής εντολών επικοινωνεί με τη βάση δεδομένων, η οποία περιέχει δημόσια ή ιδιόκτητα πακέτα npm. Το npm προσφέρει στον προγραμματιστή τη δυνατότητα να δημοσιεύει και να μοιράζεται τον κώδικά του στη μορφή πακέτων npm. Επίσης διευκολύνει το κατέβασμα, την εγκατάσταση και συντήρηση βιβλιοθηκών JavaScript, οι οποίες υπάρχουν σαν πακέτα npm.

server.js

```
1  var express = require('express');
2  var app = express();
3  var port =3000;
4
5  app.get('/', (req, res) => {
6    res.send('Hello World')
7  })
8
9  app.listen(port, (err) => {
10     console.log(`server is listening on ${port}`)
11 })
```

Εικόνα 7: Παράδειγμα απλού διακομιστή σε Node.js

3.3 Deployment Server

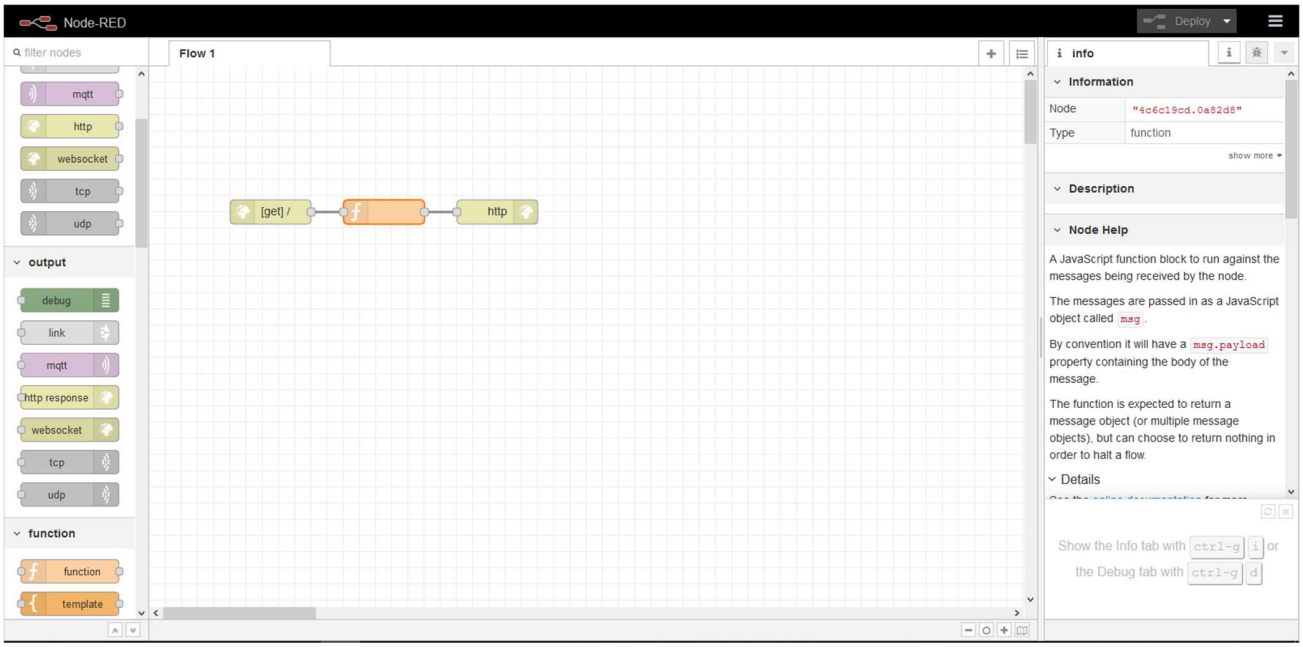
Ως Deployment Server για τις ανάγκες της παρούσας εργασίας επιλέγηκε το Node-Red, καθώς δίνει τη δυνατότητα ανάπτυξης κώδικα (code deployment) κατά τη διάρκεια εκτέλεσής του (at runtime). Παρακάτω γίνεται μια σύντομη περιγραφή του.

3.3.1 Node-RED

Το Node-Red είναι ένα ανοιχτό εργαλείο ανάπτυξης λογισμικού για το IoT και είναι βασισμένο στο Node.js. Κυκλοφόρησε για πρώτη φορά το 2013 από την IBM και το 2016 μετατράπηκε σε έργο ανοικτού κώδικα. Συντηρείται πλέον από την IBM, καθώς και από κοινότητα χρηστών.

Το Node-RED προσφέρει ένα οπτικό περιβάλλον προγραμματισμού, το οποίο χρησιμοποιεί μοντέλο προγραμματισμού βασισμένο σε ροές. Αυτό σημαίνει ότι αποκρύπτει το πώς δουλεύουν οι διαδικασίες κλείνοντας τις σε «μαύρα κουτιά» (nodes) και δίνοντας στον προγραμματιστή τη δυνατότητα να φτιάξει την εφαρμογή του συνδέοντας διάφορα nodes μεταξύ τους. Οι ροές που δημιουργούνται στο Node-RED αποθηκεύονται σαν αρχεία JSON.

Το Node-RED έχει φτιαχτεί για να υποστιρίζει το IoT. Κατά συνέπεια, διευκολύνει την καλωδίωση υλικών συσκευών, online υπηρεσιών και APIs. Η δημιουργία APIs μπορεί να γίνει με μηδενικό κώδικα, όπως φαίνεται στην Εικόνα 8. Δίνει ακόμα τη δυνατότητα στον χρήστη, μέσω του Admin API και της μεθόδου POST /flow, να αναπτύξει δυναμικά μια νέα ροή κατά τη διάρκεια εκτέλεσής του.



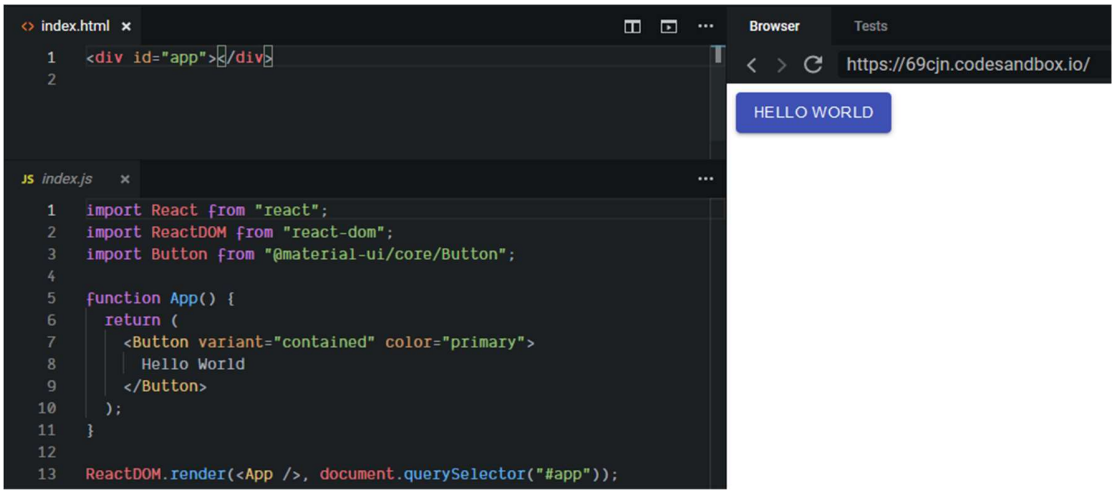
Εικόνα 8: Παράδειγμα απλού API στο Node-RED απαντάει "Hello World" στη διεύθυνσή http://localhost:1880/

3.4 Βιβλιοθήκες JavaScript

Όπως έχει είδη αναφερθεί, η JavaScript είναι μια από τις πιο διαδεδομένες γλώσσες προγραμματισμού στη σημερινή εποχή, με χρήσεις που κυμαίνονται από εφαρμογές ιστού μέχρι εφαρμογές διακομιστών ακόμα και εφαρμογές σε έξυπνα κινητά. Όπως είναι λογικό, είναι δύσκολο και πολύπλοκο να υποστηριχθεί αυτή η ποικιλία εφαρμογών χρησιμοποιώντας απλή JavaScript. Σ' αυτό έρχονται να συμβάλουν οι βιβλιοθήκες JavaScript, οι οποίες υλοποιούν διάφορες από τις λειτουργίες που χρησιμοποιούνται συχνότερα και προσθέτουν ένα επίπεδο αφαίρεσης, προσφέροντας περίπλοκες λειτουργίες υπό τη μορφή συναρτήσεων. Πιο κάτω, γίνεται μια σύντομη περιγραφή των σημαντικότερων βιβλιοθηκών που θα χρησιμοποιηθούν στην ανάπτυξη του συστήματος.

3.4.1 Material-UI

Το Material-UI είναι μια βιβλιοθήκη της JavaScript, η οποία περιέχει έτοιμα React Components και είναι βασισμένη στο Material Design της Google. Προσφέρει τη δυνατότητα δημιουργίας δυναμικών και καλά σχεδιασμένων ιστοσελίδων χωρίς τη χρήση CSS από τον προγραμματιστή. Τα Components είναι αυτοϋποστηριζόμενα και μπορούν να εισαχθούν ένα προς ένα σε μια ιστοσελίδα μειώνοντας έτσι το μέγεθος των εισαγωγών, όπως φαίνεται στην Εικόνα 9[19].



Εικόνα 9: Παράδειγμα χρήσης του Button Component από το Material-UI

3.4.2 material-table

Το material-table είναι μια βιβλιοθήκη της JavaScript βασισμένη στο Material-UI. Η βιβλιοθήκη αυτή επεκτείνει το Table Component του Material-UI προσθέτοντας περισσότερες λειτουργίες και διευκολύνοντας τη συμπλήρωση ένθετων δεδομένων στον πίνακα[20].

3.4.3 Express

Η Express (Express.js) είναι μια βιβλιοθήκη της JavaScript για χρήση με το Node.js. Η βιβλιοθήκη αυτή είναι σχεδιασμένη για την ανάπτυξη εφαρμογών ιστού και APIs. Προσφέρει πληθώρα συναρτήσεων για τη διαχείριση συνδέσεων HTTP (Cookies, Sessions, Data Parsing), οι οποίες απλοποιούν κατά πολύ την ανάπτυξη εφαρμογών και επίσης μειώνουν κατά πολύ την ποσότητα του κώδικα που χρειάζεται σε σχέση με το HTTP module του Node.js χωρίς αρνητικό αντίκτυπο στη λειτουργικότητα. Όλα αυτά φαίνονται στο παράδειγμα «Hello World» στην Εικόνα 10[21].

ExpressServer.js

```
const express = require('express')
const app = express()

app.get('/', function (req, res) {
  res.send('Hello World')
})

app.listen(3000)
```

HTTPServer.js

```
const http = require('http');

const app=http.createServer(function (req, res) {
  if (req.url === '/') {
    res.write('Hello World!');
    res.end();
  }
  else{
    res.statusCode = 404;
    res.end();
  }
});

app.listen(3000);
```

Εικόνα 10: Σύγκριση χρήσης Express.js και HTTP module

3.4.4 node-fetch

Το node-fetch είναι επίσης μια βιβλιοθήκη της JavaScript για χρήση με το Node.js. Το node-fetch υλοποιεί το γνωστό fetch API σε περιβάλλον Node.js επιτρέποντας τη χρήση του αντί του XMLHttpRequest, το οποίο είναι δύσκολο στη χρήση. Την ίδια ώρα, το fetch επιτρέπει καλύτερη χρονική διαχείριση των αιτήσεων HTTP, αφού είναι βασισμένο σε Promises[22].

3.4.5 lodash

Μια βιβλιοθήκη της JavaScript είναι και το lodash, το οποίο παρέχει συναρτήσεις για κοινές λειτουργίες βασισμένες στο μοντέλο του συναρτησιακού προγραμματισμού. Διευκολύνει κατά πολύ τη διαχείριση πινάκων (arrays), αντικειμένων (objects), αριθμών και κειμένων (strings). Το lodash δίνει τη δυνατότητα στον προγραμματιστή είτε να εισαγάγει ολόκληρη τη βιβλιοθήκη στο έργο του είτε να εισαγάγει μόνο τις συγκεκριμένες μεθόδους που χρειάζεται, μειώνοντας έτσι το μέγεθος της εφαρμογής.

Κεφάλαιο 4 Υλοποίηση – Λειτουργία Συστήματος

Σε αυτό το κεφάλαιο επεξηγείται η υλοποίηση των βασικών κομματιών του συστήματος και δίνεται επίσης και ένα παράδειγμα λειτουργίας της.

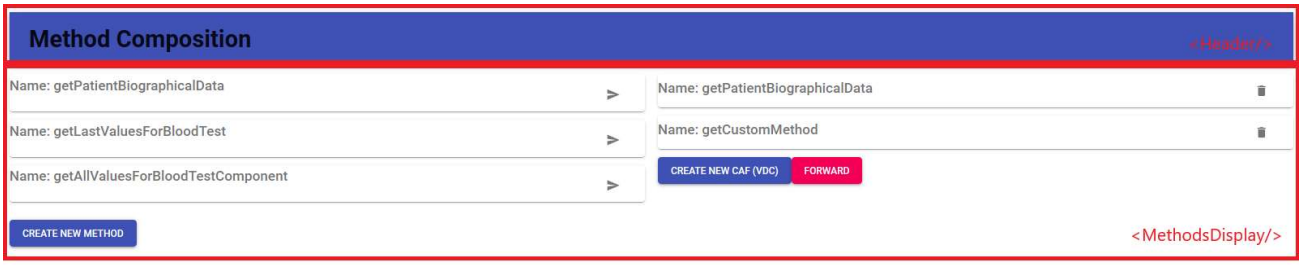
Σκοπός του συστήματος είναι να δώσει στον χρήστη τη δυνατότητα της επεξεργασίας ενός υπάρχοντος VDC Blueprint, ώστε να πάρει τα δεδομένα που χρειάζεται ακριβώς με τον τρόπο που τα χρειάζεται. Το front-end του συστήματος βρίσκεται για σκοπούς υλοποίησης στη διεύθυνση http://localhost:3000/ και στην εκκίνησή του λαμβάνει το αρχικό VDC Blueprint. Το back-end βρίσκεται στη διεύθυνση http://localhost:5000/ και περιμένει εισερχόμενες συνδέσεις. Τέλος, ο διακομιστής ανάπτυξης βρίσκεται στη διεύθυνση http://localhost:1880/ και περιμένει και αυτός με τη σειρά του εισερχόμενες συνδέσεις. Όταν το σύστημα αναπτυχθεί στα πλαίσια του DITAS, ο χρήστης θα χρειάζεται μόνο να ξέρει το domain name του front-end, καθώς ο χρήστης δεν έχει απευθείας επικοινωνία με το back-end και η διεύθυνση του διακομιστή ανάπτυξης θα περιέχεται στο νέο VDC Blueprint.

4.1 Υλοποίηση Front-End

Το front-end υλοποιείται με μια εφαρμογή μονής σελίδας (Single Page Application) με τη χρήση της βιβλιοθήκης React. Για την ανάπτυξη με τη χρήση της βιβλιοθήκης React, η σελίδα χωρίζεται σε components, γεγονός το οποίο απλοποιεί την υλοποίηση και επιτρέπει πιο εύκολη επαναχρησιμοποίηση κώδικα. Το αρχικό component της σελίδας είναι το App component μέσα στο οποίο περιέχονται όλα τα υπόλοιπα components και ουσιαστικά ολόκληρη η σελίδα.

App Component

Το App component καλεί (renders) τα components Header και MethodsDisplay, όπως φαίνεται στην Εικόνα 11. Ο κώδικας του App component υπάρχει στο Παράρτημα Α.1.



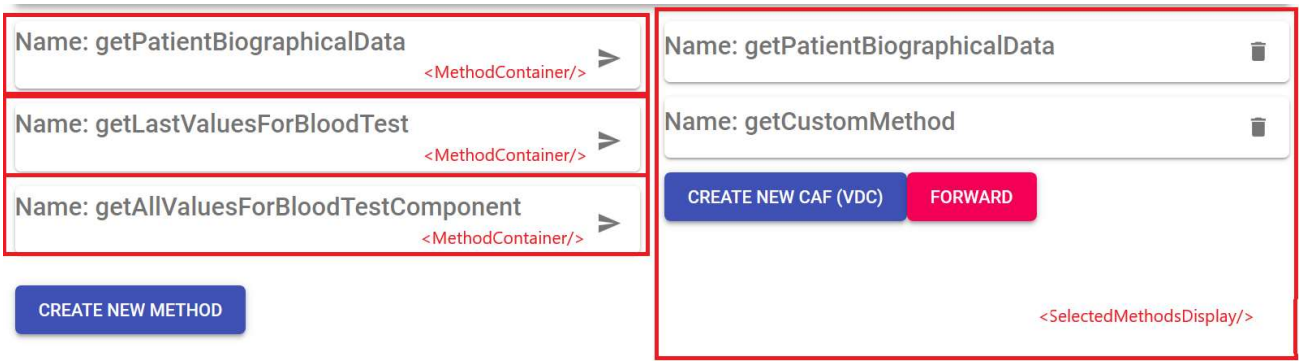
Εικόνα 11:App Component

Header Component

Το Header component έχει εξωραϊστικό σκοπό, δηλαδή δεν περιέχει κάποια λειτουργία παρά μόνο παρουσιάζει τον τίτλο της σελίδας.

MethodsDisplay Component

Το MethodsDisplay component υλοποιεί την ολική λειτουργία τις σελίδας. Εδώ υλοποιείται όλη η λογική πίσω από την επιλογή και τη δημιουργία νέων μεθόδων. Το MethodsDisplay component κάνει render τα component MethodContainer και SelectedMethodsDisplay, όπως φαίνεται στην Εικόνα 12. Ο βασικός κώδικας βρίσκεται στο Παράρτημα Α.2.



Εικόνα 12: MethodsDisplay Component

Η πρώτη μέθοδος που καλείται όταν ένα component γίνεται render είναι ο constructor, ο οποίος έχει την ευθύνη να κατασκευάσει το component και να αρχικοποιήσει τυχόν μεταβλητές ή

σταθερές τις οποίες χρησιμοποιεί το component. Στη συγκεκριμένη περίπτωση, αρχικοποιείται το state του component. Το state περιλαμβάνει μεταξύ άλλων το αντικείμενο apiDesc, το οποίο περιέχει την ενότητα EXPOSED_API του VDC Blueprint και το array selection όπου αποθηκεύονται οι επιλογές του χρήστη από τα components MethodResponseGrid. Ακόμα, περιέχει το αντίκείμενο selectedResponses, στο οποίο αποθηκεύονται οι επιλογές του χρήστη αμέσως πριν τη δημιουργία της νέας μεθόδου. Επίσης, περιέχει το array selectedMethods, το οποίο μεταφέρεται στο component SelectedMethodsDisplay και περιέχει τις μεθόδους που είτε έχει επιλέξει είτε έχει δημιουργήσει ο χρήστης. Τέλος, υπάρχει το αντικείμενο form στο οποίο αποθηκεύονται τα στοιχεία για τις νέες μεθόδους από το κουτί διαλόγου.

Το component αρχικά διαβάζει το εισερχόμενο VDC Blueprint κάνοντας χρήση της μεθόδου κύκλου ζωής της React componentDidMount και στη συνέχεια αφαιρεί όλες τις αναφορές χρησιμοποιώντας τη βιβλιοθήκη $RefParser και το αποθηκεύει στο state του component.

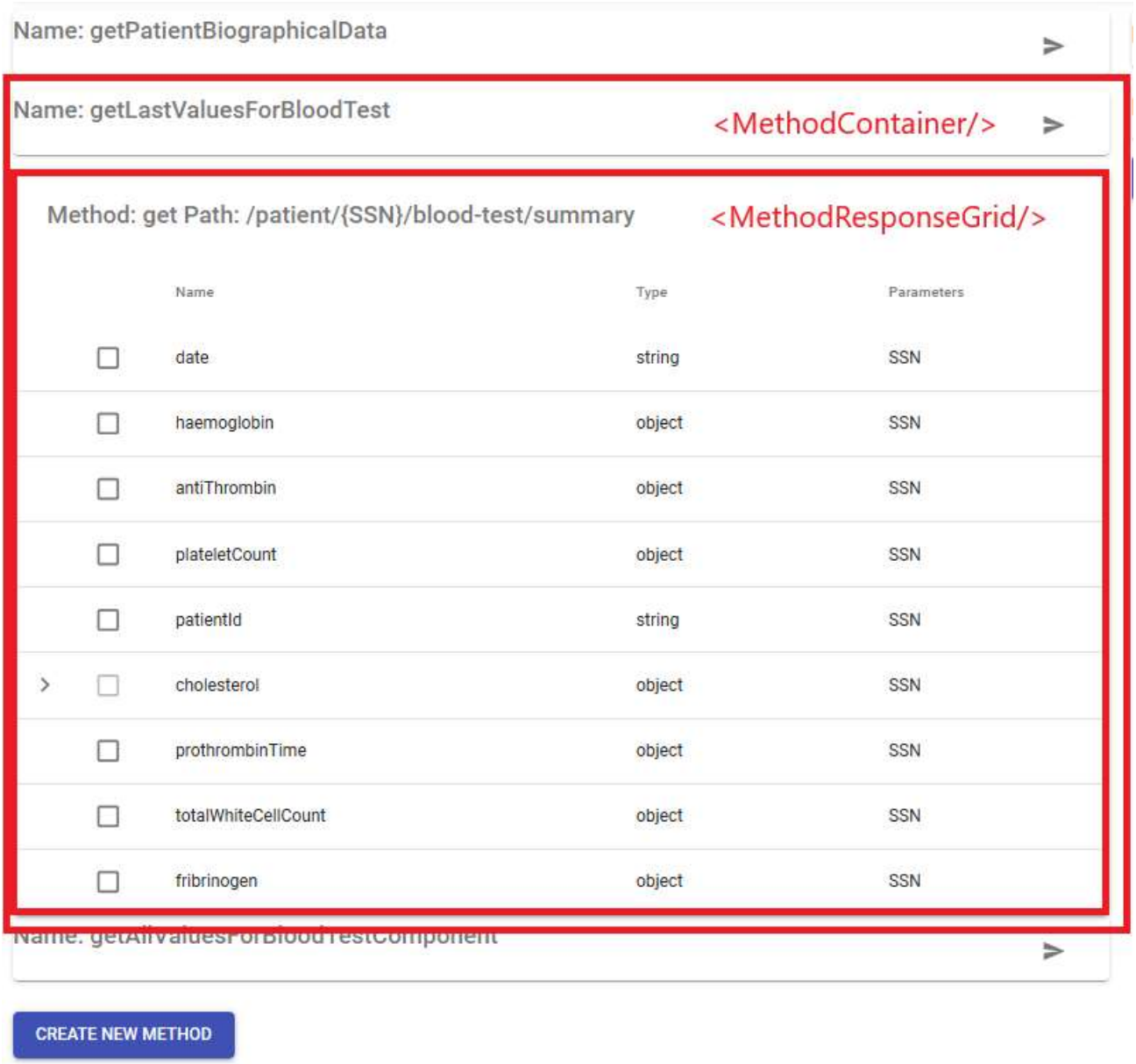
Η επόμενη μέθοδος κύκλου ζωής η οποία τρέχει είναι η render η οποία δημιουργεί τα component MethodContainer, ένα για κάθε μέθοδο του αρχικού VDC Blueprint, και περνάει σαν ορίσματα τα στοιχεία της εκάστοτε μεθόδου καθώς και δύο συναρτήσεις ελέγχου, τη handleMethodSelection και την updateGlobalSelection. Ο σκοπός της handleMethodSelection είναι ο χειρισμός επιλογής υφιστάμενης μεθόδου για χρήση στο νέο VDC και της updateGlobalSelection είναι να μεταφερθεί στο component MethodResponseGrid, ώστε να χειρίζεται τις επιλογές δεδομένων για δημιουργία νέων μεθόδων. Στη συνέχεια, η μέθοδος render δημιουργεί το κουμπί CREATE NEW METHOD, που ενεργοποιεί τη μέθοδο createNewMethod, στην οποία θα γίνει ειδική αναφορά παρακάτω. Ακόμα, δημιουργεί το component SelectedMethodsDisplay δίνοντάς του ως ορίσματα τις επιλεγμένες μεθόδους και τις μεθόδους handleDelete και clearSel. Η πρώτη χειρίζεται τη διαγραφή επιλεγμένων μεθόδων και η δεύτερη καθαρίζει το state του MethodsDisplay όταν δημιουργείται το νέο VDC. Τέλος, η render δημιουργεί το κουτί διαλόγου που ανοίγει με το πάτημα του κουμπιού CREATE NEW METHOD, για την εισαγωγή των στοιχείων της νέας μεθόδου. Στο κλείσιμο του διαλόγου καλείται η μέθοδος handleClose για να δημιουργήσει τη νέα μέθοδο.

Μια από τις σημαντικότερες μεθόδους αυτού του component είναι η μέθοδος createNewMethod. Η μέθοδος αυτή καλείται όταν πατηθεί το κουμπί CREATE NEW METHOD. Αρχικά, προετοιμάζει τις επιλογές του χρήστη για τη δημιουργία της νέας μεθόδου παίρνοντας μόνο τα επιλεγμένα responses από το array selection και περνώντας τα στη συνάρτηση applyRules, η οποία περιγράφεται στη συνέχεια για έλεγχο. Αν περάσουν επιτυχώς τον έλεγχο μεταφέρονται στο αντικείμενο selectedResponses και ανοίγει το κουτί διαλόγου για την εισαγωγή των στοιχείων της μεθόδου. Κατά το κλείσιμο του διαλόγου δημιουργείται η νέα μέθοδος και μεταφέρεται στο αντικείμενο selectedMethods.

Η μέθοδος applyRules έχει σκοπό να ελέγξει τις επιλογές του χρήστη βάσει ενός συνόλου κανόνων. Για τον έλεγχο χρησιμοποιείται η παράμετρος path, η οποία δίνεται στα responses από το MethodResponseGrid component και αντιπροσωπεύει τη θέση του κάθε response. Ο ακριβής τρόπος δημιουργίας περιγράφεται στην παράγραφο για το MethodResponseGrid component. Υπάρχουν δύο κανόνες: ο ένας είναι σημασιολογικός και ο άλλος προέρχεται από τον τρόπο υλοποίησης του συστήματος. Ο πρώτος κανόνας είναι ότι από μεθόδους των οποίων το σχήμα τους δίνεται με τη βοήθεια της λέξεις-κλειδί oneOf μπορούν να επιλεγούν responses μόνο από ένα σχήμα, αφού δεν μπορεί η μέθοδος να δώσει απάντηση από δύο διαφορετικά σχήματα σε μία κλήση της. Ο δεύτερος κανόνας έχει να κάνει με την υλοποίηση του συστήματος και λέει ότι δύο responses δεν μπορούν να έχουν το ίδιο path, αφού βάσει του path καθορίζεται το σχήμα των νέων μεθόδων που δημιουργούνται, με αποτέλεσμα αν είχαμε δύο responses με το ίδιο path μόνο το ένα θα επιστρεφόταν από τη νέα μέθοδο χωρίς να μπορούμε να προβλέψουμε ποιο.

MethodContainer Component

Το MethodContainer είναι το component το οποίο είναι υπεύθυνο για την παρουσίαση της κάθε μεθόδου που υπάρχει στο αρχικό VDC Blueprint. Όταν πατηθεί, κάνει render το MethodResponseGrid component όπως φαίνεται στην Εικόνα 13. Ο βασικός κώδικας βρίσκεται στο Παράρτημα Α.3.



Εικόνα 13:MethodContainer Component

Ο constructor του MethodContainer είναι η πρώτη μέθοδος που τρέχει πριν γίνει render το component και σκοπό έχει να αρχικοποιήσει τη μεταβλητή isHidden του state, η οποία ελέγχει το κατά πόσο φαίνεται ή όχι το component MethodResponseGrid, το οποίο καλείται στη μέθοδο render.

Στη συνέχεια τρέχει η μέθοδος render, η οποία δημιουργεί ένα component Paper και ένα MethodResponseGrid component. Το Paper component περιέχει ένα Typography component που περιέχει μέσα το όνομα της μεθόδου και ελέγχεται από τη μέθοδο handleClick και ένα IconButton

component που είναι υπεύθυνο για την επιλογή της μεθόδου ως έχει και ελέγχεται από τη μέθοδο handleMethodSelection. Το MethodResponseGrid component λαμβάνει ως ορίσματα την αρχική μέθοδο που έχει ληφθεί από το MethodsDisplay, τη μέθοδο updateState και τη μεταβλητή isHidden.

Οι μέθοδοι updateState και handleMethodSelection λειτουργούν ως ενδιάμεσες μέθοδοι για τις μεθόδους updateGlobalSelection και handleMethodSelection του MethodsDisplay component.

Η μέθοδος handleClick καλείται κάθε φορά που γίνεται κλικ στο Typography component και αλλάζει την τιμή της μεταβλητής isHidden.

MethodResponseGrid Component

Το MethodResponseGrid είναι το component το οποίο παρουσιάζει όλες τις δυνατές απαντήσεις, τις οποίες μπορεί να έχει μια μέθοδος του αρχικού VDC Blueprint σε μορφή πίνακα, δίνοντας τη δυνατότητα επιλογής responses για τη δημιουργία νέας μεθόδου. Ο βασικός κώδικας βρίσκεται στο Παράρτημα Α.4.

Η πρώτη μέθοδος η οποία καλείται είναι ξανά ο constructor, που σε αυτή την περίπτωση σκοπό έχει την αρχικοποίηση του state και συγκεκριμένα του array rows, στο οποίο θα αποθηκευτούν τα responses της αρχικής μεθόδου σε μορφή γραμμών πίνακα.

Στη συνέχεια, καλείται η μέθοδος κύκλου ζωής componentDidMount, η οποία σε συνδυασμό με την αναδρομική μέθοδο addProperties έχει σκοπό την προετοιμασία των responses της εκάστοτε αρχικής μεθόδου σε γραμμές πίνακα. Κάθε γραμμή του πίνακα είναι ένα αντικείμενο με τα στοιχεία id, name, path, type, parameters, partof και για nested αντικείμενα το στοιχείο parentId. Σε αυτό το σημείο είναι που δημιουργείται και το path. Για τη δημιουργία του ακολουθούνται οι εξής κανόνες:

- Σε περίπτωση που το σχήμα της μεθόδου δίνεται με τη λέξη κλειδί oneOf, το path όλων των responses ξεκινά με τη συμβολοσειρά “oneOf.schema”, ακολουθούμενη από τον

αριθμό του σχήματος με τη σειρά που υπάρχουν στο VDC Blueprint και ξανά τελεία. Π.χ. “oneOf.schema2.”

- Σε περίπτωση που το σχήμα της μεθόδου είναι τύπου Array, τότε προστίθεται η συμβολοσειρά “Array.” στο path. Π.χ “oneOf.schema2.Array”.
- Στη συνέχεια, επεξεργαζόμαστε τα properties του σχήματος ένα ένα, όπου το κάθε ένα παίρνει το δικό του path, στο οποίο προστίθεται το όνομα του. Π.χ “oneOf.schema2.Array.cholesterol”.
- Σε περίπτωση που κάποιο από τα properties περιέχει properties σε δεύτερο επίπεδο, τότε κάθε ένα από αυτά παίρνει δικό του path άρα και ξεχωριστή γραμμή στον πίνακα. Π.χ “oneOf.schema2.Array.cholesterol.hdl”.

Τελευταία καλείται η μέθοδος render η οποία κατ’ αρχάς ελέγχει τη μεταβλητή isHidden και στην περίπτωση που έχει τιμή true, επιστρέφει null. Σε περίπτωση που έχει τιμή false, επιστρέφει ένα MaterialTable component, το οποίο παρέχεται από τη βιβλιοθήκη material-table. Το MaterialTable component είναι ένα component που προσφέρει τεράστια λειτουργικότητα. Κατόπιν, θα παρουσιαστούν οι βασικές ρυθμίσεις που χρησιμοποιήθηκαν. Αρχικά χρησιμοποιείται το tree-data mode, οπότε ορίζεται η σχέση μεταξύ parent και child rows (για nested responses) με το όρισμα parentChildData. Με το όρισμα onSelectionChange ορίζεται το τι θα γίνεται κάθε φορά που ενεργοποιείται ή απενεργοποιείται κάποιο από τα check-boxes για κάθε γραμμή. Στο όρισμα data δίνεται το array rows από το state του component, το οποίο περιέχει πλέον τα στοιχεία του πίνακα. Στο όρισμα columns δίνεται ο τρόπος με τον οποίο τα δεδομένα που δόθηκαν στο όρισμα data θα εμφανίζονται στον πίνακα. Τέλος, υπάρχει το όρισμα options, το οποίο ελέγχει όλες τις λειτουργίες του πίνακα. Εδώ τίθεται η μεταβλητή selection σε true, ώστε να ενεργοποιηθεί η δυνατότητα επιλογής γραμμών στον πίνακα και επίσης στη μεταβλητή selectionProps δίνεται μια συνάρτηση, η οποία ορίζει ποιες γραμμές δε θα είναι ενεργοποιημένες προς επιλογή.

SelectedMethodsDisplay Component

Το SelectedMethodsDisplay component είναι υπεύθυνο να παρουσιάζει τις μεθόδους που είτε έχει επιλέξει ο χρήστης είτε έχει φτιάξει, καθώς και για την αποστολή των δεδομένων στο back-end. Ο βασικός κώδικας βρίσκεται στο Παράρτημα Α.5.

Η πρώτη μέθοδος η οποία καλείται είναι ξανά ο constructor που σε αυτή την περίπτωση σκοπό έχει την αρχικοποίηση του state, το οποίο περιέχει τις μεταβλητές open και msg που σκοπό έχουν τον έλεγχο του διαλόγου επιβεβαίωσης ή προβλήματος.

Στη συνέχεια τρέχει η μέθοδος render, η οποία σε περίπτωση που το όρισμα methods του component (δηλαδή το selectedMethods του MethodsDisplay component) είναι άδειο, τότε επιστρέφει ένα Typography component με το κείμενο “Waiting…”. Σε αντίθετη περίπτωση, δημιουργεί ένα array από Paper components, κάθε ένα από τα οποία αντιστοιχεί σε μια μέθοδο στο όρισμα methods. Τα Paper components περιέχουν ένα Typography component, με κείμενο το όνομα της νέας μεθόδου και ένα IconButton component το οποίο ελέγχεται από τη μέθοδο handleDelete. Μετά τα Paper components δημιουργείται το κουμπί Create New CAF (VDC), το οποίο ελέγχεται από τη μέθοδο createCAF. Τέλος, δημιουργείται ένα κουτί διαλόγου.

Η μέθοδος createCAF καλείται όταν πατηθεί το κουμπί Create New CAF (VDC). Η μέθοδος αυτή χρησιμοποιώντας τη μέθοδο fetch εκτελεί ένα HTTP POST request προς το back-end του συστήματος, δηλαδή ένα REST API που ακούει στη διεύθυνση http://localhost:5000/. Συγκεκριμένα, το HTTP POST request γίνεται προς τη διεύθυνση http://localhost:5000/generate. Το body του HTTP POST request περιέχει της επιλεγμένες μεθόδους, καθώς και την ενότητα EXPOSED_API του αρχικού VDC Blueprint. Το REST API αναλαμβάνει να αναπτύξει τις νέες μεθόδους στο NODE-RED και να δημιουργήσει το specification για αυτό με βάση την προδιαγραφή OpenAPI. Όταν ολοκληρωθεί η διαδικασία, επιστρέφεται πίσω στην createCAF ένα μήνημα με τον τρόπο ολοκλήρωσης της διαδικασίας. Τέλος, η createCAF ανοίγει το κουτί διαλόγου όπου παρουσιάζει το μήνημα που πήρε.

Τέλος, υπάρχει η μέθοδος handleDelete, η οποία καλείται όταν πατηθεί το IconButton δίπλα από το όνομα κάθε μεθόδου. Η handleDelete είναι μια ενδιάμεση μέθοδος αφού το μόνο που κάνει είναι να καλεί τη μέθοδο handleDelete του component MethodsDisplay.

4.2 Υλοποίηση Back-End

Το back-end υλοποιείται ακολουθώντας το μοντέλο REST και ως περιβάλλον ανάπτυξης χρησιμοποιείται το Node.js.

App.js

Το αρχείο εισόδου είναι το app.js και ο βασικός κώδικας βρίσκεται στο Παράρτημα Β.1. Αρχικά, δημιουργείται ο διακομιστής με τη χρήση της βιβλιοθήκης express. Ο διακομιστής ρυθμίζεται έτσι ώστε να αποδέχεται cross origin request με τη χρήση της βιβλιοθήκης cors. Στη συνέχεια, με τη χρήση της βιβλιοθήκης bodyParser προστίθεται η δυνατότητα για αποκωδικοποίηση του σώματος (body) των HTTP request που καταφθάνουν στον διακομιστή.

Τελικά ο διακομιστής ρυθμίζεται ώστε να περιμένει συνδέσεις στις διαδρομές που ορίζονται στο αρχείο Routes/index.js και εκκινάται ώστε να ακούει στη θύρα 5000.

Routes/index.js

Στο αρχείο αυτό ορίζονται οι διαδρομές στις οποίες ακούει το API, δηλαδή οι μέθοδοι που υποστηρίζει, και ο βασικός κώδικας βρίσκεται στο Παράρτημα Β.2. Πέρα από τον ορισμό τους, τούς ανατίθεται από μία συνάρτηση callback η οποία τρέχει κάθε φορά που λαμβάνεται ένα request στη συγκεκριμένη μέθοδο. Οι συναρτήσεις αυτές βρίσκονται στο αρχείο flowsControllers/flows.js. Η βασική μέθοδος του API είναι /generate η οποία ελέγχεται από τη συνάρτηση flowsController.genetateFlow που θα αναλυθεί πιο κάτω.

flows.js

Στο αρχείο αυτό ορίζονται οι callback συναρτήσεις που ελέγχουν τις διαδρομές του API και ο βασικός κώδικας βρίσκεται στο Παράρτημα Β.3.

Η πρώτη συνάρτηση που φαίνεται είναι η info, που σκοπό έχει να παρέχει βασικές πληροφορίες για το σύστημα στη διεύθυνση http://localhost:5000.

Στη συνέχεια, παρουσιάζεται μία από τις σημαντικότερες μεθόδους του back-end, η generateFlow. Σκοπός της είναι η δημιουργία των Node-RED flows και η αποστολή τους στο Node-RED ώστε να γίνουν deploy, αλλά και η δημιουργία της περιγραφής του νέου API με βάση την προδιαγραφή OpenApi specification.

Αρχικά, η μέθοδος ελέγχει αν έχει λάβει όλες τις απαραίτητες παραμέτρους και σε αρνητική περίπτωση επιστρέφει με status code 400 και μήνυμα λάθους. Στην περίπτωση που έχουν δοθεί όλες οι παραμέτροι σωστά, η μέθοδος ξεκινά τη δημιουργία τις περιγραφής του νέου API μέσα στο αντικείμενο newOpenAPi.EXPOSED_API, προσθέτοντας αρχικά την έκδοση του OpenAPI που χρησιμοποιείται, γενικές πληροφορίες για το API και τους διακομιστές στους οποίους θα τρέχει. Τέλος, αρχικοποιεί το αντικείμενο paths.

Έπειτα, για κάθε μια από τις νέες μεθόδους που έχει λάβει από το front-end, δημιουργεί ένα αντικείμενο Flow. Η κλάση Flow ορίζεται στο αρχείο flow.js, το οποίο αντιπροσωπεύει το Node-RED flow που θα γίνει POST στον διακομιστή Node-RED. Ακολούθως, εκτελείται η μέθοδος createFromMethod της κλάσης Flow και έτσι ολοκληρώνει τη δημιουργία του Node-RED flow και επιστρέφει την περιγραφή της νέα μεθόδου σε προδιαγραφή OpenAPI specification.

Τέλος, το αντικείμενο flow στέλνεται στον διακομιστή Node-RED με HTTP POST request με τη χρήση της μεθόδου fetch. Η μέθοδος fetch παρέχεται από τη βιβλιοθήκη node-fetch. Όταν το Node-RED flow αναπτυχθεί επιτυχώς και ληφθεί θετική απάντηση από τον διακομιστή Node-RED, η μέθοδος επαναλαμβάνει τη διαδικασία για τις υπόλοιπες μεθόδους που έλαβε από το front-end.

Όταν η διαδικασία ολοκληρωθεί για όλες τις μεθόδους, η ολική περιγραφή του API ελέγχεται για τυχόν λάθη έναντι VDC blueprint Schema με τη χρήση της βιβλιοθήκης Ajv. Εφόσον δε βρεθούν λάθη, η περιγραφή του API τυπώνεται στην κονσόλα του διακομιστή (μόνο για σκοπούς ανάπτυξης) και επιστρέφεται status code 200 και μήνυμα επιτυχίας στο front-end.

functionNodes.js

Το αρχείο functionNodes.js, του οποίου ο βασικός κώδικας βρίσκεται στο Παράρτημα Β.4, ορίζει την κλάση FunctionNode. Τα αντικείμενα τύπου FunctionNode αντιπροσωπεύουν τα function nodes του Node-RED και χρησιμοποιούνται στην κλάση Flow για να κτίσουν το Node-RED flow.

Ο constructor της κλάσης αυτής παίρνει σαν ορίσματα ένα id και μια συμβολοσειρά με το όνομα func, η οποία περιέχει το κώδικα που θα τρέχει το function node του Node-RED. Αρχικοποιεί τα πεδία του id και func με τις τιμές αυτές, καθώς και διάφορα άλλα πεδία που χρειάζονται για την ομαλή λειτουργία του Node-RED. Τέλος, αρχικοποιείται το πεδίο wires με κενό array, το οποίο αντιπροσωπεύει τις συνδέσεις του node αυτού με τα υπόλοιπα αντικείμενα στο flow.

Η κλάση αυτή περιέχει ακόμα μια συνάρτηση, την addWire, η οποία παίρνει άγνωστο αριθμό ορισμάτων και τους προσθέτει στο array wires. Οι τιμές που μπαίνουν στο πεδίο wires είναι τα ids των στοιχείων με τα οποία συνδέεται.

httpNodes.js

Το αρχείο httpNode.js έχει την ίδια ιδιότητα με το προηγούμενο, δηλαδή ορίζει κλάσεις των οποίων τα αντικείμενα αντιστοιχούν σε nodes του Node-RED. Στη συγκεκριμένη περίπτωση, ορίζονται τα nodes τα οποία θα χειρίζονται τις λειτουργίες HTTP. Ο βασικός κώδικας βρίσκεται στο Παράρτημα Β.5.

Κατ’ αρχάς, ορίζεται η κλάση HttpNode που δεν αντιστοιχεί σε κάποιο αντικείμενο, αλλά χρησιμοποιείται ως βάση για να οριστούν οι υπόλοιπες κλάσεις. Ο constructor της παίρνει ως όρισμα ξανά ένα id και τον τύπο του node που αντιπροσωπεύει. Επίσης, περιέχει τη συνάρτηση addWire, η οποία έχει την ίδια λειτουργία με προηγουμένως.

Οι κλάσεις που ορίζονται είναι η HttpNodeIn, η οποία μοντελοποιεί nodes τύπου "http in", η HttpNodeResponse, η οποία μοντελοποιεί nodes τύπου "http response" και η HttpNodeRequest, η οποία μοντελοποιεί nodes τύπου "http request". Οι διαφορές στις ρυθμίσεις τους, οι οποίες μας ενδιαφέρουν στην υλοποίηση, είναι ελάχιστες και είναι κυρίως στο πεδίο type. Στα αντικείμενα τύπου HttpNodeIn έχουμε επιπλέον την παράμετρο url, που αντιπροσωπεύει τη διεύθυνση στην οποία ακούει το node αυτό.

flow.js

Στο αρχείο flow.js ορίζεται η κλάση Flow. Ο βασικός κώδικας βρίσκεται στο Παράρτημα Β.6. Τα αντικείμενα της κλάσης Flow μοντελοποιούν τα Node-RED flows. Ουσιαστικά είναι αντικείμενο τύπου JSON το οποίο περιέχει ένα array από nodes δηλαδή αντικείμενα τύπου FunctionNode και HttpNode.

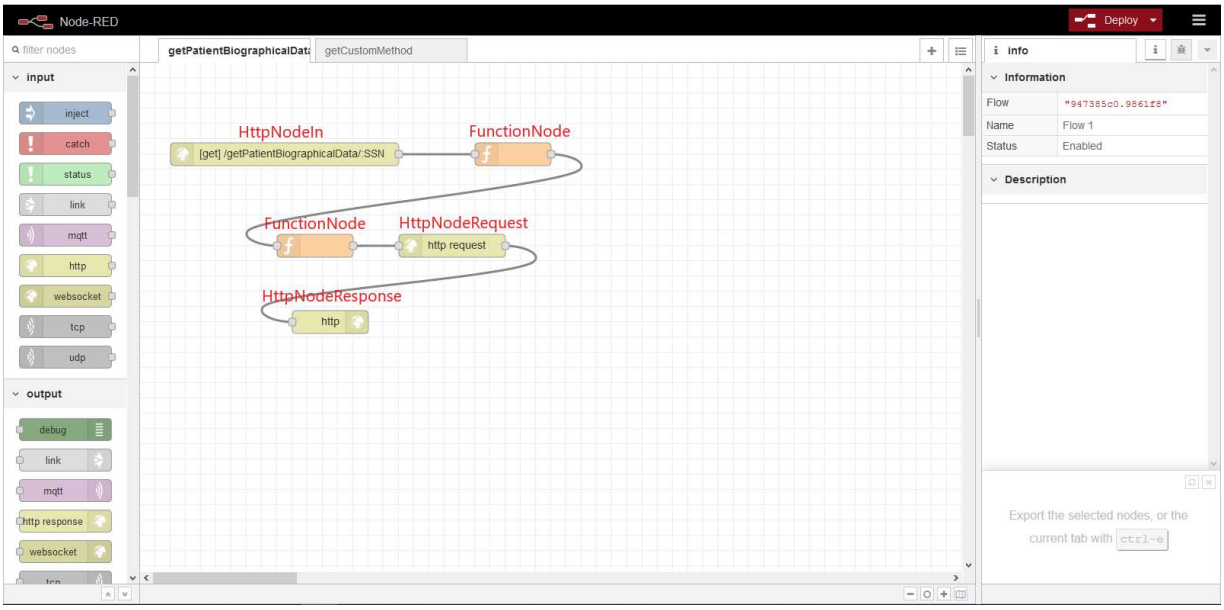
Όταν δημιουργείται ένα αντικείμενο αυτής της κλάσης τρέχει πρώτα ο constructor, ο οποίος παίρνει ως όρισμα μια συμβολοσειρά με το όνομα label. Ο constructor αρχικοποιεί το πεδίο label του αντικειμένου με το όρισμα που λαμβάνει και το πεδίο nodes με ένα κενό array.

Η κλάση έχει άλλες δύο μεθόδους, την addNode και την createFromMethod. Η μέθοδος addNode εισάγει ένα νέο node στο array nodes αναλαμβάνοντας να συνδέσει το νέο node με τα προηγούμενα με τη χρήση της μεθόδου addWire.

Ίσως η σημαντικότερη μέθοδος αυτής της κλάσης είναι η createFromMethod, η οποία παίρνει ως όρισμα μια από τις μεθόδους που λαμβάνονται από το front-end καθώς και την ενότητα EXPOSED_API του αρχικού VDC Blueprint, δημιουργεί ολόκληρο το Node-RED flow για αυτή τη μέθοδο και επιστρέφει την περιγραφή της σε OpenAPI specification. Για να γίνει αυτό αρχικά ελέγχεται αν η

μέθοδος είναι μια από αυτές που εμφανίζονται στο αρχικό VDC Blueprint. Σε αυτή την περίπτωση, δημιουργεί μια σειρά από πέντε nodes που σκοπό έχουν να δημιουργήσουν ένα περιτύλιγμα γύρω από την αρχική μέθοδο, δηλαδή καλείται απλά έμμεσα η αρχική μέθοδος και επιστρέφονται τα αποτελέσματά της, όπως φένεται στην Εικόνα 14. Τα πέντε αυτά nodes είναι:

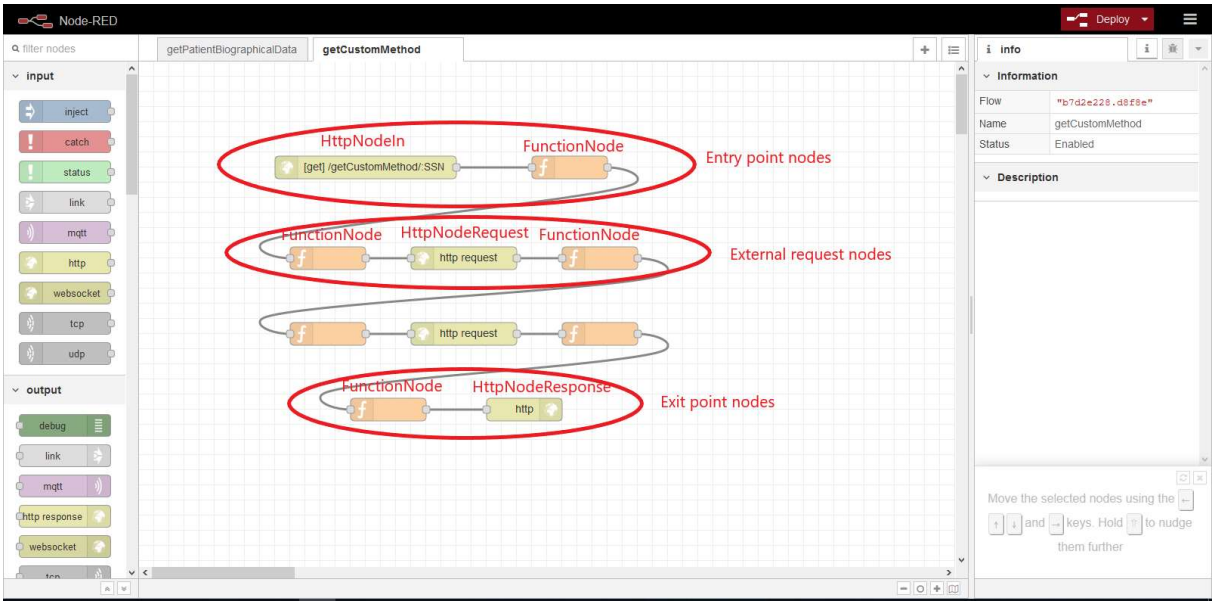
- HttpNodeIn: θέτει το endpoint (διεύθυνση) της νέας μεθόδου.
- FunctionNode: είναι υπεύθυνο για την αρχικοποίηση μεταβλητών που θα χρησιμοποιηθούν στη συνέχεια
- FunctionNode: είναι υπεύθυνο για την προετοιμασία του url της αρχικής μεθόδου
- HttpNodeRequest: εκτελεί το HTTP request στην αρχική μέθοδο.
- HttpNodeResponse: επιστρέφει τα αποτελέσματα από τη νέα μέθοδο.



Εικόνα 14: Παράδειγμα μεθόδου που υπάρχει στο Αρχικό VDC

Σε περίπτωση που μέθοδος, η οποία πήρε ως όρισμα η createFromMethod, είναι μέθοδος που κατασκεύασε ο χρήστης, τότε ακολουθείται διαφορετική διαδικασία. Κατ' αρχάς, χωρίζονται τα responses σε ομάδες ανάλογα με το από ποια αρχική μέθοδο προέρχονται. Στη συνέχεια δημιουργούνται τρείς ομάδες από nodes, όπως φένεται στην Εικόνα 15.

1. Entry point nodes: αρχικοποίει την μέθοδο
    a. HttpNodeIn: θέτει το endpoint (διεύθυνση) της νέας μεθόδου.
    b. FunctionNode: είναι υπεύθυνο για την αρχικοποίηση μεταβλητών που θα χρησιμοποιηθούν στη συνέχεια
2. External request nodes: αυτή η ομάδα είναι υπεύθυνη για να καλέσει μια από τις αρχικές μεθόδους. Δημιουργούνται τόσα σετ όσες και οι αρχικές μέθοδοι που πρέπει να κληθούν.
    a. FunctionNode: είναι υπεύθυνο για την προετοιμασία του url της αρχικής μεθόδου
    b. HttpNodeRequest: εκτελεί το HTTP request στην αρχική μέθοδο.
    c. FunctionNode: είναι υπεύθυνο να αποθηκεύσει τα αποτελέσματα από τη μέθοδο στις μεταβλητές του flow ώστε να είναι διαθέσιμα για επιστροφή στον χρήστη.
3. Exit point nodes: Σε αυτή την ομάδα συγκεντρώνονται όλα τα αποτελέσματα και επιστρέφονται στον χρήστη.
    a. FunctionNode: είναι υπεύθυνο για τη συλλογή των αποτελεσμάτων και τη δημιουργία του αντικειμένου που θα επιστραφεί στον χρήστη.
    b. HttpNodeResponse: επιστρέφει τα αποτελέσματα από τη νέα μέθοδο.

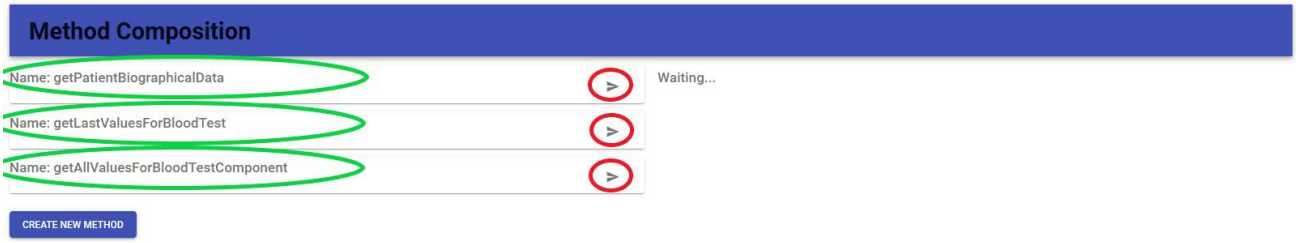


Εικόνα 15: Παράδειγμα μεθόδου που κατασκεύασε ο χρήστης

4.3 Παράδειγμα Λειτουργίας Συστήματος

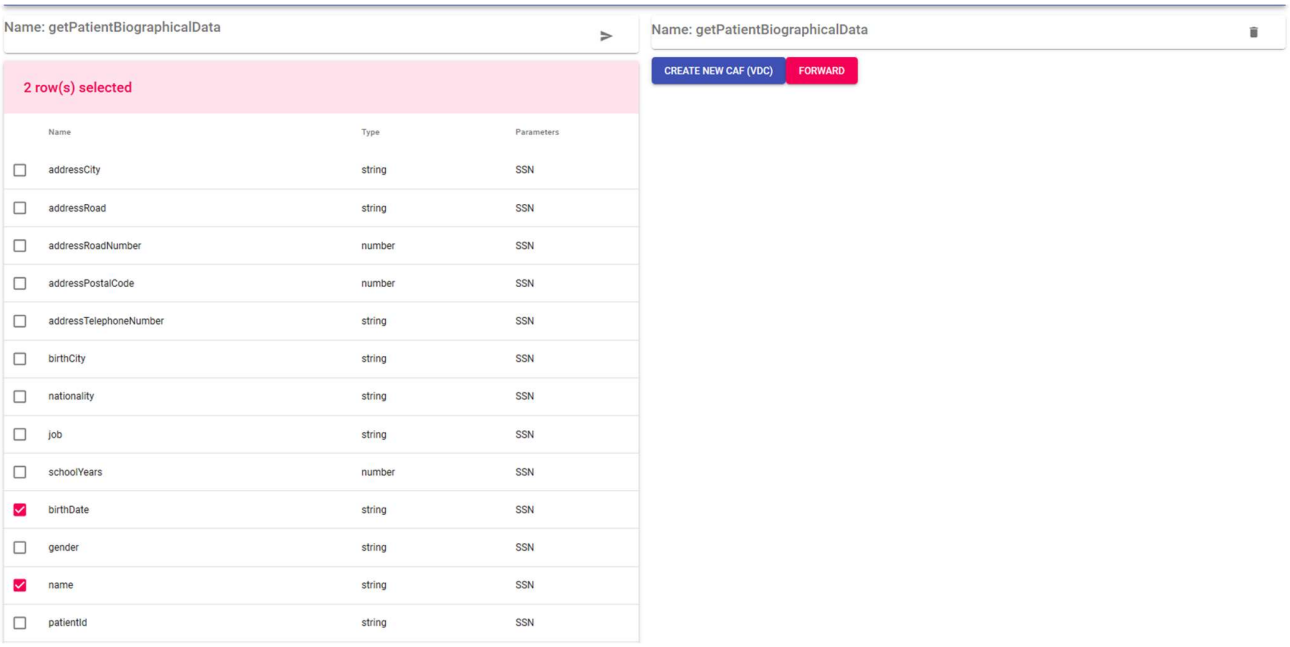
Στην παρούσα παράγραφο θα γίνει μια συνοπτική επίδειξη λειτουργίας του συστήματος από την εκκίνησή του με το επιλεγμένο VDC Blueprint μέχρι και την ανάπτυξη των νέων μεθόδων στον διακομιστή ανάπτυξης.

Μόλις ο χρήστης δρομολογηθεί στο σύστημα, βλέπει την Εικόνα 16, όπου μπορεί να επιλέξει μία ή περισσότερες από τις υπάρχουσες μεθόδους πατώντας οποιοδήποτε από τα βέλη που φαίνονται με κόκκινο κύκλο. Ακόμα, ο χρήστης μπορεί να κατασκευάσει τις δικές του μεθόδους επιλέγοντας δεδομένα από τις υπάρχουσες, πατώντας πάνω στα ονόματά τους (φαίνονται με πράσινο χρώμα).



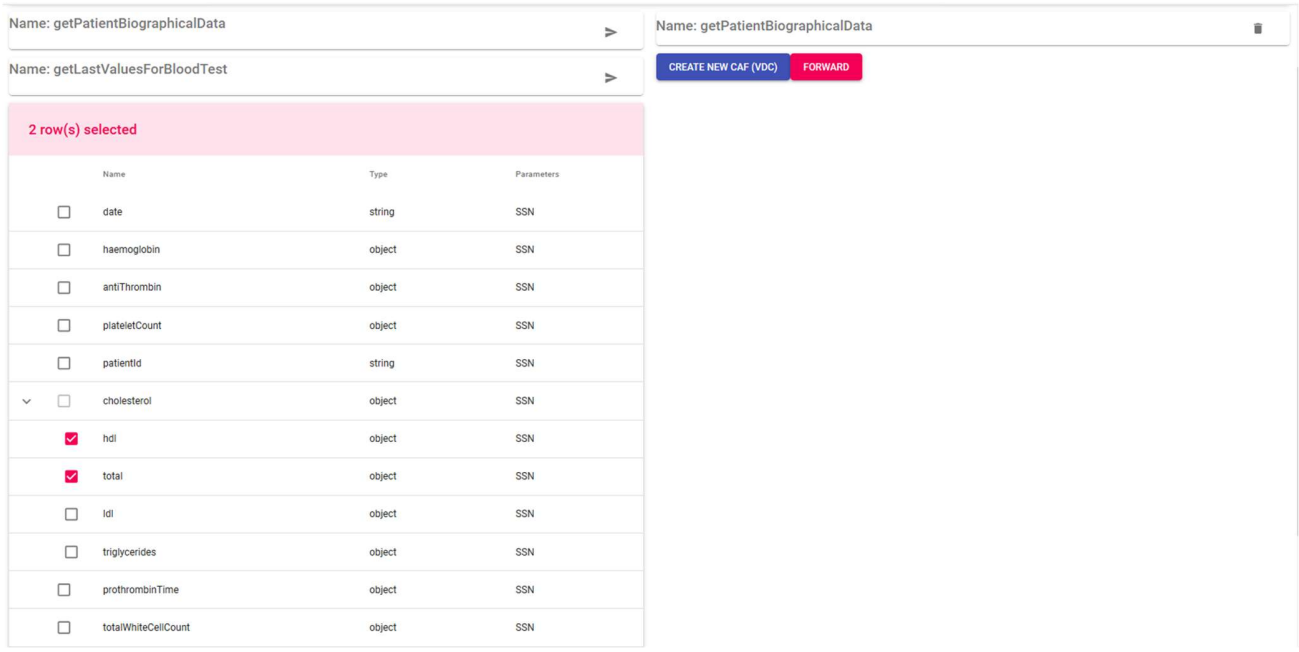
Εικόνα 16: Αρχική εικόνα συστήματος

Στο παρόν παράδειγμα, ο χρήστης επιλέγει ολόκληρη τη μέθοδο getPatientBiographicalData, πατώντας το τόξο μέσα στον πρώτο κόκκινο κύκλο. Στη συνέχεια, ξεκινά τη διαδικασία για να δημιουργήσει δική του μέθοδο πατώντας στον πρώτο πράσινο κύκλο, δηλαδή πάει να επιλέξει δεδομένα από τη μέθοδο getPatientBiographicalData, επιλέγοντας τα πεδία birthdate και name, όπως φαίνεται στην Εικόνα 17.



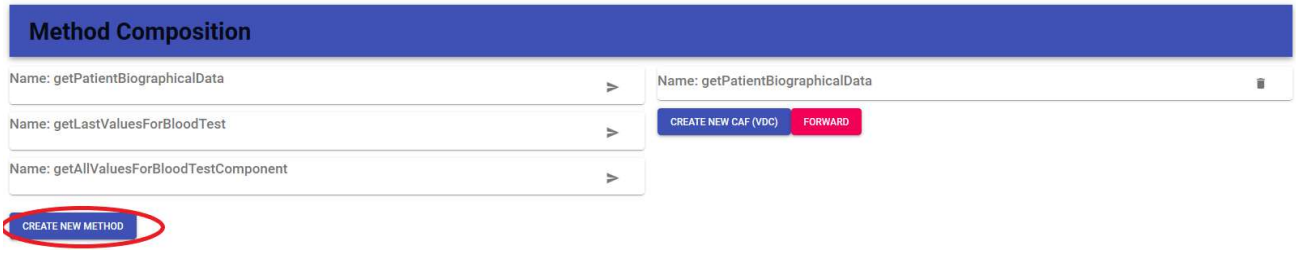
Εικόνα 17: Επιλογή πεδίων από την μέθοδο getPatientBiographicalData για την δημιουργία νέας μεθόδου

Στη συνέχεια, επιλέγονται τα πεδία hdl και total του αντικειμένου cholesterol από τη μέθοδο getLastValuesForBloodTest, όπως φαίνεται στην Εικόνα 18.



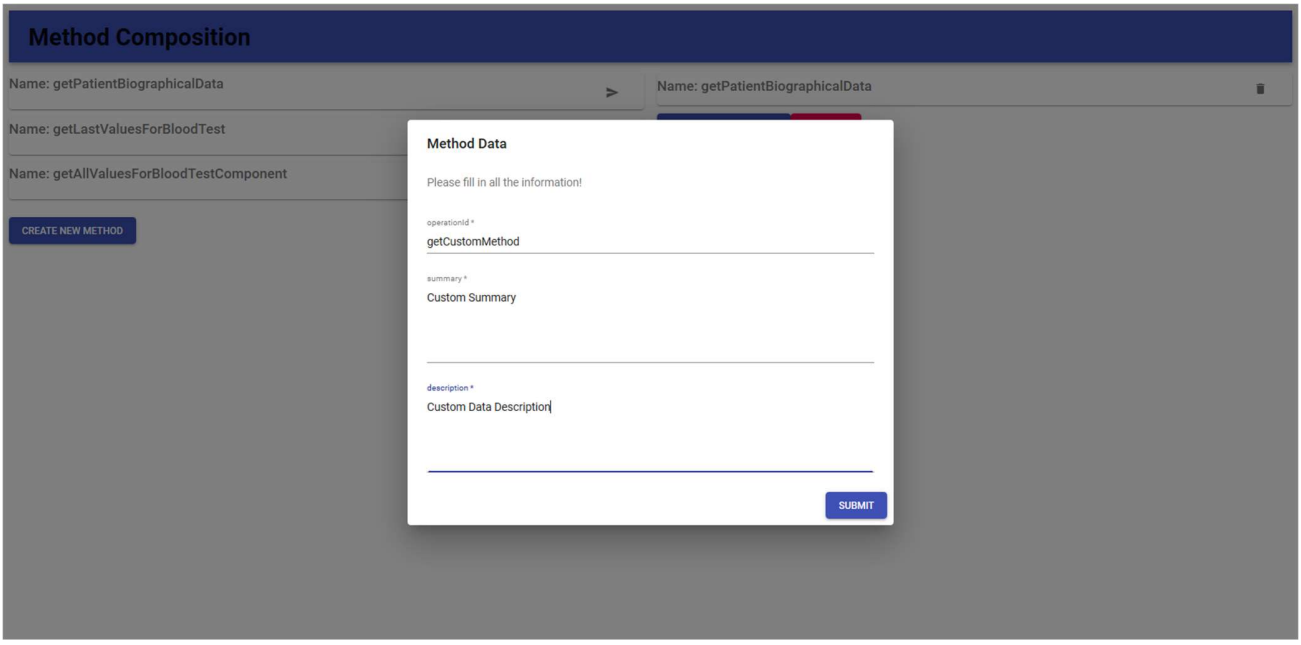
Εικόνα 18: Επιλογή πεδίων από την μέθοδο getValuesForBloodTest για την δημιουργία νέας μεθόδου

Ακολούθως, πατώντας ξανά στα ονόματα των μεθόδων, κλείνουν οι πίνακες και πατούμε το κουμπί Create New Method (βλ. Εικόνα 19).

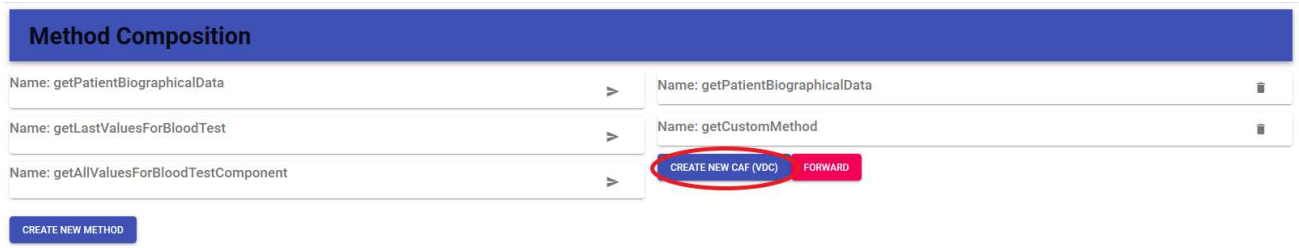


Εικόνα 19: Πατούμε το κουμπί Create New Method για την δημιουργία νέας μεθόδου

Αυτό ανοίγει τον διάλογο που φαίνεται στην Εικόνα 20, όπου συμπληρώνουμε τα απαιτούμενα στοιχεία για τη μέθοδο και πατάμε το κουμπί SUBMIT, το οποίο δημιουργεί τη νέα μέθοδο και τη μεταφέρει στη δεξιά μεριά της οθόνης, όπως φαίνεται στην Εικόνα 21.

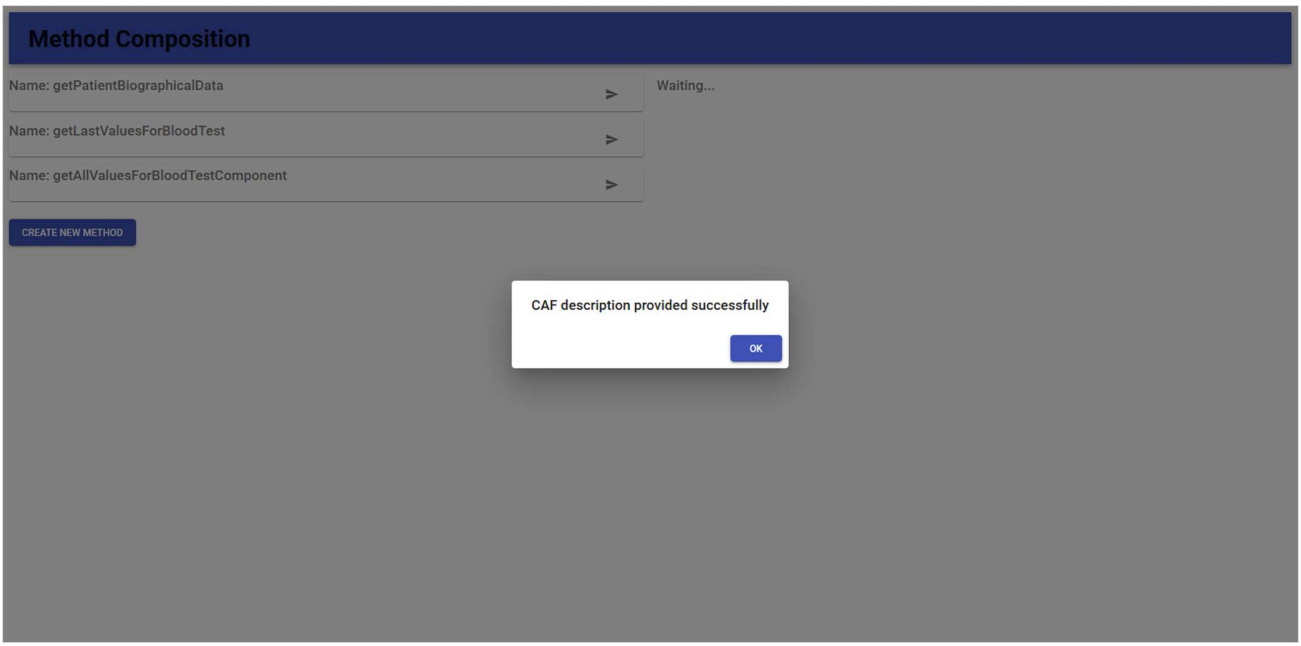


Εικόνα 20: Διάλογος για εισαγωγή στοιχείων νέας μεθόδου

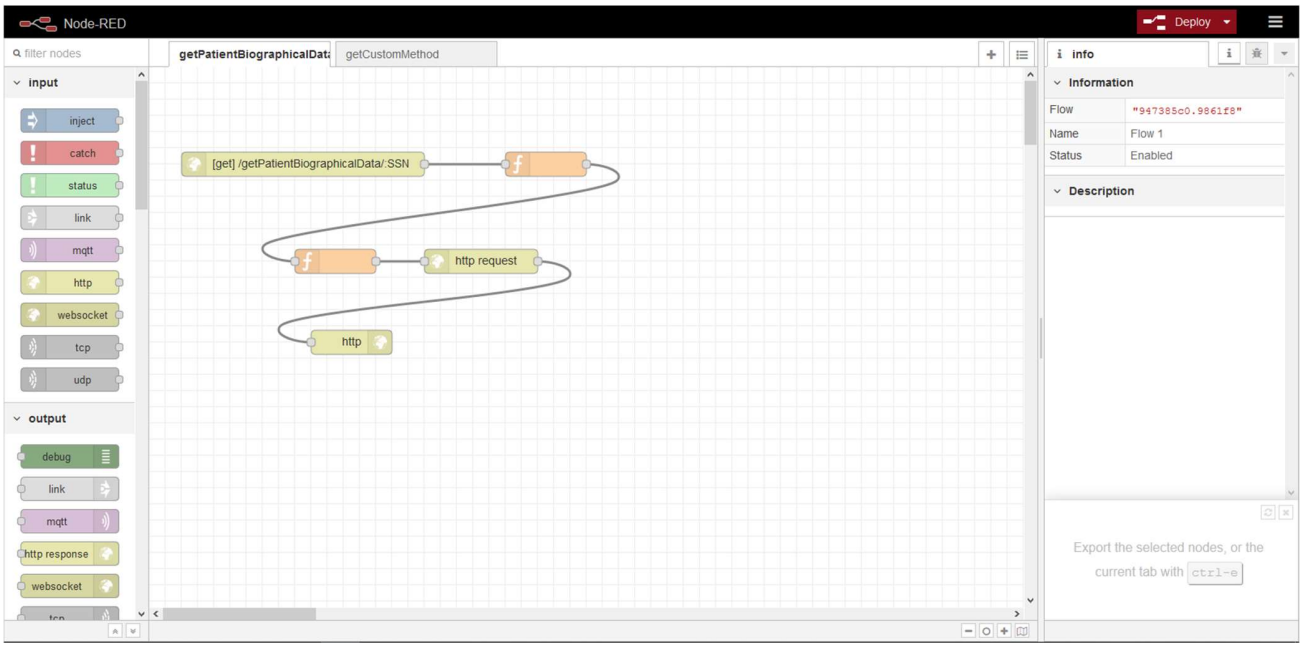


Εικόνα 21: Παρουσίαση επιλεγμένων μεθόδων στην αριστερή μεριά της οθόνης

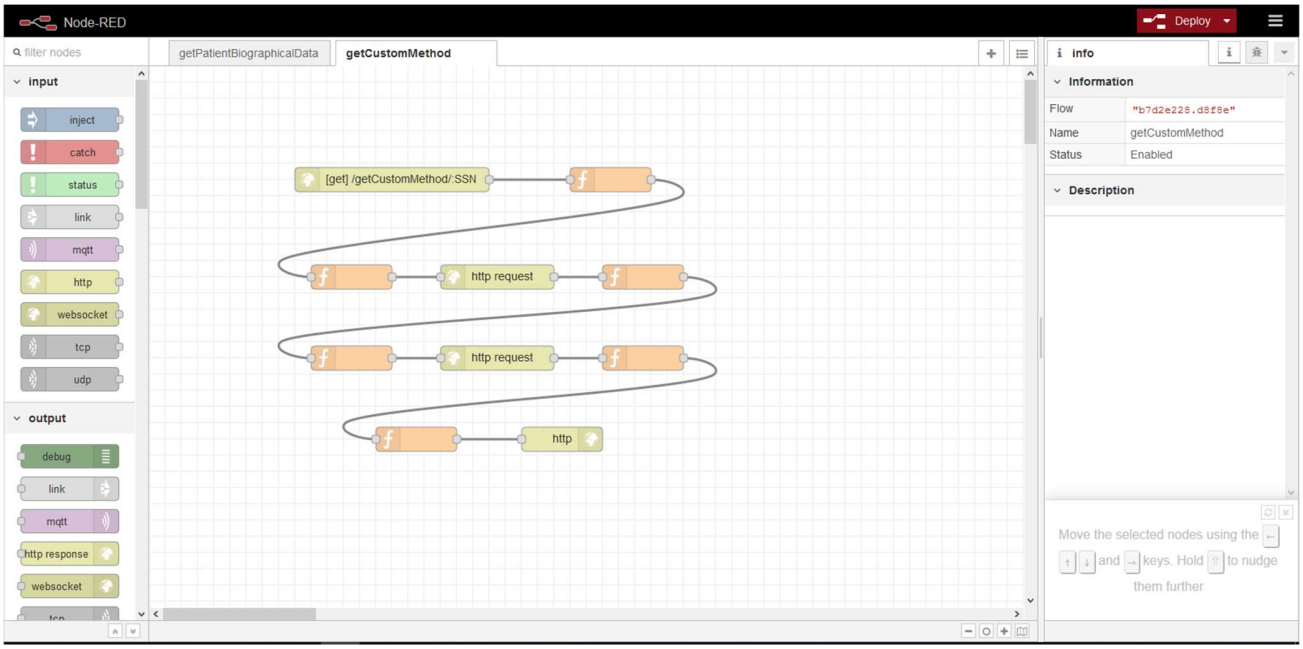
Έπειτα, πατάμε το κουμπί CREATE NEW CAF (VDC), το οποίο σημειώνεται με κόκκινο κύκλο στην Εικόνα 21. Σε περίπτωση επιτυχίας, παίρνουμε το μήνυμα της Εικόνας 22 και το VDC με τις νέες μεθόδους έχει γίνει deploy στο Node-RED. Τα νέα Node-RED flows που έγιναν deploy φένονται στην Εικόνα 23 και στην Εικόνα 24.



Εικόνα 22: Επιτυχία δημιουργίας νέου VDC



Εικόνα 23: Node-RED flow για την μέθοδο getPatientBiographicalData



Εικόνα 24: Node-RED flow για την μέθοδο getCustomMethod

Τέλος, δίνεται η ενότητα EXPOSED_API του νέου VDC, η οποία περιγράφει αναλυτικά τις νέες μεθόδους.

```
{
  "EXPOSED_API": {
    "openapi": "3.0.2",
    "info": {
      "title": "CAF API",
      "description": "Generated with method_manipulation and flow_generator",
      "version": "1.0.0"
    },
    "servers": [
      {
        "description": "NODERED server",
        "url": "http://localhost:1880"
      }
    ],
    "paths": {
      "/getPatientBiographicalData/:SSN": {
        "get": {
          "summary": "Get patient's biographical data",
          "description": "This method returns the biographical data for the
specified patient (identified via SSN), to be used by medical doctors",
          "operationId": "getPatientBiographicalData",
          "parameters": [
            {
              "name": "SSN",
              "in": "path",
              "description": "SSN of the patient",
              "required": true,
              "schema": {
                "type": "string",
                "example": "SLSDNA54P69R065W"
              }
            }
          ],
          "responses": {
            "200": {
              "content": {
                "application/json": {
                  "schema": {
                    "type": "object",
                    "properties": {
                      "addressCity": {
                        "type": "string"
                      },
                      "addressRoad": {
                        "type": "string"
                      },
```

```
    "addressRoadNumber": {
      "type": "number"
    },
    "addressPostalCode": {
      "type": "number"
    },
    "addressTelephoneNumber": {
      "type": "string"
    },
    "birthCity": {
      "type": "string"
    },
    "nationality": {
      "type": "string"
    },
    "job": {
      "type": "string"
    },
    "schoolYears": {
      "type": "number"
    },
    "birthDate": {
      "type": "string"
    },
    "gender": {
      "type": "string"
    },
    "name": {
      "type": "string"
    },
    "patientId": {
      "type": "string"
    },
    "socialId": {
      "type": "string"
    },
    "surname": {
      "type": "string"
    },
    "familyDoctorId": {
      "type": "string"
    }
  },
  "required": [
    "birthDate",
    "gender",
    "name",
```

```
                "patientId",
                "socialId",
                "surname",
                "familyDoctorId"
              ]
            }
          }
        }
      },
      "x-data-sources": [
        "patientBiographicalData"
      ]
    }
  },
  "/getCustomMethod/:SSN": {
    "get": {
      "summary": "Custom Summary",
      "description": "Custom Data Description",
      "operationId": "getCustomMethod",
      "parameters": [
        {
          "name": "SSN",
          "in": "path",
          "description": "SSN of the patient",
          "required": true,
          "schema": {
            "type": "string",
            "example": "SLSDNA54P69R065W"
          }
        }
      ],
      "responses": {
        "200": {
          "content": {
            "application/json": {
              "schema": {
                "type": "object",
                "properties": {
                  "birthDate": {
                    "type": "string"
                  },
                  "name": {
                    "type": "string"
                  },
                  "cholesterol": {
                    "type": "object",
```

```
                        "properties": {
                          "hdl": {
                            "type": "object"
                          },
                          "total": {
                            "type": "object"
                          }
                        }
                      }
                    }
                  }
                }
              }
            },
            "x-data-sources": [
              "patientBiographicalData",
              "bloodTestSource"
            ]
          }
        }
      }
    }
}
```

Κεφάλαιο 5 Σύνοψη, Περιορισμοί και Μελλοντικές Επεκτάσεις

Το κεφάλαιο αυτό παραθέτει μια γενική σύνοψη της διπλωματικής εργασίας και της πλατφόρμας που αναπτύχθηκε. Παρουσιάζονται επίσης μερικοί περιορισμοί και δυσκολίες που αντιμετωπίστηκαν κατά τη διάρκεια της έρευνας και ανάπτυξης, καθώς και κάποιες προτάσεις και ιδέες για μελλοντική ανάπτυξη της πλατφόρμας.

5.1 Σύνοψη

Ο σκοπός αυτής της διπλωματικής εργασίας ήταν η ανάπτυξη μιας πλατφόρμας, η οποία να δίνει στους χρήστες του DITAS Project τη δυνατότητα για σύνθεση μεθόδων (method composition) στα VDC Blueprints που έχουν πάρει από το DITAS Project, ώστε να πληρούν επακριβώς τις ανάγκες τους. Αρχικά, παρουσιάστηκε ο σκοπός αυτής της διπλωματικής και στη συνέχεια παρουσιάστηκαν οι τεχνολογίες στις οποίες βασίζεται. Ακολούθως, παρατέθηκαν τα διαθέσιμα εργαλεία που θα μπορούσαν να χρησιμοποιηθούν για την ανάπτυξη της συγκεκριμένης πλατφόρμας και επιλέχθηκαν τα κατάλληλα με βάση της ανάγκες και τις απαιτήσεις που υπήρχαν. Έπειτα, έγινε αναλυτική περιγραφή του τρόπου υλοποίησης των διάφορων κομματιών της πλατφόρμας και δόθηκε ένα παράδειγμα λειτουργίας.

Συνοπτικά, για την ανάπτυξη του γραφικού περιβάλλοντος, δηλαδή του front-end, χρησιμοποιήθηκε η βιβλιοθήκη ReactJS. Για την ανάπτυξη του back-end χρησιμοποιήθηκε το Node.js, ενώ τα καινούργια APIs που δημιουργούνται φιλοξενούνται σε έναν διακομιστή Node-RED. Ακόμα, στην υλοποίηση του front-end, του back-end αλλά και των νέων APIs χρησιμοποιήθηκαν διάφορες βιβλιοθήκες της JavaScript, οι σημαντικότερες από τις οποίες παρουσιάζονται στο Κεφάλαιο 3.4.

5.2 Περιορισμοί

Σε αυτή την παράγραφο παρουσιάζονται μερικοί από τους περιορισμούς που προέκυψαν, αλλά και κάποια προβλήματα που αντιμετωπίστηκαν κατά τη διάρκεια της έρευνας για την εκπόνηση αυτής της διπλωματικής εργασίας και της ανάπτυξης του κώδικα για την πλατφόρμα.

Το κυριότερο πρόβλημα ήταν η έλλειψη πηγών σχετικών με το θέμα της σύνθεσης μεθόδων (method composition). Η έρευνα έγινε με σκοπό να συνταχθεί ένα σύνολο κανόνων σε συνδυασμό με τους κανόνες λόγω υλοποίησης, οι οποίοι να διέπουν την διαδικασία σύνθεσης μεθόδων. Μέχρι τη στιγμή συγγραφής δεν έχει βρεθεί κάποιο δημοσίευμα το οποίο να παρουσιάζει κάποιους κανόνες. Ως αποτέλεσμα, εφαρμοστήκαν μόνο δύο κανόνες, όπως περιγράφηκαν στο Κεφάλαιο 4.1.

Ένας από τους σημαντικότερους περιορισμούς ήταν η ανάγκη τα νέα APIs να δημιουργούνται και να αναπτύσσονται σε πραγματικό χρόνο χωρίς την παρέμβαση ανθρώπου. Αυτό εξυπακούει ότι ο διακομιστής ο οποίος θα φιλοξενεί τα νέα APIs θα πρέπει να είναι σε θέση να αναπτύσσει νέο κώδικα χωρίς να χρειάζεται επανεκκίνηση. Η αντίθετη περίπτωση σημαίνει ότι όλα τα APIs τα οποία φιλοξενούνται στον διακομιστή δε θα ήταν διαθέσιμα κατά την ανάπτυξη ενός καινούργιου. Ακόμα, είναι φανερό ότι ο διακομιστής θα πρέπει να είναι σε θέση να δέχεται και να εκτελεί νέο κώδικα χωρίς ανθρώπινη παρέμβαση. Λύση σε αυτό το πρόβλημα ήρθε να δώσει, όπως περιγράφεται στο Κεφάλαιο 3.3, το εργαλείο Node-RED, καθώς παρέχει και τις δύο προαναφερθείσες απαιτήσεις.

Ένα άλλο πρόβλημα προέρχεται από τη χρήση του εργαλείου Node-RED ως του διακομιστή όπου θα φιλοξενούνται τα νέα APIs. Το Node-RED, για να λάβει και να εκτελέσει νέο κώδικα, πρέπει να τον λάβει υπό την μορφή ενός Node-RED flow, το οποίο είναι ένα JSON αρχείο με μία λίστα από Node-RED nodes, όπως εξηγείται και στο Κεφάλαιο 4.2. Προκύπτει έτσι η ανάγκη για τη δημιουργία αυτού του αρχείου δυναμικά, ώστε να περιέχει την κατάλληλη λίστα από Node-RED nodes, για να υλοποιούν το εκάστοτε API. Το πρόβλημα αυτό λύθηκε αξιοποιώντας την αντικειμενοστραφή φύση της JavaScript, δηλαδή δημιουργώντας κλάσεις αντικειμένων που, περιγράφουν επακριβώς τα διάφορα Node-RED nodes που θα χρειαστούμε. Αυτό επέτρεψε την εύκολη δημιουργία των αρχείων

αυτών, αφού πλέον μπορούμε να τα δημιουργήσουμε ως μια λίστα αντικειμένων από τις προαναφερθείσες κλάσεις.

5.3 Μελλοντικές Επεκτάσεις

Στο παρόν υποκεφάλαιο θα παρουσιαστούν πιθανές βελτιώσεις και ιδέες για μελλοντική ανάπτυξη της πλατφόρμας.

Αρχικά, η πλατφόρμα στην παρούσα φάση υποστηρίζει μόνο VDC Blueprints, των οποίων τα APIs αποτελούνται από μεθόδους τύπου HTTP GET. Πολλά APIs όμως υποστηρίζουν και μεθόδους τύπου HTTP POST σε περιπτώσεις όπου επιτρέπεται μεγάλη παραμετροποίηση των δεδομένων. Συμπερασματικά, μια πιθανή βελτίωση της πλατφόρμας θα ήταν η υποστήριξη μεθόδων τύπου HTTP POST, καθώς και η δημιουργία κάποιας τυποποιημένης μορφής για τις μεθόδους τύπου HTTP POST που θα υποστηρίζονται, αφού στη γενική περίπτωση δεν μπορούν να ληφθούν υπόψη όλες οι πιθανές μορφές.

Επιπλέον, μια ιδέα για μελλοντική επέκταση είναι η δημιουργία ενός συστήματος προτάσεων, το οποίο θα προτείνει στον χρήστη συνδυασμούς μεθόδων καθώς και νέες μεθόδους, βασισμένο στις μέχρι τώρα επιλογές του αλλά και στις επιλογές προηγούμενων χρηστών που δούλεψαν με το συγκεκριμένο VDC Blueprint.

Ακόμα μια μελλοντική επέκταση θα ήταν να δίνεται στον χρήστη η δυνατότητα για μια εικονική χρήση του νέου API. Δηλαδή, ο χρήστης να μπορεί να τρέξει δοκιμαστικά το νέο API πριν τη δημιουργία του, έτσι ώστε να ελέγξει ότι πράγματι τα δεδομένα επιστρέφονται όπως τα θέλει και να κάνει τις απαραίτητες αλλαγές πριν την οριστική δημιουργία του API.

Τελευταία, αλλά όχι έσχατη βελτίωση της πλατφόρμας θα ήταν στα νέα API να λαμβάνονται υπόψη οι εναλλακτικοί διακομιστές των αρχικών. Πιο συγκεκριμένα, κάποιες φορές στο VDC Blueprint του αρχικού API αναφέρονται περισσότεροι από ένας διακομιστές οι οποίοι λειτουργούν ως δικλείδα

ασφαλείας σε περίπτωση που ο κυρίως διακομιστής δεν είναι διαθέσιμος. Στην παρούσα φάση, όταν δημιουργείται ένα νέο API, λαμβάνεται υπόψη μόνο ο κύριος διακομιστής του αρχικού API. Ως εκ τούτου, σε περίπτωση που αυτός δεν είναι διαθέσιμος δε θα είναι διαθέσιμο ούτε το νέο API. Καταλήγοντας, μια μελλοντική βελτίωση θα ήταν, κατά τη δημιουργία των νέων APIs, να λαμβάνονται υπόψη όλοι οι διακομιστές που αναφέρονται στο αρχικό, έτσι ώστε να εξασφαλίζεται υψηλότερη διαθεσιμότητα.

Κεφάλαιο 6 Βιβλιογραφία

[1] “DITAS PROJECT - H2020.” [Online]. Available: https://www.ditas-project.eu/. [Accessed: 21-Jun-2019].

[2] C. C. (POLIMI) Ronen Kat (IBM), Oliver Barreto (Atos), Aitor Fernández Gómez(IDEKO), Vrettos Moulos (ICCS), Jose Antonio Sanchez (Atos), Achilleas Marinakis (ICCS), David Bermbach (TUB), Marco Peise (TUB), Andrea Micheletti (OSR), Peter Gray (CS), Maya Anderson (IBM), M, “D1.1 Initial architecture docu- ment with market analysis, SotA refresh and validation approach Project,” no. 731945, 2020.

[3] P. Christensson, “Web Browser Definition,” 2014. [Online]. Available: https://techterms.com/definition/web_browser. [Accessed: 22-Jun-2019].

[4] “What is Hypertext Markup Language (HTML)? - Definition from Techopedia.” [Online]. Available: https://www.techopedia.com/definition/1892/hypertext-markup-language-html. [Accessed: 23-Jun-2019].

[5] “The elements of HTML.” [Online]. Available: https://html.spec.whatwg.org/multipage/semantics.html#semantics. [Accessed: 08-Jul-2019].

[6] “HTML & CSS - W3C.” [Online]. Available: https://www.w3.org/standards/webdesign/htmlcss. [Accessed: 23-Jun-2019].

[7] “Using CSS animations - CSS: Cascading Style Sheets | MDN.” [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Animations/Using_CSS_animations. [Accessed: 23-Jun-2019].

[8] “JavaScript | MDN.” [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/JavaScript. [Accessed: 23-Jun-2019].

[9] “About JavaScript - JavaScript | MDN.” [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/JavaScript/About_JavaScript. [Accessed: 23-Jun-2019].

[10] ECMA International, “The JSON Data Interchange Syntax COPYRIGHT PROTECTED DOCUMENT,” 2017.

[11] Roy Thomas Fielding, “Fielding Dissertation: CHAPTER 5: Representational State Transfer (REST),” *Architectural Styles and the Design of Network-based Software Architectures*, 2000. [Online]. Available: http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm.

[Accessed: 08-Feb-2016].

[12] “OAI/OpenAPI-Specification: The OpenAPI Specification Repository.” [Online]. Available: https://github.com/OAI/OpenAPI-Specification. [Accessed: 24-Jun-2019].

[13] “raml-spec/raml-10.md at master · raml-org/raml-spec.” [Online]. Available: https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md#introduction. [Accessed: 24-Jun-2019].

[14] “API Blueprint | API Blueprint.” [Online]. Available: https://apiblueprint.org/. [Accessed: 25-Jun-2019].

[15] Z. Zheng, J. Zhu, and M. R. Lyu, “Service-Generated Big Data and Big Data-as-a-Service: An Overview,” in *2013 IEEE International Congress on Big Data*, 2013, pp. 403–410.

[16] S. Ceri, P. Fraternali, A. Bongio, M. Brambilla, S. Comai, and M. Matera, “Architecture Design,” *Des. Data-Intensive Web Appl.*, no. 731945, pp. 329–360, 2003.

[17] G. M. (POLIMI) Alexandros Psychas, Achilleas Marinakis, George Chat-zikyriakos (ICCS), David García Pérez, Jose Antonio Sanchez (ATOS), Marco Peise, Sebastian Werner (TUB), Maya Anderson (IBM), Mattia Salnitri, “D3.2 Data Virtualization SDK prototype (initial version),” no. 731945, 2020.

[18] “Top JavaScript Frameworks For 2019 | LambdaTest.” [Online]. Available: https://www.lambdatest.com/blog/top-javascript-frameworks-for-2019/. [Accessed: 26-Jun-2019].

[19] “The world’s most popular React UI framework - Material-UI.” [Online]. Available: https://material-ui.com/. [Accessed: 28-Jun-2019].

[20] “material-table.” [Online]. Available: https://material-table.com/#/. [Accessed: 28-Jun-2019].

[21] “Express - Node.js web application framework.” [Online]. Available: https://expressjs.com/. [Accessed: 28-Jun-2019].

[22] “node-fetch - npm.” [Online]. Available: https://www.npmjs.com/package/node-fetch. [Accessed: 28-Jun-2019].

Παράρτημα Α Κώδικας Front-End

Σε όποια σημεία του κώδικα χρησιμοποιείται το σύμβολο «//….» έχει παραληφθεί κώδικας για διευκόλυνση της κατανόησης. Πλήρης κώδικας του front-end υπάρχει στην διεύθυνση https://github.com/KyriacosP/method_manipulation

Α.1 Κώδικας App Component

```
class App extends React.Component {

  //app entry point
  render(){
    return(
      <ThemeProvider theme={createMuiTheme(theme)}>
        <Header/>
        <br/>
        <MethodsDisplay/>
      </ThemeProvider>
    )
  }
}

export default App;
```

A.2 Κώδικας MethodsDisplay Component

```
constructor(){
    super();
    this.state={
      //....
      apiDesc:{},
      selection:[],
      selectedMethods:[],
      //....
      form:{
        //....
      },
      selectedResponses:{}
    };
  }


//Retrieves the api specification and removes all the references using $RefParser
componentDidMount(){
  //.....
  $RefParser.dereference(postResponse.EXPOSED_API)
  .then(schema=>{this.setState({apiDesc: new ApiSpec(schema)})})
  .catch(err=>console.log(err));
  //.....
}


render(){
  //.....
  const elems=[];
  for (var m in this.state.apiDesc.methods){
    elems.push(<MethodContainer key={m} id={m}
handleMethodSelection={this.handleMethodSelection}
updateGlobalSelection={this.updateGlobalSelection}
method={this.state.apiDesc.methods[m]}/>);
  };
  return(
    <React.Fragment>
      //....
      {elems}
      <br/>
      <Button variant="contained" color="primary"
onClick={this.createNewMethod}>Create New Method</Button>
      //....
      <SelectedMethodsDisplay methods={this.state.selectedMethods}
handleDelete={this.handleDelete} clearSel={this.clearSel}/>
```

```
      //....
      <Dialog
        open={this.state.open}
        onClose={this.handleClose}
        fullWidth
      >
        <DialogTitle id="form-dialog-title">Method Data</DialogTitle>
          <DialogContent>
              <DialogContentText>
                Please fill in all the information!
              </DialogContentText>
              <form>
                //....
              </form>
          </DialogContent>
          <DialogActions>
            <Button onClick={this.handleClose} variant="contained" color="primary">
              Submit
            </Button>
          </DialogActions>
      </Dialog>
      //....
    </React.Fragment>
  );
}


//handles whole method selections from the MethodContainer component
handleMethodSelection=(methodSelected)=>{
  this.setState(prevState=>{
    let tmp=prevState.selectedMethods;
    tmp.push(methodSelected);
    return {selectedMethods:tmp};
  });
}


//handles state.selection updates from the MethodResponseGrid component
updateGlobalSelection=(id,sel)=>{
  this.setState(prevState=>{
    prevState.selection[id]=sel;
    return {selection:prevState.selection}
  })
}
```

```
//handles method delete from the SelectedMethodsDisplay component
handleDelete=(m)=>{
  this.setState(prevState=>{
    prevState.selectedMethods.splice(m,1);
    return {selectedMethods:prevState.selectedMethods};
  })
}


//clears state after a new CAF is created it's called from the
SelectedMethodsDisplay component
clearSel=()=>{
  this.setState({selectedMethods:[]});
}


//Dialog Close
//also handles the new method creation and states updates
handleClose = () => {
  this.setState({ open: false });
  let desc={};
  //....

desc.get.responses['200'].content['application/json'].schema=this.state.selectedRes
ponses;
  this.setState(prevState=>{
    let tmp=prevState.selectedMethods;
    tmp.push(new Method(prevState.form.path,desc));
    return {
      selectedMethods:tmp,
      form:{
        //....
      }
    };
  })
}
```

```
//its called when the Create new Method button is pressed
//prepares the selection array (removes duplicates and fixes parent children
duplicates due to material-table bug)
//checks the selection against a set of rules
createNewMethod=()=>{
  let sel=this.state.selection;
  let tmp=[];
  for(var i in sel){
    for(var j in sel[i]){
      tmp.push(sel[i][j]);
      //....
    }
  }
  //....
  let {pass,error}=this.applyRules(tmp);
  if(pass){
    this.setState({
      selectedResponses:{...tmp},
      selection:[]
    });
    this.handleOpen();
  }
  else {
    this.setState({
      selectedResponses:{},
      selection:[],
      //....
    });
  }
}


applyRules=(array)=>{
  //first rule: in a oneOf type methods only selections from the same schema are
allowed
  let fra=array.map(x=>x.path).filter(x=>x.includes('oneOf'));
  fra=fra.map(x=>x.split(".")[1]);
  for(let i in fra)
  {
    if(fra[0]!==fra[i])
    {
      return {pass:false,error:"Select properties only from a single Schema in
methods of type oneOf"};
    }
  }
```

```
  //second rule: no selections with the same path from different methods are
allowed (implementation constrain)
  let sra=array.map(x=>x.path);
  const dupes = sra.reduce((acc, v, i, arr) => arr.indexOf(v) !== i &&
acc.indexOf(v) === -1 ? acc.concat(v) : acc, [])
  if(dupes.length!==0){
    return {pass:false,error:"Properties with the same path have been selected:
"+dupes.join(", ")};
  }
  //return pass true if both test have passed
  return {pass:true,error:""};
}
```

A.3 Κώδικας MethodContainer Component

```
constructor(props){
  super(props);
  this.state={
    isHidden:true,
  }
}

//this method is used as a midlewear between MethodsDisplay and MethodResponseGrid
for the updateGlobalSelection method
updateState=(sel)=>{
  this.props.updateGlobalSelection(this.props.id,sel);
}

//handles the visibility of the MethodResponseGrid component
handleClick=()=>{
  this.setState(prevState => {return {isHidden:!prevState.isHidden}});
}

//this function is used as a midlewear between MethodsDisplay and MethodContainer
for the handleMethodSelection method
handleMethodSelection=(event)=>{
  this.props.handleMethodSelection(this.props.method)
}

render(){
  return(
    <React.Fragment>
      <Paper style={stylePaper}>
        <Typography style={{display:"inline-block",width:"90%"}} variant="h6"
color="textSecondary" onClick={this.handleClick} noWrap >
          Name: {this.props.method.operationId}
        </Typography>
        <IconButton style={{display:"inline-block",width:"10%"}}
onClick={this.handleMethodSelection}>
          <Send fontSize="small" />
        </IconButton>
      </Paper>
      <MethodResponseGrid method={this.props.method} updateState={this.updateState}
hidden={this.state.isHidden}/>
    </React.Fragment>
  );
}
```

A.4 Κώδικας MethodResponseGrid Component

```
constructor(props){
    super(props);
    this.state={
      rows:[]
    };
};


//recursive method to handle nested objects
addProperties(properties,rows,path,parentId,par,operationId){
  let j=parentId;
  for(let i in properties){
    j++;
    rows.push({id:j, name: i,path:path+"."+i, type: properties[i].type,
parameters:par, partof:operationId,parentId:parentId});
    if(Object.prototype.hasOwnProperty.call(properties[i], 'properties')){

j=this.addProperties(properties[i].properties,rows,path+"."+i,j,par,operationId);
    }
    else if(properties[i].type==="array")
    {

j=this.addProperties(properties[i].items.properties,rows,path+"."+i,j,par,operation
Id);
    }
  }
  return j;
}


//prepares top level responses for the MaterialTable calls addProperties for nested
objects
componentDidMount(){
  var method=this.props.method;
  var par=[];
  for (var p in method.parameters){
    par.push(method.parameters[p]);
  };
  var rows=[];
  var schema=method.responseSchema;
  if(Object.prototype.hasOwnProperty.call(schema, 'oneOf')){
    let j=1;
    rows.push({id:j, name: "oneOf", path: "oneOf", type: "oneOf", parameters:par,
partof:method.operationId});
    j++
```

```
    for(let i in schema.oneOf){
      rows.push({id:j, name: "schema: "+i, path: "oneOf.schema"+i, type: "oneOf",
parameters:par, partof:method.operationId,parentId:1});
      if(schema.type === "object"){

j=this.addProperties(schema.oneOf[i].properties,rows,"oneOf.schema"+i+".",j,par,met
hod.operationId);
      }
      else{

j=this.addProperties(schema.oneOf[i].items.properties,rows,"oneOf.schema"+i+".Array
",j,par,method.operationId);
      }
      j++;
    }

  }else if(schema.type === "object"){
    let j=1;
    for(let i in schema.properties){
      rows.push({id:j, name: i, path: i, type: schema.properties[i].type,
parameters:par, partof:method.operationId});
      if(Object.prototype.hasOwnProperty.call(schema.properties[i], 'properties')){

j=this.addProperties(schema.properties[i].properties,rows,i,j,par,method.operationI
d);
      }
      j++;
    }
  }else{
    rows.push({id:1, name: "", path:"Array", type: schema.type, parameters:par,
partof:method.operationId});

this.addProperties(schema.items.properties,rows,"Array",1,par,method.operationId);
  }
  this.setState({rows:rows});
}
```

```
render() {
    if(!this.props.hidden){
      return (
        <ThemeProvider theme={createMuiTheme(theme)}>
          <MaterialTable
            title={
              <div>
                <Typography variant="h6" color="textSecondary" >
                  Method: {this.props.method.type}  Path: {this.props.method.path}
                </Typography>
              </div>
              }
            data={this.state.rows}
            columns={[
              { title: 'Name', field: 'name' },
              { title: 'Type', field: 'type' },
              {
                title: 'Parameters',
                field: 'parameters',
                render: rowData=>rowData.parameters.map(x=>x.name).join(", ")
              }
            ]}
            parentChildData={(row, rows) => rows.find(a => a.id === row.parentId)}
            options={{
              selection: true,
              selectionProps: rowData => ({
                disabled: rowData.type === 'oneOf' ||
rowData.tableData.childRows!==null,
                color: 'secondary'
              }),
              showSelectAllCheckbox:false,
              grouping:false,
              sorting:false,
              search: false,
              paging: false
            }}
            onSelectionChange={(rows) => this.props.updateState(this.unique(rows))}

          />
        </ThemeProvider>
      )
    }else{
      return null;
    }
  }
```

A.5 Κώδικας SelectedMethodsDisplay Component

```
constructor(){
  super();
  this.state={
    open: false,
    msg: ""
  };
}


createCAF=() => {
  fetch('http://localhost:5000/generate', {
    method: 'POST',
    headers: {
      'Accept': 'application/json',
      'Content-Type': 'application/json'
    },
    body: JSON.stringify({methods:this.props.methods, exposedAPI:
postResponse.EXPOSED_API})
  })
  .then(res=>{
        if(!res.ok){
         throw Error(res);
    }
    return res;
  })
  .then(data=>data.json())
  .then(data=>this.setState({open:true,msg:data.msg}))
  .then(data=>this.props.clearSel())
  .catch(error =>{console.log(error);this.setState({open:true,msg:"error"});});
};


//handles selected method deletion calls handleDelete from MethodsDisplay
handleDelete=(id)=>{
  this.props.handleDelete(id);
}
```

```
render(){
  const elems=[];
  for (var m in this.props.methods){
    elems.push(
        <Paper key={m} style={stylePaper}  >
            <Typography style={{display:"inline-block",width:"90%"}} variant="h6"
color="textSecondary">
              Name: {this.props.methods[m].operationId}
            </Typography>
            <IconButton style={{display:"inline-block",width:"10%"}}
onClick={this.handleDelete.bind(this, m)}>
              <Delete fontSize="small" />
            </IconButton>
        </Paper>);
  }
  var res;
  if(elems.length !== 0) {
    res= (
      <React.Fragment>
        {elems}
        <Button variant="contained" color="primary" onClick={this.createCAF}>Create
New CAF (VDC)</Button>
        <Button variant="contained" color="secondary" onClick={this.handleSubmit}
>Forward</Button>
      </React.Fragment>
    )
  } else {
    res= (
      <Typography variant="h6" color="textSecondary">
        Waiting...
      </Typography>
    )
  }
  return (
    <React.Fragment>
      {res}
      <Dialog
        open={this.state.open}
        onClose={this.handleClose}
      >
        //....
      </Dialog>
      </React.Fragment>
    );
  }
```

Παράρτημα Β Κώδικας Back-End

Σε όποια σημεία του κώδικα χρησιμοποιείται το σύμβολο «//….» έχει παραληφθεί κώδικας για διευκόλυνση της κατανόησης. Πλήρης κώδικας του back-end υπάρχει στην διεύθυνση https://github.com/KyriacosP/flow_generator

Β.1 Κώδικας App.js

```
import express from 'express';
import bodyParser from 'body-parser';
import router from './routes/index.js';
import cors from 'cors';
//....

const PORT = 5000;
const app = express();

//server entry point
app.use(cors());
//....
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended: false }));
//use the routes defined in ./routes/index.js
app.use(router);

app.listen(PORT, () => {
  console.log(`server running on port ${PORT}`)
});
```

B.2 Κώδικας Routes/index.js

```
import express from 'express';
import flowsController from '../flowsControllers/flows';

const router = express.Router();

//routes definition
router.get('/', flowsController.info);
router.post('/generate', flowsController.generateFlow);
//....

export default router;
```

B.3 Κώδικας flows.js

```
import fetch from 'node-fetch';
import {HttpNodeIn,HttpNodeResponse,HttpNodeRequest} from '../nodes/httpNodes';
import FunctionNode from '../nodes/functionNodes';
import Flow from '../nodes/flow';
import Ajv from 'ajv';
import {blueprintSchema,blueprintSchemaExposedAPI} from '../schemas'

class FlowsController {
  //endpoint implementation for / and /info
  //return status and msg descriping functionality
  info(req, res) {
    return res.status(200).send({
      success: 'true',
      msg: 'Rest API for generating NODE-RED flows\nBuild for and works only with
Method_manipulation FronEnd!'
    });
  }

  //endpoint implementation for /generate
  //creates NODERED flows and posts them to NODERED
  //returns status and openapi spec
  async generateFlow(req, res) {
    if(Object.keys(req.body).length === 0){
      return res.status(400).send({
        success: 'false',
        msg: 'No CAF description provided'
```

```
    });
  } else {
    //openapi spec initialization
    let newOpenAPi={};
    newOpenAPi.EXPOSED_API={
      openapi:"3.0.2",
      info:{
        title:"CAF API",
        description:"Generated with method_manipulation and flow_generator",
        version:"1.0.0"
      },
      servers:[
        {
          description:"NODERED server",
          url:"http://localhost:1880"
        }
      ],
      paths:{}
    };
    //flows creation and posts
    let {methods, exposedAPI}=req.body;
    let noderedResponse=[];
    for(var i in methods){
      //initialize flow
      let f=new Flow(methods[i].operationId);
      //create flow from method
      let path=f.createFromMethod(methods[i],exposedAPI);

newOpenAPi.EXPOSED_API.paths[Object.keys(path)[0]]=path[Object.keys(path)[0]];
      // post flow to NODERED
      await fetch('http://localhost:1880/flow', {
        method: 'POST',
        headers: {
          'Accept': 'application/json',
          'Content-Type': 'application/json'
        },
        body: JSON.stringify(f)
      })
      .then(res=>{
        if(!res.ok){
          throw Error(res.json());
        }
        return res;
      })
      .then(data=>data.json())
      .then(data=>noderedResponse.push(data))
      .catch(error =>noderedResponse.push(error));
```

```
      }
      //validate new openapiSpec against schema
      let ajv=new Ajv();
      let valid=ajv.validate(blueprintSchemaExposedAPI,newOpenAPi);
      console.log(valid);
      console.log(JSON.stringify(newOpenAPi,null,2));
      return res.status(200).send({
          success: 'true',
          msg: 'CAF description provided successfully',
          noderedResponse:noderedResponse,
          openapiSpec:newOpenAPi
      })
    }
  }

  //....

}


const flowsController = new FlowsController();
export default flowsController;
```

B.4 Κώδικας functionNodes.js

```
class FunctionNode {
  constructor(id,func){
    this.id=id;
    this.type="function";
    this.func=func;
    //....
    this.wires=[];
  }

  addWire(){
    let wire=[];
    for (var i in arguments){
      wire.push(arguments[i]);
    }
    this.wires.push(wire);
  }
}

export default FunctionNode;
```

Β.5 Κώδικας httpNodes.js

```
class HttpNode {
  constructor(id,type){
    this.id=id;
    this.type=type;
    //....
    this.wires=[];
  }

  addWire(){
    let wire=[];
    for (var i in arguments){
      wire.push(arguments[i]);
    }
    this.wires.push(wire);
  }
}

class HttpNodeIn extends HttpNode{
  constructor(id,url){
    super(id,"http in");
    this.url=url;
    this.method="get";
        //....
  }
}

class HttpNodeResponse extends HttpNode{
  constructor(id){
    super(id,"http response");
    //....
  }
}

class HttpNodeRequest extends HttpNode{
  constructor(id){
    super(id,"http request");
    this.method="GET";
    //....
  }
}

export {HttpNodeIn,HttpNodeResponse,HttpNodeRequest};
```

B.6 Κώδικας flow.js

```
import {HttpNodeIn,HttpNodeResponse,HttpNodeRequest} from '../nodes/httpNodes';
import FunctionNode from '../nodes/functionNodes';
import uniqid from 'uniqid';
import
{generateDataSources,unique,arrayToObject,addResponses,generateSchema,addParamsAndP
ath,createEntryUrl,funcSetVar,funcPrepReq,funcStoreData,funcPrepRes} from
'../nodes/flow.utils'

class Flow{
  //initialize NODERED flow
  constructor(label){
    this.label=label,
    this.nodes=[];
  }

  //adds new Node to flow
  addNode(node){
    if (this.nodes.length===0){
      this.nodes.push(node);
    }else{
      this.nodes[this.nodes.length-1].addWire(node.id);
      this.nodes.push(node);
    }
  }

  //creates a NODERED flow based on a provided custom or original method and the
original API
  //also returns path property for new openapi spec
  createFromMethod(method,exposedAPI){
    let path={};
    let url="";
    let parameters=[];
    let schema={};
    let sources=[];

    //check if the method is a complete method from the original api
    if(Object.prototype.hasOwnProperty.call(method, 'path')){
      //if method is a complete method from original api create a wrapper endpoint
around that method
      url=createEntryUrl(method);
      this.addNode(new HttpNodeIn(uniqid(),url));
      this.addNode(new FunctionNode(uniqid(),funcSetVar()));
      this.addNode(new
FunctionNode(uniqid(),funcPrepReq(exposedAPI.servers[0].url,method)));
```

```
      this.addNode(new HttpNodeRequest(uniqid()));
      this.addNode(new HttpNodeResponse(uniqid()));
      schema=method.responseSchema;
      sources=generateDataSources([method.operationId],exposedAPI);
    }
    else{
      //else if custom method find the parameters and methods that are involved
      let methodsUsed=[];
      for (let i in method.responseSchema){
        methodsUsed.push(method.responseSchema[i].partof);
        parameters=parameters.concat(method.responseSchema[i].parameters);
      }
      methodsUsed=[...new Set(methodsUsed)];
      sources=generateDataSources(methodsUsed,exposedAPI);
      parameters=unique(parameters);
      url=createEntryUrl(method,parameters);
      let methodsObject=arrayToObject(methodsUsed);
      methodsObject=addResponses(methodsObject,method.responseSchema);
      schema=generateSchema(methodsObject);
      methodsObject=addParamsAndPath(methodsObject,exposedAPI);
      //add entry point nodes
      this.addNode(new HttpNodeIn(uniqid(),url));
      this.addNode(new FunctionNode(uniqid(),funcSetVar()));
      //add external request nodes
      for (let i in methodsObject){
        this.addNode(new
FunctionNode(uniqid(),funcPrepReq(exposedAPI.servers[0].url,methodsObject[i])));
        this.addNode(new HttpNodeRequest(uniqid()));
        this.addNode(new FunctionNode(uniqid(),funcStoreData(methodsObject[i])));
      }
      //add exit point noderedResponse]
      this.addNode(new FunctionNode(uniqid(),funcPrepRes()));
      this.addNode(new HttpNodeResponse(uniqid()));
    }
    path[url]={
      get:{
        summary:method.summary,
        description:method.description,
        operationId:method.operationId,
        parameters:method.parameters || parameters,
        responses:{
          200: {
            content:{
              'application/json':{
                schema:schema
              }
            }
```

```
            }
          },
          'x-data-sources':sources
        }
      }
      return(path);
    }
}

export default Flow;
```

Παράρτημα Γ VDC Blueprint Schema

```
{
   "type":"object",
   "description":"This is a VDC Blueprint which consists of five sections",
   "properties":{
      "INTERNAL_STRUCTURE":{
         "type":"object",
         "description":"General information about the VDC Blueprint",
         "properties":{
            "Overview":{
               "type":"object",
               "properties":{
                  "name":{
                     "type":"string",
                     "description":"This field should contain the name of the VDC
Blueprint"
                  },
                  "description":{
                     "type":"string",
                     "description":"This field should contain a short description
of the VDC Blueprint"
                  },
                  "tags":{
                     "type":"array",
                     "description":"Each element of this array should contain some
keywords that describe the functionality of each one exposed VDC method",
                     "items":{
                        "type":"object",
                        "properties":{
                           "method_id":{
                              "type":"string",
                              "description":"The id (operationId) of the method (as
indicated in the EXPOSED_API.paths field)"
                           },
                           "tags":{
                              "type":"array",
                              "items":{
                                 "type":"string"
                              },
                              "minItems":1,
                              "uniqueItems":true
                           }
                        },
                        "additionalProperties":false,
                        "required":[
                           "method_id",
```

```
                "tags"
            ]
         },
         "minItems":1,
         "uniqueItems":true
      }
   },
   "additionalProperties":false,
   "required":[
      "name",
      "description",
      "tags"
   ]
},
"Data_Sources":{
   "type":"array",
   "items":{
      "type":"object",
      "properties":{
         "id":{
            "type":"string",
            "description":"A unique identifier"
         },
         "description":{
            "type":"string"
         },
         "location":{
            "enum":[
               "cloud",
               "edge"
            ]
         },
         "class":{
            "enum":[
               "relational database",
               "object storage",
               "time-series database",
               "api",
               "data stream"
            ]
         },
         "type":{
            "enum":[
               "MySQL",
               "Minio",
               "InfluxDB",
               "rest",
```

```
                        "other"
                     ]
                  },
                  "parameters":{
                     "type":"object",
                     "description":"Connection parameters"
                  },
                  "schema":{
                     "type":"object"
                  }
               },
               "required":[
                  "id"
               ]
            },
            "minItems":1,
            "uniqueItems":true
         },
           "Methods_Input": {
            "type": "object",
            "description": "This filed contains the part of the data source
that each method needs to be executed",
            "properties": {
              "Methods": {
                "type": "array",
                "description": "The list of methods",
                "items": {
                  "type": "object",
                  "properties": {
                    "method_id": {
                      "type": "string",
                      "description": "The id (operationId) of the method (as
indicated in the EXPOSED_API.paths field)"
                    },
                    "dataSources": {
                      "type": "array",
                      "description": "The list of data sources required by the
method",
                      "items": {
                        "type": "object",
                        "properties": {
                          "dataSource_id": {
                            "type": "string",
                            "description": "The id of the data sources (as
indicated in the Data_Sources field)"
                          },
                          "dataSource_type": {
```

```
                                "type": "string",
                                "description": "The type of the data sources
(relationa/not_relational/object)"
                              },
                              "database": {
                                "type": "array",
                                "description": "the list of databases required by a
method in a data source",
                                "items": {
                                  "type": "object",
                                  "properties": {
                                    "database_id": {
                                      "type": "string",
                                      "description": "The id of the database"
                                    },
                                    "tables": {
                                      "type": "array",
                                      "description": "the list of
tables/collections required by a method in a data source",
                                      "items": {
                                        "type": "object",
                                        "properties": {
                                          "table_id": {
                                            "type": "string",
                                            "description": "The id of the
tables/collection "
                                          },
                                          "columns": {
                                            "type": "array",
                                            "items": {
                                              "type": "object",
                                              "properties": {
                                                "column_id": {
                                                  "type": "string",
                                                  "description": "The id of the
column/field"
                                                },
                                                "computeDataUtility": {
                                                  "type": "boolean",
                                                  "description": "True if it is
required for data utility computation"
                                                }
                                              }
                                            }
                                          }
                                        }
                                      }
```

```
                                    }
                                 }
                              }
                           }
                        }
                     }
                  }
               }
            }
         }
      },
      "Flow":{
         "type":"object",
         "description":"The data flow that implements the VDC",
         "properties":{
            "platform":{
               "enum":[
                  "Spark",
                  "Node-RED"
               ]
            },
            "parameters":{
               "type":"object"
            },
            "source_code":{

            }
         }
      },
      "DAL_Images":{
         "description":"Docker images that must be deployed in the DAL
indexed by DAL name. It will be used to compose the service name and the DNS entry
that other images in the cluster can access to.",
         "type":"object",
         "additionalProperties":{
            "description":"ImageSet represents a set of docker images whose
key is an identifier and value is a docker image name in the standard format
[group]/\u003cimage_name\u003e:[release]",
            "type":"object",
            "additionalProperties":{
               "description":"ImageInfo is the information about an image
that will be deployed by the deployment engine",
               "type":"object",
               "required":[
                  "image"
               ],
```

```
                    "properties":{
                       "external_port":{
                          "description":"Port in which this image must be exposed.
It must be unique across all images in all the ImageSets defined in this blueprint.
Due to limitations in k8s, the port range must be bewteen 30000 and 32767",
                          "type":"integer",
                          "format":"int64"
                       },
                       "image":{
                          "description":"Image is the image name in the standard
format [group]/\u003cimage_name\u003e:[release]",
                          "type":"string"
                       },
                       "internal_port":{
                          "description":"Port in which the docker image is
listening internally. Two images inside the same ImageSet can't have the same
internal port.",
                          "type":"integer",
                          "format":"int64"
                       }
                    }
                 }
              }
           },
           "VDC_Images":{
              "description":"ImageSet represents a set of docker images whose key
is an identifier and value is a docker image name in the standard format
[group]/\u003cimage_name\u003e:[release]",
              "type":"object",
              "additionalProperties":{
                 "description":"ImageInfo is the information about an image that
will be deployed by the deployment engine",
                 "type":"object",
                 "required":[
                    "image"
                 ],
                 "properties":{
                    "external_port":{
                       "description":"Port in which this image must be exposed. It
must be unique across all images in all the ImageSets defined in this blueprint.
Due to limitations in k8s, the port range must be bewteen 30000 and 32767",
                       "type":"integer",
                       "format":"int64"
                    },
                    "image":{
                       "description":"Image is the image name in the standard
format [group]/\u003cimage_name\u003e:[release]",
```

```
                        "type":"string"
                     },
                     "internal_port":{
                        "description":"Port in which the docker image is listening
internally. Two images inside the same ImageSet can't have the same internal
port.",
                        "type":"integer",
                        "format":"int64"
                     }
                  }
               }
            },
            "Identity_Access_Management":{
               "type":"object",
               "properties":{
                  "jwks_uri":{
                     "type":"string"
                  },
                  "iam_endpoint":{
                     "type":"string"
                  },
                  "roles":{
                     "type":"array",
                     "items":{
                        "type":"string"
                     },
                     "minItems":1
                  },
                  "provider":{
                     "type":"array",
                     "items":{
                        "type":"object",
                        "properties":{
                           "name":{
                              "type":"string"
                           },
                           "type":{
                              "type":"string"
                           },
                           "uri":{
                              "type":"string"
                           },
                           "loginPortal":{
                              "type":"string"
                           }
                        },
                        "required":[
```

```
                        "name",
                        "uri"
                     ]
                  },
                  "minItems":1
               }
            },
            "required":[
               "jwks_uri",
               "iam_endpoint"
            ]
         },
         "Testing_Output_Data":{
            "type":"array",
            "items":{
               "type":"object",
               "properties":{
                  "method_id":{
                     "type":"string",
                     "description":"The id (operationId) of the method (as
indicated in the EXPOSED_API.paths field)"
                  },
                  "zip_data":{
                     "type":"string",
                     "description":"The URI to the zip testing output data for
each one exposed VDC method"
                  }
               },
               "additionalProperties":false,
               "required":[
                  "method_id",
                  "zip_data"
               ]
            },
            "minItems":1,
            "uniqueItems":true
         }
      },
      "additionalProperties":false,
      "required":[
         "Overview",
         "Data_Sources"
      ]
   },
   "DATA_MANAGEMENT":{
      "description":"list of methods",
      "type":"array",
```

```
         "items":{
            "type":"object",
            "properties":{
               "method_id":{
                  "description":"The id (operationId) of the method (as indicated
in the EXPOSED_API.paths field)",
                  "type":"string"
               },
               "attributes":{
                  "type":"object",
                  "description":"goal trees",
                  "properties":{
                     "dataUtility":{
                        "type":"array",
                        "items":{
                           "type":"object",
                           "description":"definition of the metric",
                           "properties":{
                              "id":{
                                 "description":"id of the metric",
                                 "type":"string"
                              },
                              "name":{
                                 "description":"name of the metric",
                                 "type":"string"
                              },
                              "type":{
                                 "description":"type of the metric",
                                 "type":"string"
                              },
                              "properties":{
                                 "type":"object",
                                 "description":"properties related to the metric",
                                 "additionalProperties":{
                                    "type":"object",
                                    "description":"properties related to the
metric",
                                    "properties":{
                                       "unit":{
                                          "description":"unit of measure of the
property",
                                          "type":"string"
                                       },
                                       "maximum":{
                                          "description":"lower limit of the offered
property",
                                          "type":"number"
```

```
                                    },
                                    "minimum":{
                                       "description":"upper limit of the offered
property",
                                       "type":"number"
                                    },
                                    "value":{
                                       "description":"value of the property",
                                       "type":[
                                          "string",
                                          "number",
                                          "array",
                                          "boolean"
                                       ]
                                    }
                                 }
                              }
                           }
                        }
                     },
                     "security":{

"$ref":"#/properties/DATA_MANAGEMENT/items/properties/attributes/properties/dataUti
lity"
                     },
                     "privacy":{

"$ref":"#/properties/DATA_MANAGEMENT/items/properties/attributes/properties/dataUti
lity"
                     }
                  }
               }
            },
            "required":[
               "method_id",
               "attributes"
            ]
         }
      },
      "ABSTRACT_PROPERTIES":{

      },
      "COOKBOOK_APPENDIX":{
         "description":"Definition of the Cookbook Appendix section in the
blueprint",
         "type":"object",
```

```
         "required":[
            "name",
            "infrastructure"
         ],
         "properties":{
            "description":{
               "description":"An optional description for the deployment",
               "type":"string"
            },
            "infrastructure":{
               "description":"A list of infrastructures that should be initialized
to deploy VDCs of this blueprint",
               "type":"array",
               "items":{
                  "description":"Represents a cloud or edge site that's able to
create resources such as virtual machines or volumes",
                  "type":"object",
                  "required":[
                     "name",
                     "type",
                     "provider",
                     "resources"
                  ],
                  "properties":{
                     "description":{
                        "description":"Optional description for the
infrastructure",
                        "type":"string"
                     },
                     "name":{
                        "description":"Unique name for the infrastructure",
                        "type":"string",
                        "uniqueItems":true
                     },
                     "owner":{
                        "description":"Owner of the infrastructure",
                        "type":"string",
                        "pattern":"DataOwner|DataConsumer|"
                     },
                     "provider":{
                        "description":"Contains information about a cloud or edge
provider",
                        "type":"object",
                        "properties":{
                           "api_endpoint":{
                              "description":"Endpoint to use for this
infrastructure",
```

```
                        "type":"string"
                     },
                     "api_type":{
                        "description":"Type of the infrastructure. i.e AWS,
Cloudsigma, GCP or Edge",
                        "type":"string",
                        "example":"AWS"
                     },
                     "authentication":{
                        "description":"Authentication information to use on
the provider.",
                        "type":"object"
                     }
                  }
               },
               "resources":{
                  "description":"List of resources to deploy",
                  "type":"array",
                  "items":{
                     "description":"Represents a resource such as a virtual
machine",
                     "type":"object",
                     "required":[
                        "role",
                        "image_id"
                     ],
                     "properties":{
                        "cores":{
                           "description":"Number of cores. Ignored if type is
provided",
                           "type":"integer",
                           "format":"int64"
                        },
                        "cpu":{
                           "description":"CPU speed in MHz. Ignored if type
is provided",
                           "type":"integer",
                           "format":"int64"
                        },
                        "disk":{
                           "description":"Boot disk size in MB",
                           "type":"integer",
                           "format":"int64"
                        },
                        "drives":{
                           "description":"List of data drives to attach to
this VM",
```

```
                                    "type":"array",
                                    "items":{
                                       "description":"Contains the information of a
data drive",
                                       "type":"object",
                                       "required":[
                                          "name",
                                          "type",
                                          "size"
                                       ],
                                       "properties":{
                                          "name":{
                                             "description":"Name of the image to use.
Most of the times, it will be available as /dev/disk/by-id/${name} value in the
VM",
                                             "type":"string"
                                          },
                                          "size":{
                                             "description":"Size of the disk in MB",
                                             "type":"integer",
                                             "format":"int64"
                                          },
                                          "type":{
                                             "description":"Type of the drive. It can
be \"SSD\" or \"HDD\"",
                                             "type":"string",
                                             "pattern":"SSD|HDD",
                                             "example":"SSD"
                                          }
                                       }
                                    }
                                 },
                                 "image_id":{
                                    "description":"Boot image ID to use",
                                    "type":"string"
                                 },
                                 "ip":{
                                    "description":"IP to assign this VM. In case it's
not specified, the first available one will be used.",
                                    "type":"string"
                                 },
                                 "name":{
                                    "description":"Suffix for the hostname. The real
hostname will be formed of the infrastructure name + resource name",
                                    "type":"string"
                                 },
                                 "ram":{
```

```
                                    "description":"RAM quantity in MB. Ignored if type
is provided",
                                    "type":"integer",
                                    "format":"int64"
                                 },
                                 "role":{
                                    "description":"Role that this VM plays. In case of
a Kubernetes deployment at least one \"master\" is needed.",
                                    "type":"string",
                                    "pattern":"master|slave",
                                    "example":"master"
                                 },
                                 "type":{
                                    "description":"Type of the VM to create",
                                    "type":"string",
                                    "example":"n1-small"
                                 }
                              }
                           }
                        },
                        "tags":{
                           "description":"List of tags to apply to this
infrastructure",
                           "type":"array",
                           "items":{
                              "type":"string"
                           }
                        },
                        "type":{
                           "description":"Type of the infrastructure",
                           "type":"string",
                           "pattern":"Cloud|Edge",
                           "example":"Cloud"
                        }
                     }
                  }
               },
               "name":{
                  "description":"Unique name of the deployment",
                  "type":"string",
                  "uniqueItems":true
               },
               "Identity_Access_Management":{
                  "type":"object",
                  "properties":{
                     "validation_keys":{
                        "type":"array",
```

```
    "items":{
       "type":"object"
    }
},
"mapping":{
    "type":"array",
    "items":{
       "oneOf":[
          {
             "type":"object",
             "properties":{
                "provider":{
                   "type":"string"
                },
                "roles":{
                   "type":"array",
                   "items":{
                      "type":"string"
                   }
                },
                "role_map":{
                   "type":"array",
                   "items":{
                      "type":"object",
                      "properties":{
                         "matcher":{
                            "type":"string"
                         },
                         "roles":{
                            "type":"array",
                            "items":{
                               "type":"string"
                            }
                         },
                         "priority":{
                            "type":"number"
                         }
                      }
                   }
                },
                "mapping_url":{
                   "enum":[
                      ""
                   ]
                }
             },
             "required":[
```

```
                                "role_map"
                            ]
                        },
                        {
                            "type":"object",
                            "properties":{
                                "provider":{
                                    "type":"string"
                                },
                                "roles":{
                                    "type":"array",
                                    "items":{
                                        "type":"string"
                                    }
                                },
                                "mapping_url":{
                                    "type":"string"
                                },
                                "role_map":{
                                    "enum":[
                                        ""
                                    ]
                                }
                            },
                            "required":[
                                "mapping_url"
                            ]
                        }
                    ]
                }
            }
        },
        "required":[
            "mapping"
        ]
    }
}
},
"EXPOSED_API":{
    "title":"CAF API",
    "type":"object",
    "description":"The CAF RESTful API of the VDC, written according to the current version (3.0.1) of the OpenAPI Specification (OAS), but also adapted to DITAS requirements",
    "properties":{
        "paths":{
            "type":"object",
```

```
                "patternProperties":{
                   "^/":{
                      "type":"object",
                      "patternProperties":{
                         "^get$":{
                            "allOf":[
                               {

"$ref":"#/properties/EXPOSED_API/definitions/method"
                               },
                               {
                                  "properties":{
                                     "parameters":{

                                     }
                                  }
                               }
                            ]
                         },
                         "^post$":{
                            "allOf":[
                               {

"$ref":"#/properties/EXPOSED_API/definitions/method"
                               },
                               {
                                  "properties":{
                                     "requestBody":{
                                        "type":"object",
                                        "properties":{
                                           "content":{

"$ref":"#/properties/EXPOSED_API/definitions/content"
                                           }
                                        }
                                     },
                                     "required":[
                                        "requestBody"
                                     ]
                                  }
                               }
                            ]
                         }
                      }
                   }
                }
             }
```

```
         },
         "definitions":{
            "method":{
               "title":"An Exposed VDC Method",
               "type":"object",
               "description":"Corresponds to the Operation Object defined in the
OpenAPI Specification (OAS) version 3.0.1",
               "properties":{
                  "summary":{

                  },
                  "operationId":{

                  },
                  "responses":{
                     "type":"object",
                     "patternProperties":{
                        "^200$|^201$":{
                           "type":"object",
                           "properties":{
                              "content":{

"$ref":"#/properties/EXPOSED_API/definitions/content"
                              }
                           },
                           "required":[
                              "content"
                           ]
                        }
                     }
                  },
                  "x-data-sources":{
                     "type":"array",
                     "description":"An array that contains all the identifiers of
the data sources (as indicated in the INTERNAL_STRUCTURE.Data_Sources field) that
are accessed by the method",
                     "items":{
                        "type":"string"
                     },
                     "minItems":1,
                     "uniqueItems":true
                  },
                  "x-iam-roles":{
                     "type":"array",
                     "items":{
                        "type":"string"
                     }
```

```
                }
            },
            "required":[
                "summary",
                "operationId",
                "responses",
                "x-data-sources"
            ]
        },
        "content":{
            "type":"object",
            "patternProperties":{
                "^application/json$":{
                    "type":"object",
                    "properties":{
                        "schema":{
                            "type":"object"
                        }
                    },
                    "required":[
                        "schema"
                    ]
                }
            }
        }
    }
},
"additionalProperties":false,
"required":[
    "INTERNAL_STRUCTURE",
    "DATA_MANAGEMENT",
    "ABSTRACT_PROPERTIES",
    "COOKBOOK_APPENDIX",
    "EXPOSED_API"
]
}
```