Εθνικο Μετσοβιο Πολυτεχνειο
Σχολη Ηλεκτρολογων Μηχανικων και Μηχανικων Υπολογιστων
Τομεας Τεχνολογιας Πληροφορικης και Υπολογιστων

Αυτόματη Παραγωγή Σεναρίων Ελέγχου για Προγραμματιστικές Διεπαφές τύπου REST (RESTful APIs)

Διπλωματικη Εργασια

του

ΠΑΥΛΟΥ Α. ΣΤΑΘΟΠΟΥΛΟΥ

Επιβλέποντες: Νικόλαος Σ. Παπασπύρου
Καθηγητής ΣΗΜΜΥ, ΕΜΠ

Κώστας Σαΐδης
Ακαδημαϊκός Υπότροφος
Τμ. Πληροφορικής & Τηλεπικοινωνιών, ΕΚΠΑ

Αθήνα, Φεβρουάριος 2021

Εθνικο Μετσοβιο Πολυτεχνειο
Σχολη Ηλεκτρολογων Μηχανικων και Μηχανικων Υπολογιστων
Τομεας Τεχνολογιας Πληροφορικης και Υπολογιστων

Αυτόματη Παραγωγή Σεναρίων Ελέγχου για Προγραμματιστικές Διεπαφές τύπου REST (RESTful APIs)

Διπλωματικη Εργασια

του

ΠΑΥΛΟΥ Α. ΣΤΑΘΟΠΟΥΛΟΥ

Επιβλέποντες: Νικόλαος Σ. Παπασπύρου
Καθηγητής ΣΗΜΜΥ, ΕΜΠ

Κώστας Σαΐδης
Ακαδημαϊκός Υπότροφος
Τμ. Πληροφορικής & Τηλεπικοινωνιών, ΕΚΠΑ

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 8η Φεβρουαρίου 2021.

(Υπογραφή)	*(Υπογραφή)*	*(Υπογραφή)*
.....	.....	.....
Νικόλαος Σ. Παπασπύρου	Γεώργιος Γκούμας	Βασίλειος Βεσκούκης
Καθηγητής	Αναπληρωτής Καθηγητής	Αναπληρωτής Καθηγητής

Αθήνα, Φεβρουάριος 2021

Εθνικο Μετσοβιο Πολυτεχνειο
Σχολη Ηλεκτρολογων Μηχανικων και Μηχανικων Υπολογιστων
Τομεας Τεχνολογιας Πληροφορικης και Υπολογιστων

Η εργασία αυτή είναι επίσης διαθέσιμη ως Τεχνική Αναφορά CSD-SW-TR-1-21, Εθνικό Μετσόβιο Πολυτεχνείο, Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών, Τομέας Τεχνολογίας Πληροφορικής και Υπολογιστών, Εργαστήριο Τεχνολογίας Λογισμικού, Φεβρουάριος 2021.
URL: http://www.softlab.ntua.gr/techrep/
FTP: ftp://ftp.softlab.ntua.gr/pub/techrep/

ΔΗΛΩΣΗ ΜΗ ΛΟΓΟΚΛΟΠΗΣ ΚΑΙ ΑΝΑΛΗΨΗΣ ΠΡΟΣΩΠΙΚΗΣ ΕΥΘΥΝΗΣ

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

(Υπογραφή)

.....
Παύλος Σταθόπουλος

22 Ιανουαρίου 2021

Περίληψη

Η παρούσα εργασία έχει σκοπό τον σχεδιασμό και την υλοποίηση ενός μηχανισμού αυτόματης παραγωγής σεναρίων ελέγχου για προγραμματιστικές διεπαφές τύπου REST (RESTful APIs), με βέλτιστο για τον προγραμματιστή τρόπο.

Οι προγραμματιστικές διεπαφές τύπου REST είναι ο κατ' εξοχήν μηχανισμός επικοινωνίας εφαρμογών στον Παγκόσμιο Ιστό. Το μεγαλύτερο μέρος των διαδικτυακών υπηρεσιών σήμερα παρέχονται με τη βοήθεια προγραμματιστικών διεπαφών τύπου REST, οι οποίες αναλαμβάνουν την μεταφορά πληροφοριών μεταξύ συστημάτων. Οι τεχνολογικές εξελίξεις σε τομείς όπως η μηχανική μάθηση και το Διαδίκτυο των Πραγμάτων (IoT) εδραιώνουν περαιτέρω τον ρόλο των προγραμματιστικών διεπαφών στο σύγχρονο ψηφιακό οικοσύστημα.

Παράλληλα απαραίτητος θεωρείται ο συστηματικός έλεγχος των προγραμματιστικών διεπαφών, καθώς η πολυπλοκότητα του σχεδιασμού τους εγκυμονεί κινδύνους σφαλμάτων. Ο αποτελεσματικός έλεγχός τους είναι ένας τομέας που ήδη απασχολεί την προγραμματιστική κοινότητα τόσο σε ερευνητικό όσο και σε επιχειρησιακό επίπεδο, ενώ όσο αυξάνεται η δημοτικότητά τους, τόσο πιο απαραίτητος θα γίνεται.

Στο πλαίσιο της εργασίας σχεδιάστηκε αρχικά μία δηλωτική γλώσσα ορισμού μοντέλων RESTful API. Με αυτήν μπορεί κανείς να περιγράψει μία προγραμματιστική διεπαφή τύπου REST ως προς τα χαρακτηριστικά της, όπως είναι τα τελικά σημεία (endpoints) και οι μέθοδοι (methods) που υποστηρίζονται. Από το μοντέλο της διεπαφής που παράγεται, δημιουργείται πλήρως αυτόματα ένα σύνολο από σενάρια ελέγχου για το RESTful API. Αυτά ελέγχουν την ορθότητά του μέσα από την αλληλεπίδρασή τους με αυτό. Ακόμα, για την βέλτιστη προγραμματιστική εμπειρία, η διαδικασία παραγωγής των σεναρίων ελέγχου υλοποιήθηκε έτσι ώστε να υποστηρίζεται η ενσωμάτωσή της σε εργαλείο κατασκευής λογισμικού.

Για την επαλήθευση της ορθής λειτουργίας των μηχανισμών που αναπτύχθηκαν στο πλαίσιο της παρούσας εργασίας παρατίθενται οι εφαρμογές τους σε δύο RESTful API του διαδικτύου, ένα παρατηρητήριο τιμών και ένα ψηφιακό μητρώο δικτυακών υποδομών. Μέσα από αυτές αποδεικνύεται η προσφορά της βέλτιστης εμπειρίας προγραμματισμού κατά τον δηλωτικό ορισμό ενός RESTful API και τελικά της πλήρως αυτόματης παραγωγής σεναρίων ελέγχου που επιβεβαιώνουν την ορθή λειτουργία του.

Λέξεις Κλειδιά

API, REST, Έλεγχος λογισμικού, Δηλωτική γλώσσα, Μοντέλο RESTful API, Κατασκευή λογισμικού, Γεννήτρια κώδικα

Abstract

The purpose of this thesis is the design and implementation of a mechanism for automated RESTful API test case generation, in the best possible programming experience.

RESTful APIs are the de facto communication mechanism for applications used in World Wide Web. Most of web services today are provided using RESTful APIs, which help transfer information between computer systems. Technological advances in fields such as machine learning and the Internet of Things (IoT) strengthen the role of RESTful APIs in today's digital ecosystem.

At the same time, the systematic testing of RESTful APIs is considered necessary as their design complexity carries risk of errors. Software testing is currently a field of much interest for the programming community both in research and in business level and with the rise in their popularity, it becomes even more needed.

For the purpose of this thesis, a declarative language for RESTful API models was initially designed. Using it, a RESTful API can be described by its features, like the supported endpoints and methods. Using the created model, a set of test cases for the RESTful API is automatically generated. These test the RESTful API's functionality by interacting with it. Also, for the best programming experience, the mechanism for the automated RESTful API test case generation was designed in a way that it is offered for integration in build automation tools.

In order to evaluate the tools developed within the thesis, their application on two RESTful Web APIs is listed, a price observatory and a digital networking infrastructure register. Through the above, the best programming experience is proved to be offered for the declarative specification of a RESTful API and finally the fully automated generation of test cases which validate its proper functioning.

Keywords

API, REST, Software testing, declarative domain-specific language, RESTful API Model, Software Builder, Code generator

Περιεχόμενα

Κατάλογος Σχημάτων

Κεφάλαιο 1

Εισαγωγή

Από την εποχή που γεννήθηκε η ιδέα του Παγκόσμιου Ιστού ως ένα διεθνές σύστημα διακίνησης πληροφοριών μέχρι σήμερα, έχουμε διανύσει τρεις δεκαετίες απροσδόκητης τεχνολογικής έκρηξης [1]. Η αξιοποίηση του διαδικτύου πλέον δεν αφορά μόνο ακαδημαϊκούς υπολογιστές για ερευνητικούς σκοπούς, αλλά αποτελεί αναπόσπαστο κομμάτι της καθημερινότητας της πλειονότητας των ανθρώπων παγκοσμίως. Με την επικράτηση των προσωπικών υπολογιστών και στη συνέχεια των έξυπνων κινητών (smartphones), η ψηφιακή διακίνηση πληροφορίας διευρύνεται και ενισχύεται ασταμάτητα. Όσο για το άμεσο μέλλον, βέβαιες πρέπει να θεωρούνται η σταδιακή μεταβίβαση στο Διαδίκτυο των Πραγμάτων (IoT) και η καθολικότητα των έξυπνων συσκευών.

Για την αξιοποίηση του τεράστιου όγκου της πληροφορίας που διακινείται μέσω του Παγκόσμιου Ιστού χρησιμοποιούνται διεπαφές προγραμματισμού. Σκοπός αυτού του μηχανισμού επικοινωνίας είναι η αποδοχή αιτημάτων από άλλα προγράμματα του διαδικτύου και η επεξεργασία τους, με στόχο τη μεταφορά κατάλληλων μηνυμάτων σε υπολογιστές και την επιστροφή των αντιστοίχων αποτελεσμάτων στους χρήστες [2].

Η αδιάλειπτη ανάπτυξη του Παγκόσμιου Ιστού καθιστά περισσότερο αναγκαίο από ποτέ τον αποτελεσματικό έλεγχο των διεπαφών. Οι προγραμματιστές οφείλουν να είναι βέβαιοι πως οι εφαρμογές που αναπτύσσουν παράγουν πάντα τα επιθυμητά αποτελέσματα, με την κατάλληλη επεξεργασία κάθε φορά της μεταδιδόμενης πληροφορίας. Μόνο έτσι μπορεί να εδραιωθεί η αξιοπιστία των διαδικτυακών εφαρμογών, τόσο για τους προγραμματιστές που καλούνται να αναπτύξουν νέα προγράμματα που θα αλληλεπιδρούν με τα υπάρχοντα, όσο και για τους απλούς χρήστες που απαιτούν ασφάλεια και σιγουριά κατά την ψηφιακή πλοήγησή τους.

1.1 Αντικείμενο διπλωματικής εργασίας

Αντικείμενο της διπλωματικής εργασίας είναι η ανάπτυξη ενός εργαλείου για τον αυτόματο έλεγχο διεπαφών προγραμματισμού τύπου REST (RESTful APIs). Με το εργαλείο αυτό οι προγραμματιστές θα μπορούν να ελέγχουν πιο αποτελεσματικά τον κώδικά τους, καθώς θα τους παρέχει έτοιμα σενάρια ελέγχου. Τα σενάρια αυτά θα είναι παραμετροποιήσιμα, θα καλύπτουν κατά το δυνατόν πληρέστερα τις δυνατότητες μιας διεπαφής και η δημιουργία τους θα γίνεται αυτόματα. Το μόνο που απαιτείται από τον χρήστη είναι η περιγραφή της διεπαφής σε κατάλληλη μορφή.

1.2 Συνεισφορά διπλωματικής εργασίας

Σκοπός της διπλωματικής εργασίας είναι η προσφορά στην κοινότητα προγραμματιστών Java ενός εργαλείου για τον αποδοτικό έλεγχο προγραμματιστικών διεπαφών τύπου REST.

Το εργαλείο αυτό σχεδιάστηκε με γνώμονα την βέλτιστη δυνατή εμπειρία προγραμματισμού. Χρησιμοποιώντας το, οι προγραμματιστές είναι σε θέση να παρέχουν εγγυήσεις ποιότητας για τα προϊόντα τους, μιας και η αυτόματη παραγωγή των σεναρίων ελέγχου συνεπάγεται την έλλειψη ανθρωπίνων λαθών, τόσο συντακτικών όσο και λογικών.

Ταυτόχρονα επιτυγχάνεται αύξηση της παραγωγικότητας, αφού αρκεί η κατάλληλη περιγραφή των προδιαγραφών της προγραμματιστικής διεπαφής τύπου REST για να παραχθούν αμέσως όλα τα δυνατά σενάρια ελέγχου που καλύπτονται από το πεδίο εφαρμογής της εργασίας.

1.3 Οργάνωση του τόμου

Η εργασία αυτή είναι οργανωμένη σε επτά κεφάλαια, συμπεριλαμβανομένου του παρόντος και πρώτου που περιλαμβάνει το αντικείμενο της εργασίας, τη συνεισφορά της στην προγραμματιστική κοινότητα και την οργάνωσή της. Στο Κεφάλαιο 2 δίνεται μια εισαγωγή στον αυτόματο έλεγχο προγραμματιστικών διεπαφών διαδικτύου, όπου αναπτύσσονται οι έννοιες των Web APIs και του ελέγχου λογισμικού. Στο Κεφάλαιο 3 περιγράφεται ο σχεδιασμός της εργασίας, ενώ στο Κεφάλαιο 4 παρουσιάζεται αναλυτικά η υλοποίησή της. Στη συνέχεια, στο Κεφάλαιο 5 παρατίθενται εφαρμογές σε δύο διαδικτυακά RESTful APIs που επικυρώνουν την ορθή λειτουργία της εργασίας. Στο Κεφάλαιο 6 γίνεται αναφορά σε συναφή προγραμματιστικά εργαλεία και ακαδημαϊκές μελέτες. Τέλος, στο Κεφάλαιο 7 δίνονται τα συμπεράσματα της εργασίας και σχολιάζονται ορισμένες μελλοντικές επεκτάσεις της.

Κεφάλαιο 2

Αυτόματος Έλεγχος Προγραμματιστικών Διεπαφών Διαδικτύου

Στο κεφάλαιο αυτό παρουσιάζεται το θεωρητικό πλαίσιο και αναλυτικά οι τεχνολογίες που σχετίζονται με την παρούσια εργασία. Πιο συγκεκριμένα, περιγράφεται η έννοια του ελέγχου λογισμικού καθώς και της αρχιτεκτονικής REST.

2.1 Έλεγχος Λογισμικού

Σε αντίθεση με τα απτά υλικά που μπορούμε να εποπτεύσουμε και να ελέγξουμε εύκολα και γρήγορα τη λειτουργία τους, καθώς και τις αδυναμίες τους, με τα προγράμματα η διαδικασία αυτή γίνεται αρκετά σύνθετη.

Υπάρχουν οι εξής τρόποι ελέγχου ενός λογισμικού:

- Επιθεώρηση του κώδικα (review), μέσα από την περιήγηση (walkthrough) ή την επισκόπησή του (inspection) [3]
- Χρήση εργαλείων στατικής ανάλυσης [4]
- Δυναμικός έλεγχος

Οι πρώτοι δύο τρόποι ανήκουν στην κατηγορία του στατικού ελέγχου. Η παρούσα εργασία επικεντρώνεται στον δυναμικό έλεγχο που γίνεται με την εκτέλεση του προγράμματος με σκοπό την εύρεση σφαλμάτων [5][6].

Ο έλεγχος λογισμικού πλέον θεωρείται αναπόσπαστο τμήμα της βιομηχανίας, με έρευνες να δείχνουν πως η πλειονότητα των εταιρειών σήμερα χρησιμοποιούν είτε επίσημα ορισμένες διαδικασίες ελέγχου, είτε γενικές αρχές ορθής λειτουργίας [7]. Όμως άλλη έρευνα δείχνει πως ακόμα και εταιρείες που εξειδικεύονται σε ανάπτυξη λογισμικού δεν έχουν ιδανικές μεθόδους ελέγχου [8]. Επομένως είναι ένας τομέας που εξακολουθεί να παραμένει επίκαιρος.

Οποιαδήποτε εφαρμογή σχεδιάζεται και υλοποιείται, από ένα απλό μαθηματικό εργαλείο μέχρι μία περίπλοκη προγραμματιστική διεπαφή, είναι πιθανό να περιλαμβάνει λάθη. Όσο αυξάνεται η πολυπλοκότητα της εφαρμογής, τόσο πιο απαραίτητος είναι ο ενδελεχής και αποτελεσματικός έλεγχός της.

2.1.1 Είδη δυναμικού ελέγχου

Ως προς τον σκοπό του δυναμικού ελέγχου, μπορούμε να ταξινομήσουμε την πληθώρα μεθόδων και τεχνικών στις εξής κατηγορίες [9]:

1. Έλεγχος Ορθότητας (Correctness Testing)

   Ο βασικότερος έλεγχος που μπορεί να γίνει σε ένα λογισμικό είναι αυτός της ορθότητάς του. Έχοντας γνώση της σωστής αλλά και της λανθασμένης συμπεριφοράς μιας εφαρμογής, υπάρχουν δύο βασικοί τρόποι αξιολόγησης:

   (α') Black-box Testing - Έλεγχος βασισμένος σε δεδομένα εισόδου/εξόδου

   Σε αυτή την περίπτωση η εφαρμογή θεωρείται ένα απομονωμένο σύστημα του οποίου η δομή παραμένει άγνωστη. Η μόνη δυνατότητα που έχουμε είναι να δίνουμε στο σύστημα συγκεκριμένα δεδομένα στην είσοδό του και να αξιολογούμε τα αποτελέσματα στην έξοδό του.

   (β') White-box Testing - Έλεγχος βασισμένος στη λογική

   Με αυτόν τον τρόπο δε βασιζόμαστε μόνο στις προδιαγραφές της εφαρμογής αλλά γνωρίζουμε πλήρως και την υλοποίησή της. Σκοπός είναι να εποπτεύσουμε όλα τα σημεία της εφαρμογής, να εντοπίσουμε όλους τους πιθανούς τρόπους εκτέλεσής της και να ελέγξουμε τον καθένα για λογικά λάθη.

2. Έλεγχος Απόδοσης (Performance Testing)

   Ορισμένα προγράμματα ενδεχομένως έχουν συγκεκριμένες απαιτήσεις στην απόδοσή τους. Γενικότερα όμως υπάρχουν σιωπηρές απαιτήσεις απόδοσης για κάθε εφαρμογή, όπως το να χρειάζονται περιορισμένους πόρους συστήματος για την εκτέλεσή τους και αυτή να ολοκληρώνεται εντός λογικών χρονικών περιθωρίων. Επομένως για αυτά είναι χρήσιμος ο έλεγχος των παραμέτρων τους που μπορεί να οδηγήσουν σε συμφόρηση των πόρων του συστήματος.

   Τέτοιες παράμετροι είναι για παράδειγμα το εύρος ζώνης δικτύου, οι κύκλοι ρολογιού της Κεντρικής Μονάδας Επεξεργασίας (CPU), ο χώρος στον δίσκο και η χρήση μνήμης [10].

   Αυτού του είδους ο έλεγχος πραγματοποιείται με ειδικές εφαρμογές αρεικρίνησης (benchmarking) που προσομοιώνουν τις τυπικές απαιτήσεις του προγράμματος που θέλουμε να αξιολογήσουμε [11].

3. Έλεγχος Αξιοπιστίας (Reliability Testing)

   Ένα πρόγραμμα θα πρέπει γενικά όταν εκτελείται να μην εμφανίζει απρόσμενη συμπεριφορά. Παρότι δεν προσδιορίζεται με ακρίβεια, δεχόμαστε ότι ένα πρόγραμμα είναι αξιόπιστο όταν δεν αποτυγχάνει με αναπάντεχους ή καταστροφικούς τρόπους [12].

   Για να επιβεβαιώσουμε την αξιοπιστία ενός προγράμματος, χρησιμοποιούμε μεθόδους stress testing (προσομοίωση ακραίων συνθηκών λειτουργίας με σκοπό την εύρεση ευάλωτων σημείων) [13] και load testing (ταυτόχρονη πρόσβαση από πολλούς χρήστες) [9].

4. Έλεγχος Ασφαλείας (Security Testing)

   Σε αντίθεση με τις προηγούμενες κατηγορίες που σχετίζονται με αυθόρμητα σφάλματα λειτουργίας, ο έλεγχος ασφαλείας μελετά περιπτώσεις σκόπιμης εκμετάλλευσης των αδυναμιών μιας εφαρμογής.

   Υπό αυτό το πρίσμα, περιλαμβάνονται όλες οι διαδικασίες που πραγματοποιούνται στις φάσεις ανάπτυξης λογισμικού και διασφαλίζουν την προστασία του από κακόβουλες επιθέσεις [14].

2.1.2 Αυτόματη παραγωγή σεναρίων ελέγχου

Όσο εξελίσσονται τα προγράμματα που χρησιμοποιούμε στην καθημερινότητά μας, τόσο αυξάνεται και η σημασία του αποτελεσματικού ελέγχου τους. Ευτυχώς παράλληλη εξέλιξη παρατηρείται και στα διαθέσιμα εργαλεία και στις τεχνικές ελέγχου που χρησιμοποιούνται [15].

Ένας τρόπος να μειωθεί ο χρόνος και το κόστος της διαδικασίας ελέγχου ενός λογισμικού είναι μέσα από την αυτοματοποίησή της [16]. Το ιδανικό θα ήταν να υπάρχει αυτόματη και πλήρης εξέταση μιας εφαρμογής αξιοποιώντας μόνο τις προδιαγραφές της, τις οποίες θα περιγράφαμε με κάποιο κατάλληλο μοντέλο [17].

Αυτό που συμβαίνει στην πράξη είναι προσπάθειες αυτοματοποίησης κάποιων τμημάτων της διαδικασίας ελέγχου. Αυτές περιλαμβάνουν για παράδειγμα την παραγωγή συγκεκριμένων σεναρίων που διασφαλίζουν την ορθότητα μιας εφαρμογής για ορισμένες περιπτώσεις [18].

2.2 Η αρχιτεκτονική REST (Representational State Transfer)

Μια διεπαφή προγραμματισμού υπολογιστών (API) είναι ένα σύνολο από διασυνδέσεις μεταξύ εφαρμογών, συστημάτων και συσκευών. Ένα πλαίσιο επικοινωνίας δηλαδή μεταξύ τους, αυστηρά καθορισμένο ως προς τη μορφή των μηνυμάτων που επιτρέπεται να ανταλλάσσονται, το περιεχόμενο που μπορεί αυτά να έχουν και τον τρόπο που δημιουργούνται και τελικά διακινούνται.

Στην πράξη είναι τμήματα λογισμικού ή υλικού που χρησιμοποιούνται με σκοπό την απλούστευση της διαδικασίας προγραμματισμού, μέσω της αφαιρετικής παρουσίασης των λειτουργιών ενός συστήματος και της εύκολης αξιοποίησής τους από τους χρήστες [2].

Το πιο δημοφιλές αρχιτεκτονικό στυλ τέτοιων διεπαφών είναι το REST (Representational State Transfer), που ορίστηκε το 2000 από τον Fielding [19] ως ένα σύνολο περιορισμών και κατευθυντήριων γραμμών σχεδιασμού. Πιο συγκεκριμένα, για να χαρακτηριστεί ένα σύστημα RESTful, δηλαδή να είναι τύπου REST, οφείλει να πληροί τους εξής περιορισμούς:

1. Αρχιτεκτονική Πελάτη-Διακομιστή (Client-Server)

   Διαχωρίζοντας το περιβάλλον του χρήστη από αυτό των δεδομένων, βελτιώνουμε τόσο την φορητότητα του περιβάλλοντος του χρήστη μεταξύ πλατφορμών, όσο και την επεκτασιμότητα, κάτι που αποτελεί ισχυρό πλεονέκτημα για τις απαιτήσεις του Διαδικτύου.

2. Έλλειψη Κατάστασης (Statelessness)

   Κάθε αίτημα από τον πελάτη προς τον διακομιστή οφείλει να περιλαμβάνει όλες τις απαραίτητες πληροφορίες για την εκτέλεση των διεργασιών. Έτσι επιτυγχάνεται ευκολότερη παρακολούθηση των εντολών, καλύτερη αξιοπιστία σε περιπτώσεις ανάκτησης δεδομένων και βελτιωμένη επεκτασιμότητα, αφού απλοποιείται η υλοποίηση των επιμέρους συστημάτων. Ο συμβιβασμός για όλα αυτά είναι η επαναλαμβανόμενη αποστολή ορισμένων δεδομένων από την πλευρά του χρήστη.

3. Δυνατότητα αξιοποίησης κρυφής μνήμης (Cacheability)

   Η προσωρινή αποθήκευση δεδομένων στην κρυφή μνήμη (cache) από τον χρήστη και τα ενδιάμεσα συστήματα βελτιώνει την απόδοση του δικτύου, αποφεύγοντας την περιττή λήψη δεδομένων, όμως εγκυμονεί κινδύνους κατοχής προηγούμενων εκδόσεων αρχείων, σε περίπτωση που αυτά έχουν ενημερωθεί από τον διακομιστή αλλά δεν έχουν ληφθεί ανανεωμένα από τον χρήστη.

4. Ομοιόμορφη Διεπαφή (Uniform Interface)

   Οι ομοιόμορφες διεπαφές μεταξύ των τμημάτων του συστήματος προσφέρουν απλούστερη αρχιτεκτονική και καλύτερη εποπτεία των αλληλεπιδράσεων. Το αρνητικό όμως είναι η μείωση της αποδοτικότητας, μιας και η πληροφορία μεταδίδεται τυποποιημένη, αγνοώντας τις ιδιαιτερότητες κάθε εφαρμογής.

5. Πολυεπίπεδο Σύστημα (Layered System)

   Μεταξύ πελάτη και διακομιστή μπορεί να παρεμβληθεί οποιοδήποτε ενδιάμεσο στάδιο, χωρίς να υπάρχει εμφανής επίδραση στην επικοινωνία και χωρίς να απαιτούνται τροποποιήσεις στις υλοποιήσεις των τελικών συστημάτων.

6. Κώδικας κατά παραγγελία (Code-On-Demand)

   Ο τελευταίος περιορισμός είναι ο μόνος προαιρετικός και αφορά στη δυνατότητα του χρήστη να κατεβάσει και να εκτελέσει τμήματα κώδικα με τη μορφή μικροεφαρμογών.

Τα RESTful Web APIs είναι προγραμματιστικές διεπαφές που ικανοποιούν τα παραπάνω κριτήρια και λειτουργούν στον Παγκόσμιο Ιστό [20]. Συνεπώς οι διεπαφές αυτού του είδους έχουν τα εξής χαρακτηριστικά [21]:

- Έχουν μία διεύθυνση βάσης (base URL), όπως *https://www.example.com/api*
- Χρησιμοποιούν τις μεθόδους HTTP, δηλαδή GET, PUT, POST, DELETE κ.λπ.
- Περιλαμβάνουν στα μηνύματά τους αναγνωριστικό για τη μορφή των δεδομένων που αποστέλλονται, όπως για παράδειγμα 'text/html' ή 'application/json'

Η πλειονότητα των εταιρειών πληροφορικής σήμερα χρησιμοποιούν προγραμματιστικές διεπαφές διαδικτύου τύπου REST για τις υπηρεσίες τους, όπως για παράδειγμα οι Google[1],

[1] https://developers.google.com/gmail/api/reference/rest

Twitter[2] και Wikipedia[3]. Η πιο συνηθισμένη χρήση είναι η δυνατότητα που προσφέρουν σε οποιονδήποτε προγραμματιστή επιθυμεί να αλληλεπιδράσει με τις εφαρμογές τους μέσω των διεπαφών τους.

Σε αυτή την περίπτωση δημιουργούν μια σειρά από τελικά σημεία, δηλαδή σημεία με τα οποία μπορεί κάποιος να αλληλεπιδράσει με τη διεπαφή. Αυτά έχουν τη μορφή ξεχωριστών διευθύνσεων που επεκτείνουν τη διεύθυνση βάσης, όπως για παράδειγμα *https://www.example.com/api/users/*, και προσφέρουν λειτουργίες που πραγματοποιούνται με την αποστολή και λήψη HTTP μηνυμάτων από και προς την ίδια διεύθυνση [22].

[2]https://developer.twitter.com/en/docs/api-reference-index
[3]https://www.mediawiki.org/wiki/API:REST_API

Κεφάλαιο 3

Σχεδιασμός

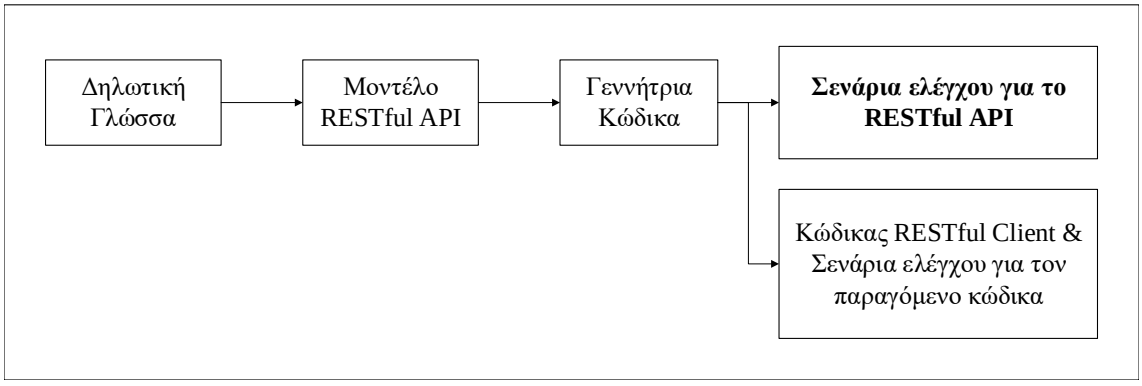
Στο κεφάλαιο περιγράφονται αναλυτικά τα αντικείμενα που σχεδιάστηκαν στο πλαίσιο της εργασίας.

Αρχικά, ορίστηκε μία δηλωτική γλώσσα ορισμού μοντέλων διεπαφών (Domain-Specific Language - DSL). Χρησιμοποιώντας τη μπορεί κανείς να περιγράψει μία προγραμματιστική διεπαφή τύπου REST, προσδιορίζοντας όλα τα χαρακτηριστικά της.

Στη συνέχεια, τα χαρακτηριστικά αυτά μοντελοποιήθηκαν με τρόπο αντικειμενοστρεφή, δηλαδή ως ξεχωριστές οντότητες. Με άλλα λόγια, θεωρούμε πως μία προγραμματιστική διεπαφή είναι ένα σύνολο αντικειμένων, το καθένα με τις δικές του ξεχωριστές ιδιότητες και λειτουργίες, που αλληλεπιδρούν μεταξύ τους.

Για να επιβεβαιώσουμε την ορθή λειτουργία της εργασίας, σχεδιάστηκαν σενάρια ελέγχου για καθένα από τα αντικείμενα που αναπαριστούν τόσο την προγραμματιστική διεπαφή, όσο και τα χαρακτηριστικά της. Αυτά τα σενάρια ελέγχου επιβεβαιώνουν όχι μόνο ότι η διεπαφή που αναπαριστάται αντιστοιχεί στις προδιαγραφές που δίνει ο χρήστης, αλλά ακόμα και ότι δεν θα δημιουργηθεί μία διεπαφή με προφανή προβλήματα, όπως για παράδειγμα διεπαφή χωρίς τελικά σημεία ή μέθοδος χωρίς αίτηση.

Τέλος, για την ολοκληρωμένη λειτουργία του συστήματος, σχεδιάστηκε μία γεννήτρια κώδικα. Αυτή είναι ένα πρόγραμμα που δέχεται την αναπαράσταση μιας προγραμματιστικής διεπαφής και παράγει σενάρια ελέγχου και βοηθητικά εργαλεία. Για να το καταφέρει αυτό, αξιοποιεί τα πρότυπα που της παρέχονται, δηλαδή έτοιμα κομμάτια κώδικα που προσαρμόζονται με βάση τα δεδομένα εισόδου.



Σχήμα 3.1: *Διάγραμμα ροής χρήσης εργασίας*

3.1 Δηλωτική γλώσσα ορισμού μοντέλων διεπαφών

Βασικός σκοπός της εργασίας είναι να μπορεί οποιοσδήποτε προγραμματιστής να περιγράφει με απλό τρόπο τα χαρακτηριστικά μιας προγραμματιστικής διεπαφής τύπου REST και να παίρνει ένα σύνολο από σενάρια ελέγχου που την καλύπτουν κατά το δυνατό περισσότερο. Για τον λόγο αυτό ορίστηκε μία δηλωτική γλώσσα ορισμού μοντέλων διεπαφών. Μέσω αυτής, μπορεί κανείς να προσδιορίσει με σαφήνεια όλες τις παραμέτρους που υποστηρίζονται, δίχως να απαιτείται να έχει γνώση του εσωτερικού τρόπου λειτουργίας της εργασίας και πώς αυτή μοντελοποιεί τα δεδομένα.

Για παράδειγμα, ας υποθέσουμε πως έχουμε μία προγραμματιστική διεπαφή τύπου REST με διεύθυνση βάσης *'/observatory/api'* και ετικέτα *'my test api'*. Έστω ότι αυτή η διεπαφή διαθέτει ένα τελικό σημείο διαπίστευσης χρηστών, το οποίο υποστηρίζει μέσω της μεθόδου POST δύο παραμέτρους σώματος για όνομα χρήστη και κωδικό πρόσβασης και σε περίπτωση επιτυχίας επιστρέφει απάντηση με κωδικό κατάστασης 201.

Ο τρόπος με τον οποίο εκφράζονται τα παραπάνω στοιχεία μέσω της δηλωτικής γλώσσας είναι ο ακόλουθος:

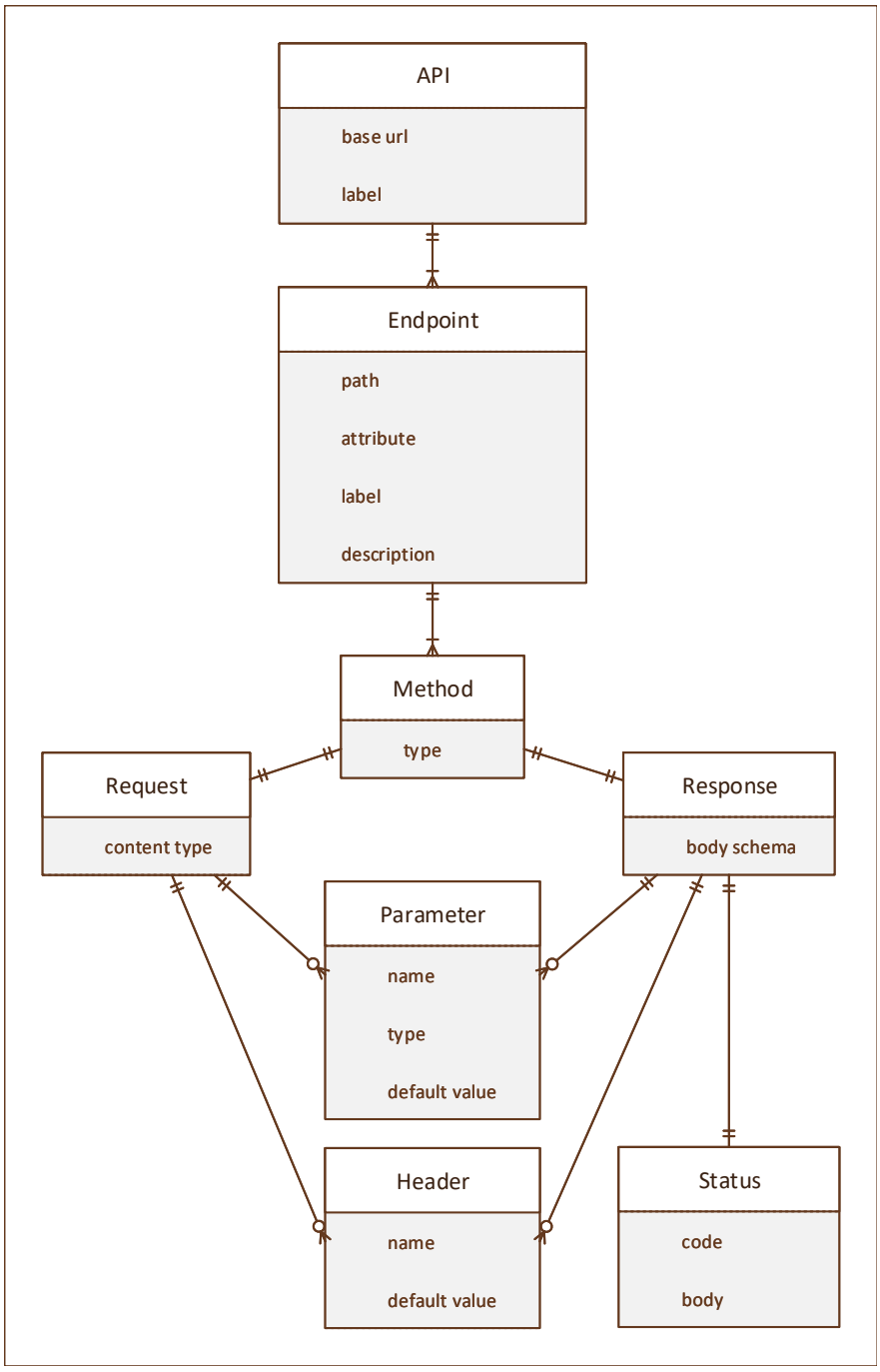
```
api {

   baseUrl '/test/api'
   label 'my test api'
   endpoint(/login) {
      label 'Endpoint login'
      description 'Endpoint for user login with username and password'
      method(POST) {
         request(URL) {
            withBodyParameter(username, String)
            withBodyParameter(password, String)
         }
         response(JSON) {
            withStatus(201) {
               body 'Created'
            }
         }
      }
   }
}
```

Η δηλωτική γλώσσα ορισμού μοντέλων διεπαφών σχεδιάστηκε με γνώμονα την ακριβέστερη απεικόνιση των χαρακτηριστικών ενός RESTful API. Κοιτώντας μία προγραμματιστική διεπαφή εκφρασμένη στη συγκεκριμένη γλώσσα μπορεί κανείς εύκολα να κατανοήσει τις λειτουργίες και τις δυνατότητές της. Παράλληλα κάθε έκφραση της γλώσσας είναι ξεκάθαρη και δεν αφήνει αμφιβολίες στον προγραμματιστή ως προς τη χρήση της.

Στο τέλος του κεφαλαίου παρατίθεται η πλήρης γραμματική της γλώσσας.

3.2 Μοντελοποίηση προγραμματιστικής διεπαφής τύπου REST

Θεωρούμε ότι κάθε προγραμματιστική διεπαφή (API) περιλαμβάνει τουλάχιστον ένα τελικό σημείο (endpoint), που το καθένα έχει τις δικές του μεθόδους (methods). Κάθε μέθοδος έχει από μία αίτηση (request) και μία απάντηση (response). Μία αίτηση αποτελείται από κεφαλίδες (headers) και παραμέτρους (parameters), οι οποίες μπορεί να είναι είτε σώματος (body) είτε αιτήματος (query). Από την άλλη, μία απάντηση αποτελείται από κεφαλίδες και μία κατάσταση (status).



Σχήμα 3.2: *Διάγραμμα οντοτήτων-συσχετίσεων*

- Προγραμματιστική Διεπαφή τύπου REST (RESTful API)

  Μία προγραμματιστική διεπαφή τύπου REST, όπως περιγράφηκε και στο Κεφάλαιο 2, είναι ένα τμήμα λογισμικού ενός συστήματος που παρέχει σε προγραμματιστές πρόσβαση σε ένα σύνολο από λειτουργίες και διαδικασίες και ακολουθεί το αρχιτεκτονικό στυλ REST (Representational State Transfer).

  Περιλαμβάνει:

  - Μία διεύθυνση βάσης (base URL), όπως https://api.example.com/
  - Μία ετικέτα (προαιρετικό), όπως 'API example'
  - Ένα τουλάχιστον τελικό σημείο (endpoint)

- Τελικό Σημείο (Endpoint)

  Ένα τελικό σημείο είναι ο δίαυλος επικοινωνίας μιας προγραμματιστικής διεπαφής με τον Παγκόσμιο Ιστό.

  Είναι μία πύλη που επιτρέπει σε προγραμματιστές να στείλουν συγκεκριμένα αιτήματα προς τη διεπαφή και να τους επιστραφούν απαντήσεις. Στην πράξη κάθε τελικό σημείο αναπαριστά μία ομάδα παρεμφερών λειτουργιών που υποστηρίζει η διεπαφή, όπως αλληλεπίδραση με δεδομένα ή εκτέλεση εντολών.

  Συνήθως ένα τελικό σημείο έχει το δικό του μονοπάτι που λειτουργεί ως πρόθεμα στη διεύθυνση της διεπαφής.

  Περιλαμβάνει:

  - Ένα μονοπάτι
  - Μία ετικέτα (προαιρετικό)
  - Μία περιγραφή (προαιρετικό)
  - Ένα ή περισσότερα προσδιοριστικά (προαιρετικά)
  - Μία τουλάχιστον μέθοδο

- Μέθοδος (Method)

  Μία μέθοδος είναι μια επιθυμητή εντολή σε κάποιον πόρο του συστήματος. Οι δυνατές εντολές είναι ορισμένες από το Πρωτόκολλο Μεταφοράς Υπερκειμένου (HyperText Transfer Protocol, HTTP) και δρουν σε ένα συγκεκριμένο τελικό σημείο της διεπαφής [23]. Σε αυτό μεταφέρουν ένα αίτημα και από αυτό δέχονται μία απάντηση.

  Περιλαμβάνει:

  - Έναν τύπο
  - Μία αίτηση
  - Μία απάντηση

  Στο πλαίσιο της εργασίας θεωρούμε ως επιτρεπτές τιμές του τύπου μιας μεθόδου τις GET, POST, PUT, PATCH και DELETE.

- Αίτηση (Request)

  Μία αίτηση είναι ένα μήνυμα που στέλνει ένας χρήστης ή ένα πρόγραμμα σε μια διεπαφή. Αυτό το μήνυμα περιέχει όλες τις απαραίτητες πληροφορίες που χρειάζεται ο διακομιστής που θα την λάβει για να την διεκπεραιώσει. Κάθε αίτηση πραγματοποιείται σε μια συγκεκριμένη διεύθυνση που συνήθως είναι ένα τελικό σημείο μιας διεπαφής.

  Περιλαμβάνει:

  - Έναν τύπο περιεχομένου (MIME)
  - Ένα σύνολο από κεφαλίδες (προαιρετικό)
  - Ένα σύνολο από παραμέτρους σώματος (προαιρετικό)
  - Ένα σύνολο από παραμέτρους αιτήματος (προαιρετικό)

  Ως τύπος περιεχομένου ορίζεται ένα αναγνωριστικό της μορφής 'τύπος/υπότυπος' που περιγράφει τη μορφή του περιεχομένου της αίτησης.

  Για τους σκοπούς της διπλωματικής, οι τύποι περιεχομένου που υποστηρίζονται είναι αποκλειστικά οι 'application/json' και 'application/x-www-form-urlencoded'.

- Απάντηση (Response)

  Μία απάντηση είναι το αποτέλεσμα που παράγεται από μια διεπαφή που δέχεται μία αίτηση. Δίνεται από το σύστημα που έχει την διεπαφή στον χρήστη ή το πρόγραμμα που στέλνει την αίτηση στο ίδιο τελικό σημείο που στάλθηκε αυτή.

  Περιλαμβάνει:

  - Ένα σχήμα σώματος
  - Ένα σύνολο από κεφαλίδες (προαιρετικό)
  - Μία κατάσταση

  Στο πλαίσιο της διπλωματική εργασίας επιλέχθηκε η αναφορά του σχήματος του σώματος της απάντησης ως προαπαιτούμενο για την αυτόματη παραγωγή των αντιστοίχων σεναρίων ελέγχου.

  Για την ακρίβεια, υποστηρίζονται οι μορφές JSON, συμβολοσειράς και ακεραίου αριθμού.

- Κεφαλίδα (Header)

  Μία κεφαλίδα είναι το μέρος ενός μηνύματος αίτησης ή απάντησης που περιλαμβάνει πρόσθετες πληροφορίες επικοινωνίας.

  Περιλαμβάνει:

  - Ένα όνομα
  - Μία προεπιλεγμένη τιμή σε περίπτωση που είναι προαιρετική και δεν παρέχεται

  Μία Κεφαλίδα πρέπει επίσης να δηλώνεται αν είναι υποχρεωτική ή προαιρετική.

- Παράμετρος (Parameter)

  Μία παράμετρος είναι ένα στοιχείο ενός μηνύματος που βοηθάει στον προσδιορισμό παραμέτρων και στην μεταφορά πρόσθετων πληροφοριών.

  Μπορεί να είναι είναι στο σώμα του μηνύματος, είτε στη διεύθυνση αιτήματος.

  Περιλαμβάνει:

  - Ένα όνομα
  - Έναν τύπο
  - Μία προεπιλεγμένη τιμή σε περίπτωση που είναι προαιρετική και δεν παρέχεται

  Μία παράμετρος πρέπει επίσης να δηλώνεται αν είναι υποχρεωτική ή προαιρετική.

- Κατάσταση (Status)

  Μία κατάσταση είναι το τμήμα της απάντησης που εκδίδεται από τον διακομιστή και περιγράφει συνοπτικά το αποτέλεσμα της αίτησης.

  Περιλαμβάνει:

  - Έναν κωδικό (πχ. 201)
  - Ένα σώμα (πχ. 'Created')

  Ανάλογα με το είδος του αποτελέσματος της αίτησης, έχουν καθιερωθεί οι ακόλουθες ομάδες κωδικών κατάστασης:

  - Ενημερωτικές απαντήσεις (100–199)
  - Επιτυχείς απαντήσεις (200–299)
  - Ανακατευθύνσεις (300–399)
  - Προβλήματα χρήστη (400–499)
  - Προβλήματα διακομιστή (500–599)

3.3 Σενάρια ελέγχου της μοντελοποίησης

Για να είμαστε βέβαιοι πως η υλοποίηση των παραπάνω αντικειμένων δεν έχει λάθη, σχεδιάστηκαν σενάρια ελέγχου για καθένα από αυτά. Έτσι, για κάθε αντικείμενο ελέγχουμε αν τα δεδομένα που του δίνουμε κατά τη δημιουργία είναι ίδια με αυτά που επιστρέφονται από αυτό που έχει παραχθεί.

Επιπλέον ελέγχουμε αν έχουν δοθεί τα υποχρεωτικά χαρακτηριστικά, τα οποία ανάλογα την προδιαγραφή είναι:

- Για την διεπαφή απαιτείται τουλάχιστον ένα τελικό σημείο.
- Για το τελικό σημείο απαιτείται το μονοπάτι και τουλάχιστον μία υποστηριζόμενη μέθοδος.
- Για τη μέθοδο απαιτούνται ο τύπος της, η αίτηση και η απάντηση.

- Για την απάντηση απαιτείται η κατάστασή της.
- Για την κεφαλίδα απαιτείται το όνομά της, ενώ σε περίπτωση που είναι υποχρεωτική απαιτείται και ένα προκαθορισμένο σώμα.
- Για την παράμετρο απαιτούνται το όνομά της και ο τύπος της, ενώ σε περίπτωση που είναι υποχρεωτική απαιτείται και ένα προκαθορισμένο σώμα.
- Τέλος, για την κατάσταση απαιτούνται ο κωδικός και το σώμα της.

Υπάρχουν ορισμένοι επιπρόσθετοι έλεγχοι για την ορθότητα των δεδομένων. Έτσι, σε μία προγραμματιστική διεπαφή ελέγχουμε αν η διεύθυνση βάσης είναι έγκυρη διεύθυνση του Παγκοσμίου Ιστού και σε μία κατάσταση αν ο κωδικός είναι εντός των υποστηριζόμενων ορίων (100-599).

3.4 Γεννήτρια Κώδικα

Έχοντας ορίσει αναλυτικά τις προδιαγραφές μιας διεπαφής προγραμματισμού τύπου REST, σχεδιάστηκε μία γεννήτρια κώδικα που λειτουργεί ως επεξεργαστής προτύπων. Ένας επεξεργαστής προτύπων είναι ένα λογισμικό που συνδυάζει πρότυπα και μοντέλα δεδομένων με σκοπό την παραγωγή ενός τελικού κειμένου [24].

Η γεννήτρια κώδικα που σχεδιάστηκε χρησιμοποιεί ως μοντέλο δεδομένων την αναπαράσταση μιας διεπαφής με τα χαρακτηριστικά που ορίστηκαν παραπάνω.

Με τα πρότυπα που σχεδιάστηκαν, η γεννήτρια κώδικα παράγει τα ακόλουθα:

3.4.1 Πελάτης (Client) RESTful API

Ο πελάτης διεπαφής είναι ένα λογισμικό που αξιοποιεί το πρωτόκολλο επικοινωνίας HTTP. Με αυτό μπορούν να πραγματοποιηθούν όλες οι μορφές επικοινωνίας που δηλώνονται στις προδιαγραφές μιας διεπαφής. Για τη διπλωματική εργασία είναι ένα απαραίτητο εργαλείο που στόχο έχει την εκτέλεση των τμημάτων των σεναρίων ελέγχου που αφορούν την επικοινωνία με μία προγραμματιστική διεπαφή τύπου REST.

Για να το δημιουργήσει η γεννήτρια κώδικα, αρχικά διαβάζει από τις προδιαγραφές της διεπαφής τη διεύθυνση βάσης, καθώς και όλα τα τελικά σημεία που υποστηρίζονται. Για καθένα από αυτά, ελέγχει αν χρησιμοποιεί κάποιο προσδιοριστικό, καθώς και ποιες μεθόδους υποστηρίζει. Λαμβάνεται υπόψη ακόμα και το σχήμα σώματος της απάντησης κάθε μεθόδου.

3.4.2 Σενάρια ελέγχου για τον παραγόμενο κώδικα με εικονικό διακομιστή (Mock Server)

Τα σενάρια ελέγχου με εικονικό διακομιστή είναι απαραίτητα για να βεβαιωθούμε πως ο πελάτης διεπαφής που θα δημιουργηθεί θα εμφανίζει την επιθυμητή συμπεριφορά.

Η γεννήτρια κώδικα, έχοντας παράξει τον πελάτη, χρησιμοποιεί τις προδιαγραφές της διεπαφής και συντάσσει μια σειρά από σενάρια ελέγχου που περιλαμβάνουν την αποστολή και λήψη μηνυμάτων μέσα από τον πελάτη.

Αρχικά αναλύει κάθε τελικό σημείο ως προς τις μεθόδους που υποστηρίζει, ενώ παράλληλα ελέγχει αν χρησιμοποιεί κάποιο προσδιοριστικό.

Στη συνέχεια, προετοιμάζει μια εικονική αίτηση που θα χρησιμοποιηθεί για τους σκοπούς του ελέγχου. Αυτή προκύπτει από τις προδιαγραφές της αίτησης κάθε μεθόδου και πιο συγκεκριμένα από τα παρακάτω στοιχεία:

Ανάλογα με τον τύπο περιεχομένου της αίτησης και πιο συγκεκριμένα αν αυτός είναι 'application/json' ή 'application/x-www-form-urlencoded', παράγεται ένα αρχείο JSON ή μια συμβολοσειρά της μορφής 'field1=value1&field2=value2' με ζεύγη ονομάτων παραμέτρων και τιμών αντίστοιχα. Σε κάθε περίπτωση περιλαμβάνονται τιμές για κάθε παράμετρο σώματος που υποστηρίζεται.

Με όμοιο τρόπο σχηματίζονται τιμές για κάθε κεφαλίδα και κάθε παράμετρο αίτησης σε μορφή πινάκων κατατεμαχισμού (hashmap).

Όπου χρειάζεται να μπει τιμή μεταβλητής σε παραμέτρους και κεφαλίδες, έχουν επιλεγεί για τους σκοπούς της διπλωματικής εργασίας οι εξής τιμές:

- Για συμβολοσειρές, τιμή της μορφής 'headerValue'
- Για ακεραίους, ο αριθμός 42
- Για αριθμούς κινητής υποδιαστολής, ο αριθμός 42,5
- Για τύπους δεδομένων Αληθείας (Boolean), η τιμή Αληθής/True

Σε μεθόδους τύπου GET, PUT, PATCH και DELETE που χρησιμοποιούν προσδιοριστικό, αυτό θεωρείται πως είναι ακέραιος αριθμός και για τους σκοπούς των ελέγχουν του ανατίθεται η τιμή 2.

Αμέσως μετά προδιαγράφεται ένας εικονικός διακομιστής (Mock Server). Η λειτουργία του είναι όταν δέχεται στη διεύθυνση του τελικού σημείου αίτηση με την αντίστοιχη μέθοδο και με τις κεφαλίδες και τις παραμέτρους σώματος και αίτησης των προδιαγραφών, τότε να επιστρέφει μία συγκεκριμένη απάντηση στον πελάτη. Αυτή θα έχει την κατάσταση, τις κεφαλίδες της και το σώμα της κατάστασης. Το τελευταίο επιλέγεται ανάλογα με τον τύπο σχήματος και μπορεί να είναι είτε της μορφής JSON, είτε συμβολοσειρά είτε ακέραιος.

Η διαδικασία αυτή επαναλαμβάνεται για κάθε μέθοδο κάθε τελικού σημείου της διεπαφής.

3.4.3 Σενάρια ελέγχου για το RESTful API

Τέλος, τα σενάρια ελέγχου για το μοντέλο RESTful API παράγονται με παρόμοιο τρόπο με τα προαναφερθέντα, με βασική διαφορά την έλλειψη εικονικού διακομιστή. Αντί για αυτόν, η γεννήτρια κώδικα συντάσσει σενάρια ελέγχου που πραγματοποιούν αιτήσεις στη διεύθυνση και τη θύρα που έχουν οριστεί από τον χρήστη και επαληθεύουν το αποτέλεσμα που δέχεται ο πελάτης διεπαφής.

Οι αιτήσεις προετοιμάζονται με τον ίδιο τρόπο ακριβώς με παραπάνω, λαμβάνοντας υπόψη όλα τα χαρακτηριστικά της διεπαφής.

Οι έλεγχοι καλύπτουν:

- Την ομαλή λειτουργία του διακομιστή και την επιτυχή σύνδεση σε αυτόν μέσω της θύρας που δίνεται.
- Τη δυνατότητα αποστολής μιας αίτησης με τις παραμέτρους και τις κεφαλίδες που υποστηρίζονται.
- Αν η απάντηση έχει τη μορφή που αναμένεται.
- Την ύπαρξη στις κεφαλίδες της απάντησης των πεδίων που αναμένονται.
- Την αποτυχία μιας αίτησης σε περίπτωση που λείπει μία υποχρεωτική παράμετρος ή κεφαλίδα.

3.5 Γραμματική δηλωτικής γλώσσας ορισμού μοντέλων διεπαφών

```
API : '{' 'baseUrl' BASEURL ('label' LABEL)? (('endpoint(' PATH+ ')' | ('endpoint('
    PATH ',' ATTRIBUTE ')') ENDPOINT)+ '}'

ENDPOINT : '{' ('label' LABEL)? ('description' DESCRIPTION)? 'method(' METHODTYPE ')'
    METHOD+ '}'

METHOD : '{' 'request(' REQUESTTYPE ')' REQUEST 'response(' RESPONSESCHEMA ')'
    RESPONSE '}'

REQUEST : '{' ('withBodyParameter(' PARAMETER ')')?+
                  ('withQueryParameter(' PARAMETER ')')?+
                  ('withHeader(' HEADER ')')?+ '}'

RESPONSE : '{' 'withStatus' STATUS ('withHeader(' HEADER ')')?+ '}'
STATUS : '(' INT ')' '{' 'body' WORD '}'
PARAMETER : '(' (  ',' TYPE | NAME ',' TYPE ',' DEFAULTVALUE )')'
HEADER : '(' (  | NAME ',' DEFAULTVALUE )')'
BASEURL : WORD
LABEL : WORD
PATH : '/' WORD
ATTRIBUTE : WORD
METHODTYPE : 'GET' | 'POST' | 'PUT' | 'PATCH' | 'DELETE'
REQUESTTYPE : 'JSON' | 'URL'
RESPONSESCHEMA : 'JSON' | 'String' | 'Integer'
NAME : WORD
TYPE : WORD
DEFAULTVALUE : WORD
INT : [0-9]+
```

Κεφάλαιο 4

Υλοποίηση

Στο κεφάλαιο αυτό παρουσιάζεται η υλοποίηση της εργασίας. Περιγράφεται το περιβάλλον στο οποίο αναπτύχθηκε και αναλύονται οι γλώσσες και τα εργαλεία προγραμματισμού που επιλέχθηκαν.

4.1 Ολοκληρωμένο Περιβάλλον Ανάπτυξης (IDE)

Η εργασία υλοποιήθηκε στο ολοκληρωμένο περιβάλλον ανάπτυξης IntelliJ IDEA Ultimate 2020[1]. Αυτό είναι ένα λογισμικό ανάπτυξης εφαρμογών που περιλαμβάνει μεταξύ άλλων έναν επεξεργαστή πηγαίου κώδικα, έναν μεταγλωττιστή και εξειδικευμένα πρόσθετα υποβοήθησης προγραμματισμού.

Επιλέχθηκε το συγκεκριμένο πρόγραμμα εξαιτίας της καθολικής υποστήριξης των γλωσσών προγραμματισμού που επιλέχθηκαν και των εργαλείων που χρησιμοποιηθήκαν.

4.2 Γλώσσα Προγραμματισμού

Η γλώσσα προγραμματισμού που επιλέχθηκε για την αναπαράσταση των προδιαγραφών μιας προγραμματιστικής διεπαφής τύπου REST είναι η Java.

Η Java είναι μία γλώσσα αντικειμενοστρεφούς προγραμματισμού που σχεδιάστηκε από τον James Gosling και κυκλοφόρησε από την Sun Microsystems το 1995. Οι λόγοι προτίμησης της συγκεκριμένης γλώσσας συμπίπτουν με τους πρωταρχικούς στόχους δημιουργίας της και πιο συγκεκριμένα επειδή είναι ανεξάρτητη από την πλατφόρμα του εξυπηρετητή, περιλαμβάνει εργαλεία και βιβλιοθήκες διαδικτύου και έχει σχεδιαστεί για να εκτελεί κώδικα από εξωτερικές πηγές με ασφάλεια [25].

Στο πλαίσιο του αντικειμενοστρεφούς προγραμματισμού, η Java χρησιμοποιεί την έννοια της κλάσης για να προσδιορίζει μια κατηγορία αντικειμένων και παράλληλα να περιγράφει τα χαρακτηριστικά και τις λειτουργίες τους [26]. Αυτή ακριβώς η έννοια αξιοποιήθηκε για την περιγραφή των προδιαγραφών μιας προγραμματιστικής διεπαφής τύπου REST όπως αυτές σχεδιάστηκαν στο Κεφάλαιο 3.

[1] https://www.jetbrains.com/idea/

4.2.1 Διεπαφές (Interfaces)

Οι διεπαφές της Java είναι σύνολα ορισμένων αλλά μη υλοποιημένων λειτουργιών και διαδικασιών. Αυτές ονομάζονται μέθοδοι και υλοποιούνται από κάποια κλάση[2].

Αρχικά, δημιουργήθηκαν διεπαφές (interfaces) για καθένα από τα αντικείμενα που χρησιμοποιηθήκαν, δηλαδή την προγραμματιστική διεπαφή τύπου REST, το τελικό σημείο, τη μέθοδο, την αίτηση, την απάντηση, την κεφαλίδα, την παράμετρο και την κατάσταση. Κάθε διεπαφή περιλαμβάνει τις απαραίτητες διαχειριστικές μεθόδους ανάλογα με τα χαρακτηριστικά της οντότητας που περιγράφει. Καθεμιά από αυτές ορίζει τη δυνατότητα ενός αντικειμένου να επιστρέφει ως απάντηση στην κλήση της το αντίστοιχο χαρακτηριστικό των προδιαγραφών.

Για παράδειγμα, μία διεπαφή προγραμματιστικής διεπαφής περιλαμβάνει μεθόδους για τη διεύθυνση βάσης, την ετικέτα της και το σύνολο από τα τελικά σημεία της.

Για λόγους ευκολίας, από τη στιγμή που η εργασία υποστηρίζει τους τύπους περιεχομένου 'application/json' και 'application/x-www-form-urlencoded', ο περιορισμός αυτός μεταφέρθηκε στον κώδικα στο σημείο των διεπαφών. Έτσι, πέρα από την γενική διεπαφή της αίτησης, υπάρχουν δύο επιπλέον, η RequestJSON και η RequestURL που έχουν προκαθορισμένο τον αντίστοιχο τύπο. Αυτές είναι που χρησιμοποιούνται στην υπόλοιπη εργασία.

4.2.2 Υλοποιήσεις (Implementations)

Μια κλάση στη Java θεωρείται ότι υλοποιεί μία διεπαφή όταν περιλαμβάνει τις μεθόδους που έχουν οριστεί από αυτήν και περιγράφει τον τρόπο που εκτελούνται από το αντικείμενο.

Επομένως για κάθε διεπαφή δημιουργήθηκε μία αντίστοιχη κλάση, η οποία υλοποιεί όλες τις μεθόδους και επιπλέον περιέχει μεταβλητές για τα επιμέρους χαρακτηριστικά της, καθώς και μία συνάρτηση δόμησης (constructor).

Οι κλάσεις ακολουθούν την τεχνική της ενθυλάκωσης (encapsulation). Πιο συγκεκριμένα, τα ευαίσθητα δεδομένα των μεταβλητών προστατεύονται από τις μεθόδους της κλάσης, οι οποίες τα διαχειρίζονται και επιτρέπουν την πρόσβαση σε αυτά μόνο για ανάγνωση και όχι για επεξεργασία.

Όλες οι μεταβλητές των κλάσεων που υλοποιούν διεπαφές είναι ιδιωτικές (private), που σημαίνει πως πρόσβαση σε αυτές έχουν μόνο οι μέθοδοι της ίδιας κλάσης.

Αντίθετα, οι διαχειριστικές μέθοδοι που υλοποιούνται είναι όλες δημόσιες (public). Αυτό σημαίνει πως ένα αντικείμενο μπορεί να μεταβιβάσει σε οποιοδήποτε άλλο αντικείμενο, είτε ίδιας είτε διαφορετικής κλάσης, πληροφορίες για τα χαρακτηριστικά του, χωρίς όμως να δίνει πρόσβαση στις ίδιες τις μεταβλητές, κάτι που θα εγκυμονούσε κινδύνους ασφαλείας και προβλήματα αστάθειας.

Για παράδειγμα, η κλάση που υλοποιεί τη διεπαφή της προγραμματιστικής διεπαφής τύπου REST περιλαμβάνει τις ιδιωτικές μεταβλητές της συμβολοσειράς διεύθυνσης βάσης, της συμβολοσειράς ετικέτας και του συνόλου από αντικείμενα της κλάσης του τελικού σημείου. Για τις τρεις αυτές ιδιωτικές μεταβλητές υπάρχουν οι αντίστοιχες δημόσιες μέθοδοι, ίδιου τύπου, χωρίς παραμέτρους, που έχουν ως έξοδο στην κλήση τους τις αντίστοιχες μεταβλητές.

[2]https://docs.oracle.com/javase/tutorial/java/concepts/interface.html

Τέλος, υπάρχει η δημόσια συνάρτηση δόμησης που δέχεται ως παραμέτρους τρεις μεταβλητές, μία συμβολοσειρά για την διεύθυνση βάσης, μία συμβολοσειρά για την ετικέτα και ένα σετ από αντικείμενα τελικού σημείου για τα τελικά σημεία. Η συνάρτηση αυτή χρησιμοποιείται για τη δημιουργία αντικειμένων της κλάσης της προγραμματιστικής διεπαφής τύπου REST με χαρακτηριστικά που αντιστοιχούν στις μεταβλητές που δέχεται ως παραμέτρους.

4.2.3 Σχεδιαστικό Πρότυπο

Ως πρότυπο σχεδίασης της εργασίας επιλέχθηκε αυτό των κατασκευαστών (builder pattern). Σκοπός του προτύπου σχεδίασης των κατασκευαστών είναι ο διαχωρισμός της διαδικασίας κατασκευής ενός σύνθετου αντικειμένου από την παρουσίασή του [27].

Με άλλα λόγια, για κάθε κλάση δημιουργήθηκε μία επιπλέον, ένας κατασκευαστής για αντικείμενα της αντίστοιχης κλάσης. Ο κάθε κατασκευαστής, πέρα από ιδιωτικές μεταβλητές για τα χαρακτηριστικά του αντικειμένου που δημιουργεί, περιέχει και δημόσιες μεθόδους που επιτρέπουν την επεξεργασία των μεταβλητών. Αυτές δέχονται μέσω των ορισμάτων τους τιμές για τις αντίστοιχες μεταβλητές και στο τέλος δημιουργούν ένα αντικείμενο με τα χαρακτηριστικά που έχει δώσει βήμα-βήμα ο προγραμματιστής ή το αντικείμενο που καλεί τον κατασκευαστή.

Στη συνάρτηση που δημιουργεί το τελικό αντικείμενο εμπεριέχονται και οι έλεγχοι για την ορθότητά του με βάση τους περιορισμούς που ορίστηκαν κατά την σχεδίαση των αντικειμένων στο Κεφάλαιο 3. Εφόσον τα δεδομένα που δέχεται ο κατασκευαστής δεν είναι έγκυρα, η διαδικασία κατασκευής τερματίζεται και εγείρεται μήνυμα σφάλματος (Runtime Exception). Αλλιώς καλείται η συνάρτηση δόμησης της αντίστοιχης κλάσης, η οποία δέχεται ως ορίσματα τις μεταβλητές του κατασκευαστή με τις τελικές τιμές του, όπως ορίστηκαν από τις μεθόδους του, σε μορφή αμετάβλητων (immutable) αντικειμένων.

Για παράδειγμα, για τη δημιουργία ενός αντικειμένου της κλάσης της προγραμματιστικής διεπαφής τύπου REST, αρκεί να κληθεί ο αντίστοιχος κατασκευαστής με την εξής μορφή:

```
APISpecBuilder newApiBuilder = new APISpecBuilder();
APISpec newAPI = newApiBuilder
            .setLabel("api")
            .setBaseUrl("https://www.example.com/api")
            .addEndpoint(newEndpoint)
            .build();
```

Με αυτόν τον τρόπο πετυχαίνουμε μεγαλύτερη ευχέρεια στην κατασκευή των αντικειμένων, ενώ παράλληλα ισχυροποιείται ο έλεγχος που έχει ο προγραμματιστής κατά τη διαδικασία, ελέγχοντας ταυτόχρονα την εγκυρότητα των δεδομένων.

Οι κατασκευαστές αντικειμένων που έχουν χαρακτηριστικά που αποτυπώνονται με σύνολα, όπως για παράδειγμα μια προγραμματιστική διεπαφή που μπορεί να έχει πολλαπλά τελικά σημεία, περιλαμβάνουν μεθόδους για την προσθήκη τόσο ενός από αυτά (πχ. addEndpoint) όσο και περισσοτέρων (πχ. addEndpoints). Σε αντίθεση με τις μεθόδους για μεταβλητές που δέχονται μοναδική τιμή και αντικαθιστούν την τιμή με το όρισμα που δέχονται

(πχ. setBaseUrl), κάθε κλήση μιας μεθόδου προσθήκης δεδομένων διατηρεί τις υπάρχουσες τιμές του συνόλου και προσθέτει σε αυτές τα ορίσματά της.

4.3 Κατασκευαστής Groovy

Με βάση τη δηλωτική γλώσσα ορισμού μοντέλων διεπαφής που σχεδιάστηκε, υλοποιήθηκε ένας κατασκευαστής που υποστηρίζει πλήρως τη γραμματική της. Σκοπός του είναι να δέχεται από τον χρήστη την περιγραφή της διεπαφής για την οποία θέλει να παράξει σενάρια ελέγχου και να δημιουργεί την αντίστοιχη μοντελοποίηση από αντικείμενα Java. Όπως βλέπουμε και στο σχήμα 4.1, στην είσοδό του δέχεται την δηλωτική περιγραφή ενός RESTful API, την οποία αναλύει με σκοπό να παράξει ένα runtime μοντέλο του, δηλαδή στιγμιότυπα (instances) των κλάσεων Java που αναπαριστούν τα χαρακτηριστικά του.

Η γλώσσα προγραμματισμού που επιλέχθηκε για την υλοποίηση του συγκεκριμένου κατασκευαστή είναι η Groovy. Η Groovy είναι και αυτή μία γλώσσα αντικειμενοστρεφούς προγραμματισμού, παρόμοιου συντακτικού με την Java. Ένα βασικό χαρακτηριστικό της γλώσσας Groovy που αξιοποιήθηκε για την εργασία είναι η υποστήριξη κομματιών κώδικα που λέγονται closures. Ένα closure μπορεί να ανατεθεί σε μια μεταβλητή, να δέχεται ορίσματα και να επιστρέφει τιμές.

Στον Groovy κατασκευαστή που υλοποιήθηκε (Groovy Builder), χρησιμοποιήθηκαν ως closures οι περιγραφές που δίνει ο χρήστης για την προγραμματιστική διεπαφή του και τις ιδιότητές της. Για παράδειγμα, ο ορισμός μιας διεπαφής γίνεται με την κλήση της συνάρτησης api(), η οποία δέχεται ως όρισμα ένα closure στο οποίο περιλαμβάνονται η διεύθυνση βάσης, η ετικέτα και τα τελικά σημεία.

Ο τρόπος που θα κατασκευάζαμε μέσω του Groovy Builder μια προγραμματιστική διεπαφή με ένα τελικό σημείο διαπίστευσης χρηστών είναι ο ακόλουθος:

```
api {

   baseUrl '/test/api'
   label 'my test api'
   endpoint('/login') {
      label 'Endpoint login'
      description 'Endpoint for user login with username and password'
      method('POST') {
         request('URL') {
            withBodyParameter('username', 'String')
            withBodyParameter('password', 'String')
         }
         response('JSON') {
            withStatus(201) {
               body 'Created'
            }
         }
      }
```

```
    }
}
```

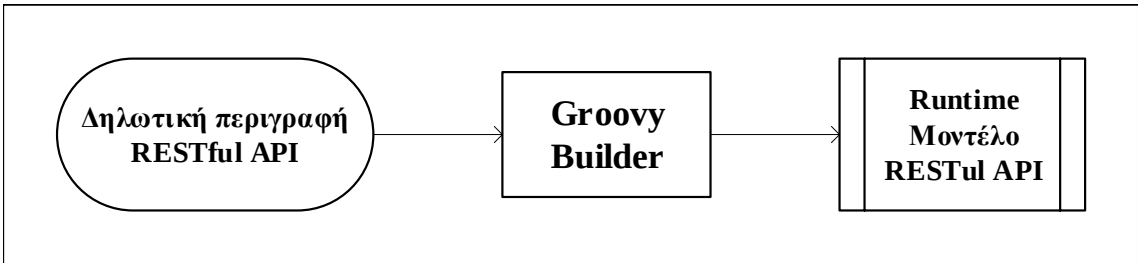
Επιπλέον ο Groovy Builder δέχεται υποχρεωτικά ορισμένες πληροφορίες για αρχεία που παράγει η γεννήτρια κώδικα. Οι πληροφορίες αυτές είναι οι τοποθεσίες, τα πακέτα και τα ονόματα του REST API Client και των σεναρίων ελέγχου, καθώς και την θύρα (port) του server του χρήστη.

4.3.1 Μέθοδοι

```
baseUrl(String baseUrl)
label(String label)
endpoint(String path, Closure configuration)
endpoint(String path, String attribute, Closure configuration)
description(String description)
method(String type, Closure configuration)
request(String type, Closure configuration) # type: JSON or URL
response(String schema, Closure configuration)
withBodyParameter(String name, String type)
withBodyParameter(String name, String type, Object defaultValueIfOptionalAndMissing)
withQueryParameter(String name, String type)
withQueryParameter(String name, String type, Object defaultValueIfOptionalAndMissing)
withHeader(String name)
withHeader(String name, String defaultValueIfOptionalAndMissing)
withStatus(Integer code, Closure configuration)
body(String body)
```

4.3.2 Μεταβλητές

```
// Rest API Client
clientFolder # "/Project/src/main/java/"
clientPackage # "app/restapiclient"
clientName # "RestAPIClient"
serverPort # 8765

// Rest API Server Tests
testFolder # "/Project/src/test/groovy/"
testPackage # "app/restapitests"
testName # "TestServer"

// Rest API Client Tests with Mock Server
mockFolder # "/Project/src/test/groovy/"
mockPackage # "app/restapimocktests"
mockName # "MockServer"
```



Σχήμα 4.1: *Διάγραμμα ροής κατασκευαστή*

4.4 Πλαίσιο Ελέγχου Spock

Το Spock είναι ένα πλαίσιο ελέγχου για Java και Groovy εφαρμογές, ικανό να διαχειριστεί τον πλήρη κύκλο ανάπτυξης ενός λογισμικού συμβάλλοντας στην αυτοματοποίηση του ελέγχου [28].

Με τη χρήση του εργαλείου Spock πραγματοποιούνται όλοι οι έλεγχοι στο πλαίσιο της εργασίας. Ο τρόπος με τον οποίο γίνεται αυτό είναι ότι κάθε έλεγχος έχει τη μορφή ενός σεναρίου, όπου δίνεται μία αρχική κατάσταση και παρατηρούμε αν ισχύουν οι συνθήκες που επιθυμούμε μετά την εκτέλεση ορισμένων εντολών.

Πιο συγκεκριμένα, για την αξιοποίησή του σε μία εφαρμογή Java πρέπει πρώτα να εισάγουμε την αντίστοιχη βιβλιοθήκη και μετά να δημιουργήσουμε κλάσεις Groovy που επεκτείνουν την *'spock.lang.Specification'*. Σε αυτές γράφουμε κομμάτια κώδικα ('blocks') που ακολουθούν τις φάσεις ('phases') setup, stimulus, response και cleanup.

Στη φάση setup ορίζονται οι αρχικές συνθήκες του σεναρίου, όπως οι μεταβλητές και τα αντικείμενα που θα χρησιμοποιηθούν στην πορεία. Στις φάσεις stimulus και response περιλαμβάνεται ο κύριος κώδικας του ελέγχου, όπου εκτελούνται μία ή περισσότερες ενέργειες και ελέγχονται τα αποτελέσματά τους. Η τελευταία φάση cleanup αφορά απελευθέρωση πόρων του συστήματος και εκτελείται ακόμα κι αν προηγουμένως έχει εγερθεί κάποια εξαίρεση.

Ο τρόπος με τον οποίο αποφασίζει το Spock αν ένα σενάριο ελέγχου επιτυγχάνει ή αποτυγχάνει είναι να παρατηρεί τις συνθήκες που ορίζονται στη φάση response και που αφορούν τα δεδομένα που προκύπτουν από τη φάση stimulus. Αν αυτές ικανοποιούνται όλες, τότε το σενάριο ελέγχου θεωρείται ότι επιτυγχάνει. Αλλιώς, αν ακόμα και μία δεν ικανοποιείται, τότε αποτυγχάνει.

Ακολουθεί ένα παράδειγμα χρήσης του εργαλείου Spock. Σε αυτό ελέγχουμε αν η διαδικασία αφαίρεσης ενός στοιχείου από μία λίστα ακεραίων αριθμών λειτουργεί σωστά.

```
def "Remove element from list"() {
   given:
      def list = [1, 2, 4, 8]

   when:
      list.remove(0)

   then:
```

```
    list == [2, 4, 8]
}
```

Στο κομμάτι κώδικα 'given' που αντιστοιχεί στη φάση setup ορίζουμε την αρχική κατάσταση του σεναρίου ελέγχου και πιο συγκεκριμένα δημιουργούμε μία λίστα με τέσσερις ακεραίους αριθμούς. Στο κομμάτι κώδικα 'when' που αντιστοιχεί στη φάση stimulus ορίζουμε την εντολή που θέλουμε να εκτελεστεί και της οποίας το αποτέλεσμα θέλουμε να ερευνήσουμε. Στην περίπτωσή μας είναι η αφαίρεση του πρώτου στοιχείου από τη λίστα. Τέλος, στο κομμάτι κώδικα 'then' που αντιστοιχεί στη φάση response ορίζουμε την συνθήκη που θέλουμε να ισχύει μετά το πέρας της εντολής αφαίρεσης, δηλαδή η λίστα να περιέχει πλέον όλα τα υπόλοιπα στοιχεία πέρα από το πρώτο που αφαιρέθηκε.

4.5 Μηχανή Προτύπων Freemarker

Η γεννήτρια κώδικα υλοποιήθηκε με χρήση της μηχανής προτύπων Freemarker.

Το Freemarker είναι μία μηχανή προτύπων (Template Engine) που χρησιμοποιείται ως βιβλιοθήκη της Java με σκοπό την παραγωγή αρχείων κειμένου, όπως στην περίπτωσή μας πηγαίος κώδικας.

Για τη λειτουργία της απαιτούνται έτοιμα κομμάτια κώδικα που προσαρμόζουν τμήματά τους με βάση τα δεδομένα εισόδου της μηχανής προτύπων. Αυτά τα κομμάτια κώδικα είναι γραμμένη σε μια συγκεκριμένη γλώσσα που ορίζει η βιβλιοθήκη και τα δεδομένα που προσαρμόζονται λαμβάνονται από αντικείμενα Java.

Για παράδειγμα, έστω ότι δίνουμε στη μηχανή προτύπων δώσουμε ένα αντικείμενο διεπαφής με διεύθυνση βάσης '/my/api' και το πρότυπο περιλαμβάνει την ακόλουθη γραμμή:

```
String BASE_URL = "${api.baseUrl}"
```

Τότε το παραγόμενο αρχείο θα αντικαταστήσει τη μεταβλητή με την τιμή που δίνει το αντικείμενο:

```
String BASE_URL = "/my/api"
```

4.6 Εργαλείο Κατασκευής Λογισμικού Gradle

Το Gradle είναι ένα από τα πιο διαδεδομένα εργαλεία αυτοματοποίησης της κατασκευής λογισμικού. Υποστηρίζει τις σχετικές διεργασίες όλων των σταδίων ανάπτυξης μιας εφαρμογής, από τη μεταγλώττιση του πηγαίου κώδικα μέχρι την εκτέλεση σεναρίων ελέγχου και τη δημοσίευση του συστατικού (artifact) σε διαδικτυακές αποθήκες. Υποστηρίζει πολλές γλώσσες προγραμματισμού, με τις πιο δημοφιλείς να είναι η Java, η C/C++ και η JavaScript.

Το Gradle λειτουργεί με σενάρια κατασκευής (build scripts) που περιλαμβάνουν όλα τα projects που αναλαμβάνει να αναπτύξει. Κάθε project αποτελείται από εργασίες (tasks). Αυτές είναι εντολές που μπορεί να αφορούν την εκτέλεση κώδικα, τη μεταγλώττιση ενός προγράμματος και την είσοδο ή έξοδο αρχείων ή καταλόγων.

Για την οργάνωση των εργασιών που πρέπει να γίνουν, το Gradle κατασκευάζει έναν κατευθυνόμενο άκυκλο γράφο (DAG) με βάση τις εξαρτήσεις τους. Μέσα από τον γράφο

καθορίζεται η προτεραιότητα των εργασιών, καθώς κάθε εργασία για να πραγματοποιηθεί πρέπει πρώτα να έχουν ολοκληρωθεί όσες αντιστοιχούν σε προηγούμενους κόμβους.

Η διαδικασία κατασκευής του Gradle αποτελείται από τρεις φάσεις. Στη φάση της αρχικοποίησης (Initialization) ρυθμίζεται το περιβάλλον εργασίας και εντοπίζονται τα projects που παίρνουν μέρος στη διαδικασία. Στη φάση της διαμόρφωσης (Configuration) δημιουργείται ο γράφος εργασιων και καθορίζεται η σειρά εκτέλεσής τους. Τέλος, στη φάση εκτέλεσης (Execution) εκτελούνται οι εργασίες με βάση την προτεραιότητα που αποφασίστηκε πριν.

Για την οργάνωση των projects, το Gradle χρησιμοποιεί αρχεία με όνομα 'build.gradle'. Σε αυτά περιλαμβάνονται όλες οι απαραίτητες πληροφορίες για τις εργασίες του Gradle, όπως τα plugins που χρησιμοποιούνται, τα συστατικά (artifacts) στα οποία υπάρχουν εξαρτήσεις, καθώς και τα αποθετήρια στα οποία βρίσκονται. Στο αρχείο αυτό συμπεριλαμβάνονται και οι ορισμοί των εργασιών.

Στο πλαίσιο της εργασίας, ο Groovy Builder υλοποιήθηκε έτσι ώστε να μπορεί να χρησιμοποιηθεί ως ένα Gradle plugin. Πιο συγκεκριμένα, για τη χρήση του σε ένα project πρέπει πρώτα να υπάρχει ο αντίστοιχος κώδικας σε έναν φάκελο με όνομα 'buildSrc'. Στη συνέχεια, ο χρήστης με τις παρακάτω εντολές εισάγει τον Groovy Builder ως plugin:

```
apply plugin: GroovyApiSpecBuilder
```

Έπειτα αρκεί να γράψει τον κώδικα στη δηλωτική γλώσσα DSL που διαβάζει ο Groovy Builder, καθώς και να παραθέσει τις απαραίτητες πληροφορίες για τον διακομιστή και τους φακέλους αποθήκευσης. Η παραγωγή των σεναρίων ελέγχου, καθώς και του κώδικα RESTful API και των σεναρίων ελέγχου για τον παραγόμενο κώδικα, πραγματοποιείται με την εντολή 'generate'.

Κεφάλαιο 5

Επαλήθευση Ορθής Λειτουργίας

Στο κεφάλαιο αυτό παρουσιάζονται οι εφαρμογές σε δύο διαδικτυακά RESTful APIs που επαληθεύουν την ορθή λειτουργία των εργαλείων που αναπτύχθηκαν στο πλαίσιο της παρούσας εργασίας.

5.1 Παρατηρητήριο τιμών

Η πρώτη προγραμματιστική διεπαφή τύπου REST που χρησιμοποιήθηκε για την επαλήθευση της εργασίας είναι τμήμα μιας εφαρμογής παρατηρητηρίου τιμών. Ο κώδικας είναι διαθέσιμος στο διαδίκτυο σε github repository[1].

Τα χαρακτηριστικά της διεπαφής είναι τα ακόλουθα:

- **Διεύθυνση**

  Ως διεύθυνση βάσης χρησιμοποιείται το 'https://localhost:8765/observatory/api'.

  Οι πόροι του συστήματος δίνονται από τα τελικά σημεία στη μορφή {baseURL}/{path-to-resource}.

  Έτσι για παράδειγμα το τελικό σημείο για τα προϊόντα είναι το {baseURL}/products, ενώ για το προϊόν με κωδικό 12 το {baseURL}/products/12.

- **Διαπίστευση**

  Το τελικό σημείο διαπίστευσης χρηστών έχει το μονοπάτι '{baseURL}/login'.

  Η μοναδική μέθοδος που υποστηρίζει είναι η POST.

  Δέχεται ως παραμέτρους σώματος το username και το password, ενώ σε περίπτωση επιτυχούς σύνδεσης επιστρέφει ένα authentication token. Αυτό θα πρέπει να περιλαμβάνεται ως κεφαλίδα με όνομα 'X-OBSERVATORY-AUTH' σε όποια αίτηση απαιτείται πιστοποίηση.

- **Αποσύνδεση**

  Τπ τελικό σημείο αποσύνδεσης χρηστών έχει το μονοπάτι '{baseURL}/logout'.

  Η μοναδική μέθοδος που υποστηρίζει είναι η POST.

[1] https://github.com/PavlosSta/SoftEng

Θα πρέπει να έχει προηγηθεί διαπίστευση, η οποία θα έχει επιστρέφει ένα authentication token. Αυτό θα πρέπει να περιλαμβάνεται στην αίτηση ως κεφαλίδα ταυτοποίησης με όνομα 'X-OBSERVATORY-AUTH' και αν είναι έγκυρο, η απάντηση θα είναι ένα JSON της μορφής '{ "message": "OK" }'.

- **Προϊόντα**

  Το τελικό σημείο των προϊόντων έχει το μονοπάτι '{baseURL}/products'.

  Σε αυτό υποστηρίζονται οι ακόλουθες μέθοδοι:

  1. **GET**: Επιστρέφεται η λίστα των προϊόντων

     Υποστηριζόμενες παράμετροι (ως μέρος του URL query):

     - start: Integer, default 0
     - count: Integer, default 20
     - status: String, default ACTIVE
     - sort: String, default id|DESC

     Για παράδειγμα:

     GET *{baseURL}/products?start=0&count=100&sort=id|ASC&status=ALL*

     Η απάντηση είναι ένα αρχείο JSON που περιλαμβάνει τις εξής παραμέτρους:

     - start: Integer
     - count: Integer
     - total: Integer
     - products: List<Product>

     Οι τρεις πρώτες παράμετροι αφορούν τη σελίδοποίηση των αποτελεσμάτων, ενώ η τελευταία περιλαμβάνει μία λίστα με τα προϊόντα της αντίστοιχης σελίδας.

     Κάθε προϊόν είναι ένα αρχείο JSON με τις εξής παραμέτρους:

     - id: String
     - name: String
     - description: String
     - category: String
     - tags: List<String>
     - withdrawn: Boolean

  2. **POST**: Δημιουργείται νέο προϊόν

     Οι πληροφορίες που απαιτούνται για τη δημιουργία ενός προϊόντος διατυπώνονται ως παράμετροι σώματος στην αίτηση. Αυτές είναι το name, το description, το category και τα tags.

     Προαιρετική παράμετρος είναι το withdrawn.

     Η απάντηση στην αίτηση δημιουργίας ενός προϊόντος είναι η πλήρης κωδικοποίηση των δεδομένων του, δηλαδή ένα αρχείο JSON με τις αντίστοιχες πληροφορίες.

Επιπλέον υποστηρίζει προσδιοριστικό (attribute) για το id των προϊόντων, με μονοπάτι της μορφής '{baseURL}/products/{id}'.

Με το id υποστηρίζονται οι ακόλουθες μέθοδοι:

1. **GET**: Επιστρέφονται οι πληροφορίες του προϊόντος
   Για παράδειγμα, GET *{baseURL}/products/2*.
   Το αποτέλεσμα της αίτησης είναι ένα αρχείο JSON με τα δεδομένα του προϊόντος με το αντίστοιχο id.
   Αυτά όπως και στην περίπτωση της μεθόδου GET χωρίς προσδιοριστικό id είναι τα ακόλουθα:
   - id: String
   - name: String
   - description: String
   - category: String
   - tags: List<String>
   - withdrawn: Boolean
2. **PUT**: Ενημερώνονται οι πληροφορίες του προϊόντος
   Η αίτηση πρέπει να περιλαμβάνει δεδομένα για όλα τα στοιχεία του προϊόντος, τα οποία αντικαθιστούν τα προηγούμενα (Full Update).
   Τα δεδομένα αυτά περιλαμβάνονται στην αίτηση ως παράμετροι σώματος.
   Η απάντηση είναι η πλήρης κωδικοποίηση των δεδομένων του προϊόντος, δηλαδή ένα αρχείο JSON με τις αντίστοιχες πληροφορίες.
3. **PATCH**: Ενημερώνονται μερικώς οι πληροφορίες του προϊόντος
   Σε αντίθεση με τη μέθοδο PUT, η PATCH επιτρέπει τη μερική επεξεργασία ενός προϊόντος (Partial Update).
   Τα δεδομένα για τα στοιχεία εκείνα που θέλουμε να ενημερώσουμε περιλαμβάνονται στην αίτηση ως παράμετροι σώματος.
   Η απάντηση είναι η πλήρης κωδικοποίηση των δεδομένων του προϊόντος, δηλαδή ένα αρχείο JSON με τις αντίστοιχες πληροφορίες.
4. **DELETE**: Διαγράφεται το προϊόν
   Το προϊόν με το αντίστοιχο id αποκτά την τιμή withdrawn=true αν η αίτηση γίνεται από Εθελοντή, αλλιώς διαγράφεται από τη βάση δεδομένων αν γίνεται από Διαχειριστή.
   Το αποτέλεσμα είναι ένα αρχείο JSON της μορφής { "message": "OK" }.

Για λόγους συντομίας θα επικεντρωθούμε στα παραπάνω τελικά σημεία, αγνοώντας τα καταστήματα και τις τιμές, που έχουν όμοια λειτουργία με τα προϊόντα.

Ας δούμε πώς μπορούμε να δηλώσουμε τις παραπάνω προδιαγραφές της διεπαφής με σκοπό της παραγωγή σεναρίων ελέγχου.

Καταρχάς, η διεύθυνσή του δίνεται ως η 'https://localhost:8765/observatory/api', επομένως δηλώνουμε την θύρα του διακομιστή και τη διεύθυνση βάσης με τις εξής εντολές:

```
baseUrl '/observatory/api'
serverPort = 8765
```

Το πρώτο τελικό σημείο που έχουμε είναι αυτό της διαπίστευσης χρηστών. Παρατηρούμε πως υποστηρίζει μόνο τη μέθοδο POST, για την οποία η αίτηση είναι τύπου URL με κωδικοποιημένες τις παραμέτρους σώματος username και password. Σε περίπτωση επιτυχούς σύνδεσης, έχουμε μία απάντηση με κατάσταση με κωδικό 201 και σώμα 'Created'.

Όλα τα παραπάνω γράφονται στον Groovy Builder με τον ακόλουθο τρόπο:

```
endpoint('/login') {

   label 'login endpoint'
   description 'the endpoint for user login'
   method('POST') {
      request('URL') {
         withBodyParameter('username', 'String')
         withBodyParameter('password', 'String')
      }
      response('JSON') {
         withStatus(201) {
            body 'Created'
         }
      }
   }
}
```

Με όμοιο τρόπο περιγράφεται και το τελικό σημείο της αποσύνδεσης, το οποίο βέβαια αντί για username και password απαιτεί μία κεφαλίδα με όνομα 'X-OBSERVATORY-AUTH'.

```
endpoint('/logout') {

   label 'logout endpoint'
   description 'the endpoint for user logout'
   method('POST') {
      request('URL') {
         withHeader('X-OBSERVATORY-AUTH')
      }
      response('JSON') {
         withStatus(201) {
            body 'Created'
         }
      }
   }
}
```

Σχετικά με το τελικό σημείο των προϊόντων, θα πρέπει να το δηλώσουμε μία φορά ως endpoint χωρίς προσδιοριστικό και άλλη μία ως endpoint με id.

Για την περίπτωση χωρίς αναγνωριστικό, βλέπουμε ότι υποστηρίζονται οι μέθοδοι GET και POST. Αυτές δέχονται URL αιτήσεις και οι απαντήσεις τους είναι σε μορφή JSON.

Λαμβάνοντας υπόψη τις παραμέτρους και τις κεφαλίδες, προκύπτει ο ακόλουθος κώδικας για τον Groovy Builder:

```
endpoint('/products') {

   label 'products endpoint'
   method('GET') {
      request('URL') {
         withQueryParameter('start', 'Integer', 0)
         withQueryParameter('count', 'Integer', 20)
         withQueryParameter('status', 'String', 'ACTIVE')
         withQueryParameter('sort', 'String', 'id%7CDESC')
      }
      response('JSON') {
         withStatus(200) {
            body 'OK'
         }
      }
   }
   method('POST') {
      request('URL') {
         withHeader('X-OBSERVATORY-AUTH')
         withBodyParameter('name', 'String')
         withBodyParameter('description', 'String')
         withBodyParameter('category', 'String')
         withBodyParameter('tags', 'String')
         withBodyParameter('withdrawn', 'boolean')
      }
      response('JSON') {
         withStatus(201) {
            body 'OK'
         }
      }
   }
}
```

Στη συνέχεια έχουμε την περίπτωση ενός συγκεκριμένου προϊόντος με βάση το προσδιοριστικό id του.

Δηλώνοντας τη μεταβλητή 'id' μαζί με το μονοπάτι του τελικού σημείου και λαμβάνοντας υπόψη τις παραμέτρους και τις κεφαλίδες, προκύπτει ο ακόλουθος κώδικας για Groovy Builder:

```
endpoint('/products', 'id') {
```

```
label 'products endpoint'
method('GET') {
   request('URL') {}
   response('JSON') {
      withStatus(200) {
         body 'OK'
      }
   }
}
method('PUT') {
   request('URL') {
      withHeader('X-OBSERVATORY-AUTH')
      withBodyParameter('name', 'String')
      withBodyParameter('description', 'String')
      withBodyParameter('category', 'String')
      withBodyParameter('tags', 'String')
      withBodyParameter('withdrawn', 'boolean')
   }
   response('JSON') {
      withStatus(201) {
         body 'Created'
      }
   }
}
method('PATCH') {
   request('URL') {
      withHeader('X-OBSERVATORY-AUTH')
      withBodyParameter('name', 'String')
      withBodyParameter('description', 'String')
      withBodyParameter('category', 'String')
      withBodyParameter('tags', 'String')
      withBodyParameter('withdrawn', 'boolean')
   }
   response('JSON') {
      withStatus(201) {
         body 'Created'
      }
   }
}
method('DELETE') {
   request('URL') {
      withHeader('X-OBSERVATORY-AUTH')
   }
```

```
        response('JSON') {
            withStatus(201) {
                body 'Created'
            }
        }
    }
}
```

Εκτελώντας την εντολή *generate* στο Gradle, έχοντας πρώτα ορίσει και τις επιθυμητές τοποθεσίες, παράγονται τα εξής αρχεία:

- REST API Client

  Το πρώτο και βασικό αρχείο που παράγεται είναι ο πελάτης (client) της διεπαφής. Αυτός πέρα από βοηθητικές διαδικασίες περιλαμβάνει για κάθε τελικό σημείο συναρτήσεις που υποστηρίζουν όλες τις μεθόδους του, μαζί με παραμέτρους και κεφαλίδες εφόσον αυτές παρέχονται.

  Για παράδειγμα, η συνάρτηση που αντιστοιχεί στη μέθοδο POST του τελικού σημείου διαπίστευσης ('login') είναι η εξής:

```
public Map<String, Object> post_to_login_with_headers_and_queryParams(String
    input, Map<String, String> headers, Map<String, List<String>> queryParams) {

   if(queryParams.isEmpty()) {
      return sendRequestAndParseResponseBodyAsUTF8Text(
            () -> newPostRequest(urlPrefix + "/login", "application/x-www-form-
                urlencoded", HttpRequest.BodyPublishers.ofString(input), headers
                ),
            ClientHelper::parseJsonObject
      );
   }
   else {
      String queryParamString = queryParamsToString(queryParams);
      return sendRequestAndParseResponseBodyAsUTF8Text(
            () -> newPostRequest(urlPrefix + "/login" + queryParamString, "
                application/x-www-form-urlencoded", HttpRequest.BodyPublishers.
                ofString(input), headers),
            ClientHelper::parseJsonObject
      );
   }
}
```

  Με παρόμοιο τρόπο δημιουργούνται οι υπόλοιπες μέθοδοι, ενώ καθεμιά, ανάλογα τον τύπο της, αξιοποιεί τις παρακάτω βοηθητικές διαδικασίες:

```
private HttpRequest newPostRequest(String url, String contentType, HttpRequest.
    BodyPublisher bodyPublisher, Map<String, String> headers) {
```

```
    return newRequest("POST", url, contentType, bodyPublisher, headers);
}

private HttpRequest newGetRequest(String url, Map<String, String> headers) {

    return newRequest("GET", url, "application/x-www-form-urlencoded",
        HttpRequest.BodyPublishers.noBody(), headers);
}

private HttpRequest newPutRequest(String url, String contentType, HttpRequest.
    BodyPublisher bodyPublisher, Map<String, String> headers) {

    return newRequest("PUT", url, contentType, bodyPublisher, headers);
}

private HttpRequest newPatchRequest(String url, String contentType, HttpRequest.
    BodyPublisher bodyPublisher, Map<String, String> headers) {

    return newRequest("PATCH", url, contentType, bodyPublisher, headers);
}

private HttpRequest newDeleteRequest(String url, Map<String, String> headers) {

    return newRequest("DELETE", url, "application/x-www-form-urlencoded",
        HttpRequest.BodyPublishers.noBody(), headers);
}
```

Κάθε μέθοδος για να πραγματοποιήσει μια αίτηση χρησιμοποιεί την παρακάτω κοινή διαδικασία:

```
private HttpRequest newRequest(String method, String url, String contentType,
                        HttpRequest.BodyPublisher bodyPublisher, Map<String,
                            String> headers) {

    HttpRequest.Builder builder = HttpRequest.newBuilder();

    builder.method(method, bodyPublisher)
          .header(CONTENT_TYPE_HEADER, contentType);

    for (Map.Entry<String,String> entry : headers.entrySet())
       builder.header(entry.getKey(), entry.getValue());

    return builder
```

```
        .uri(URI.create(url))
        .build();
}
```

Τέλος, η πραγματοποίηση της HTTP αίτησης και η λήψη της απάντησης γίνονται από την ακόλουθη διαδικασία:

```
private Map<String, Object> sendRequestAndParseResponseBodyAsUTF8Text(Supplier<
    HttpRequest> requestSupplier,
                                                        Function<Reader, Map<
                                                            String, Object>>
                                                            bodyProcessor) {

   HttpRequest request = requestSupplier.get();

   Map<String, Object> bodyHeaders = new HashMap<>();

   try {
      System.out.println("Sending␣" + request.method() + "␣to␣" + request.uri())
          ;
      HttpResponse<InputStream> response = client.send(request, HttpResponse.
          BodyHandlers.ofInputStream());
      int statusCode = response.statusCode();
      if (statusCode == 200 || statusCode == 201) {
         try {
            if (bodyProcessor != null) {
               bodyHeaders.put("headers", response.headers().map());
               bodyHeaders.put("body", bodyProcessor.apply(new InputStreamReader
                   (response.body(), StandardCharsets.UTF_8)));
               return bodyHeaders;
            }
            else {
               return null;
            }
         }
         catch(Exception e) {
            throw new ResponseProcessingException(e.getMessage(), e);
         }
      }
      else {
         throw new ServerResponseException(statusCode, ClientHelper.readContents
             (response.body()));
      }
   }
```

```
        catch(IOException | InterruptedException e) {
            throw new ConnectionException(e.getMessage(), e);
        }
    }
```

Η παραπάνω διαδικασία υποστηρίζει αιτήσεις που έχουν μορφή JSON. Με όμοιο τρόπο παράγονται για String και για Integer.

- Σενάρια ελέγχου για τον παραγόμενο κώδικα

  Μαζί με τον παραπάνω REST API Client, παράγονται και τα κατάλληλα σενάρια ελέγχου για αυτόν σε γλώσσα Spock.

  Για παράδειγμα, για το τελικό σημείο διαπίστευσης και τη μέθοδο POST που υποστηρίζει, δημιουργείται το παρακάτω σενάριο ελέγχου:

```
def "POST to login with headers and queryParams"() {
    given:
    String requestBody = "username=bodyParamValue&password=bodyParamValue"
    ObjectMapper responseMapper = new ObjectMapper()
    JsonNode resultJSON = responseMapper.readTree("{\"value\":\"ok\"}")
    wms.givenThat(
          post(urlMatching("/observatory/api/login\\\\?.*"))
          .withRequestBody(containing(requestBody))
          .willReturn(aResponse()
                .withStatus(201)
                .withJsonBody(resultJSON)
          )
    )
    Map<String, String> headers = new HashMap<>()
    Map<String, List<String>> queryParams = new HashMap<>()

    when:
    Map<String, Object> result = caller.
        post_to_login_with_headers_and_queryParams(requestBody, headers,
        queryParams)

    then:
    result.get("body").toString().matches("[\\{\\[].*[\\}\\]]")
}
```

Βλέπουμε ότι προσδιορίζεται ένας εικονικός διακομιστής (Mock Server), που λαμβάνει τις αιτήσεις του REST API Client με βάση μία κανονική έκφραση (regular expression) που ελέγχει τη διεύθυνση.

Παράλληλα δημιουργούνται δοκιμαστικές αιτήσεις με βάση τις ιδιότητες της μεθόδου και του τελικού σημείου. Στο παράδειγμά μας οι παράμετροι σώματος 'username' και

'password' παίρνουν τυχαίες τιμές και αποκτούν την κωδικοποιημένη μορφή URL.

Ο εικονικός διακομιστής αναμένει αίτηση που περιλαμβάνει το περιεχόμενο που στέλνουμε και επιστρέφει μία απάντηση με κωδικό κατάστασης 201 και ένα τυχαίο αρχείο JSON.

Ο τελευταίος έλεγχος αφορά την απάντηση που δέχεται ο REST API Client από τον εικονικό διακομιστή και μέσω κανονικής έκφρασης ελέγχει αν είναι μορφής JSON.

- Σενάρια ελέγχου για το RESTful API

  Το τελευταίο αρχείο που παράγεται περιλαμβάνει σενάρια ελέγχου για τον 'RESTful API', αυτή τη φορά θεωρώντας πως έχουμε πραγματικό διακομιστή αντί για εικονικό.

```
def "POST to login with headers and queryParams"() {

    given:
    String requestBody = "username=bodyParamValue&password=bodyParamValue"
    Map<String, String> headers = new HashMap<>()
    Map<String, List<Object>> queryParams = new HashMap<>()

    when:
    Map<String, Object> result = caller.
        post_to_login_with_headers_and_queryParams(requestBody, headers,
        queryParams)
    Map<String, Object> resultHeaders = result.get("headers") as Map<String,
        Object>

    then:
    println("Body:")
    println(result.get("body"))
    println("Headers:")
    println(resultHeaders)
    result.toString().matches("[\\{\\[].*[\\}\\]]")
}
```

Όπως βλέπουμε, παρότι λείπει ο εικονικός διακομιστής, δημιουργούμε μία αίτηση με βάση τις παραμέτρους και τις κεφαλίδες που υποστηρίζονται.

Στα σενάρια ελέγχου για το RESTful API έχουμε επιπλέον την εκτύπωση του σώματος και των κεφαλίδων της απάντησης. Αυτό είναι χρήσιμες σε περιπτώσεις όπως αυτή του τελικού σημείου διαπίστευσης, μιας και ο χρήστης μπορεί εύκολα να αντικαταστήσει τις προκαθορισμένες τιμές με ένα λειτουργικό ζεύγος username και password και να πάρει το authentication token είτε αυτό επιστρέφεται μέσω του σώματος της απάντησης, είτε μέσω κάποιας κεφαλίδας της.

Εκτελώντας το παραπάνω σενάριο ελέγχου, έχουμε την ακόλουθη έξοδο στην κονσόλα:

```
Sending POST to https://localhost:8765/observatory/api/login
```

```
Body:
[token:$2a$10$hwOh6PJZeV1Nl9HKmOwsdO7RLWPxwEgojuYINKvDWwAYOkWvK11Uu]
Headers:
[content-length:[75], content-type:[application/json], date:[Mon, 11 Jan 2021
    10:13:41 GMT], vary:[Origin, Access-Control-Request-Method, Access-Control-
    Request-Headers]]
```

Επιπλέον παράγονται έλεγχοι για κάθε παράμετρο και κεφαλίδα που είναι υποχρεωτικές. Στην περίπτωσή μας και οι δύο παράμετροι σώματος της αίτησης (username και password) είναι υποχρεωτικές. Επομένως το ακόλουθο σενάριο ελέγχου πραγματοποιεί μία αίτηση χωρίς username και θεωρείται έγκυρο αν ο διακομιστής εγείρει εξαίρεση.

```
def "POST to login without mandatory bodyParam: username"() {

   given:
   String requestBody = "password=bodyParamValue"
   Map<String, String> headers = new HashMap<>()
   Map<String, List<Object>> queryParams = new HashMap<>()

   when:
   caller.post_to_login_with_headers_and_queryParams(requestBody, headers,
       queryParams)

   then:
   thrown RuntimeException
}
```

5.2 Ψηφιακό μητρώο δικτυακών υποδομών

Η δεύτερη προγραμματιστική διεπαφή τύπου REST που χρησιμοποιήθηκε για την επαλήθευση της εργασίας είναι τμήμα ενός ψηφιακού μητρώου για της δικτυακές υποδομές μιας χώρας. Σκοπός του είναι η καταγραφή και η παρακολούθηση της κατάστασης του συνόλου των διαθέσιμων δικτυακών υποδομών της χώρας.

Πιο συγκεκριμένα, η διεπαφή έχει τα εξής χαρακτηριστικά:

- **Διεύθυνση**

  Ως διεύθυνση βάσης χρησιμοποιείται το 'https://localhost:44349/api'.

  Οι πόροι του συστήματος δίνονται από τα τελικά σημεία στη μορφή {baseURL}/{path-to-resource}.

  Έτσι το τελικό σημείο των παρόχων είναι το {baseURL}/NationalNetwork, ενώ για τον πάροχο με κωδικό 3 το {baseURL}/NationalNetwork/3.

- **Διαπίστευση**

  Το τελικό σημείο διαπίστευσης χρηστών έχει το μονοπάτι '{baseURL}/login'.

  Η μοναδική μέθοδος που υποστηρίζει είναι η POST.

  Υπάρχουν δύο ειδών χρήστες στο σύστημα, οι Διαχειριστές που έχουν δικαίωμα επεξεργασίας όλων των δεδομένων και οι Πάροχοι που έχουν δικαίωμα επεξεργασίας μόνο των πληροφοριών των συνδέσεων.

  Το τελικό σημείο της διαπίστευσης δέχεται ένα αρχείο JSON με το username και το password. Σε περίπτωση επιτυχούς σύνδεσης επιστρέφει αρχείο JSON με το id του χρήστη, το username του, τον ρόλο του και ένα Bearer token. Αυτό θα πρέπει να περιλαμβάνεται ως κεφαλίδα ταυτοποίησης με όνομα 'Authorization' σε όποια αίτηση απαιτείται πιστοποίηση.

  Η διάρκεια του JSON Web Token είναι 7 ημέρες.

- **Πάροχοι**

  Το τελικό σημείο των παρόχων έχει το μονοπάτι '{baseURL}/NationalNetwork'.

  Σε αυτό υποστηρίζονται οι ακόλουθες μέθοδοι:

  1. **GET**: Επιστρέφεται η λίστα των παρόχων

     Δικαίωμα προβολής των πληροφοριών των παρόχων έχει οποιοσδήποτε χρήστης, χωρίς ανάγκη πιστοποίησης.

     Η απάντηση είναι ένα αρχείο JSON που περιλαμβάνει μία λίστα με τα στοιχεία των παρόχων, της οποίας κάθε καταχώρηση περιλαμβάνει τις εξής παραμέτρους:

     - id: Integer
     - name: String
     - location: String
     - created: String
     - email: String

  2. **POST**: Δημιουργείται νέα καταχώρηση παρόχου

     Οι πληροφορίες που απαιτούνται για τη δημιουργία ενός παρόχου διατυπώνονται ως παράμετροι σώματος στην αίτηση. Αυτές είναι το name, το location και το email.

     Η απάντηση στην αίτηση δημιουργίας ενός προϊόντος είναι η πλήρης κωδικοποίηση των δεδομένων του, δηλαδή ένα αρχείο JSON με τις αντίστοιχες πληροφορίες.

  Επιπλέον υποστηρίζει προσδιοριστικό (attribute) για το id των παρόχων, με μονοπάτι της μορφής '{baseURL}/NationalNetwork/{id}'.

  Με το id υποστηρίζονται οι ακόλουθες μέθοδοι:

  1. **GET**: Επιστρέφονται οι πληροφορίες του παρόχου

     Για παράδειγμα, GET *{baseURL}/products/2*.

Το αποτέλεσμα της αίτησης είναι ένα αρχείο JSON με τα δεδομένα του παρόχου με το αντίστοιχο id.

Αυτά όπως και στην περίπτωση της μεθόδου GET χωρίς προσδιοριστικό id είναι τα ακόλουθα:

- id: Integer
- name: String
- location: String
- created: String
- email: String

2. **PATCH**: Ενημερώνονται μερικώς οι πληροφορίες του παρόχου

   Η μέθοδος PATCH επιτρέπει τη μερική επεξεργασία ενός παρόχου (Partial Update).

   Τα δεδομένα για τα στοιχεία εκείνα που θέλουμε να ενημερώσουμε περιλαμβάνονται στην αίτηση ως παράμετροι σώματος.

   Η απάντηση είναι η πλήρης κωδικοποίηση των δεδομένων του παρόχου, δηλαδή ένα αρχείο JSON με τις αντίστοιχες πληροφορίες.

3. **DELETE**: Διαγράφεται ο πάροχος

   Ο πάροχος με το αντίστοιχο id διαγράφεται από τη βάση δεδομένων, εφόσον η αίτηση γίνεται από χρήστη με ρόλο Διαχειριστή.

   Το αποτέλεσμα είναι ένα αρχείο String της μορφής "success at deletion".

Για λόγους συντομίας θα ασχοληθούμε μόνο με το τελικό σημείο των παρόχων. Όμοια με αυτό λειτουργούν τα τελικά σημεία των συνδέσεων και των χρηστών, ενώ η διαπίστευση δε διαφέρει σημαντικά από αυτήν του παρατηρητηρίου τιμών του πρώτου παραδείγματος.

Η διεύθυνση της διεπαφής δίνεται ως η 'https://localhost:44349/api', επομένως δηλώνουμε την θύρα του διακομιστή και τη διεύθυνση βάσης με τις εξής εντολές:

```
baseUrl '/api'
serverPort = 44349
```

Όπως και πριν, έτσι και τώρα θα δηλώσουμε μία φορά το τελικό σημείο χωρίς προσδιοριστικό και μία με το id.

Μία βασική διαφορά των δύο διεπαφών είναι ότι το ψηφιακό μητρώο δικτυακών υποδομών δέχεται αιτήσεις με τύπο περιεχομένου JSON αντί για URL. Αυτό όμως δεν είναι πρόβλημα ούτε απαιτεί κάποια σημαντική αλλαγή πέρα από τη δήλωση 'JSON' στην αίτηση κάθε μεθόδου.

Έτσι, για παράδειγμα, για να περιγράψουμε στον Groovy Builder το τελικό σημείο των παρόχων με και χωρίς προσδιοριστικό, γράφουμε τα εξής:

```
endpoint('/NationalNetwork') {

   label 'providers networks'
   description 'an endpoint for providers'
```

```
   method('GET') {
      request('JSON') {}
      response('JSON') {
         withStatus(200) {
            body 'OK'
         }
      }
   }
   method('POST') {
      request('JSON') {
         withHeader('Authorization')
         withBodyParameter('name', 'String')
         withBodyParameter('location', 'String')
         withBodyParameter('email', 'String')
      }
      response('JSON') {
            withStatus(201) {
               body 'Created'
            }
      }
   }
}

endpoint('/NationalNetwork', 'id') {

   label 'providers networks with id'
   description 'an endpoint for providers by id'
   method('PATCH') {
      request('JSON') {
         withHeader('Authorization')
         withBodyParameter('name', 'String')
         withBodyParameter('location', 'String')
         withBodyParameter('email', 'String')
      }
      response('JSON') {
         withStatus(200) {
            body 'OK'
         }
      }
   }
   method('DELETE') {
      request('JSON') {
         withHeader('Authorization')
         withBodyParameter('name', 'String')
```

```
        withBodyParameter('location', 'String')
        withBodyParameter('email', 'String')
    }
    response('String') {
        withStatus(201) {
            body 'Created'
        }
    }
  }
}
```

Με την εκτέλεση της εντολής *generate* στον Gradle, πάραγονται όπως και πριν τα παρακάτω τρία αρχεία:

- REST API Client

  Βλέπουμε ότι στον πελάτη που δημιουργείται, για κάθε μέθοδο της διεπαφής που δέχεται αιτήσεις τύπου JSON, αυτόματα αλλάζει το όρισμα του τύπου δεδομένων σε 'application/json'.

```
public Map<String, Object>
    post_to_NationalNetwork_with_headers_and_queryParams(String input, Map<
    String, String> headers, Map<String, List<String>> queryParams) {

    if(queryParams.isEmpty()) {
        return sendRequestAndParseResponseBodyAsUTF8Text(
              () -> newPostRequest(urlPrefix + "/NationalNetwork", "application
                  /json", HttpRequest.BodyPublishers.ofString(input), headers),
              ClientHelper::parseJsonObject
        );
    }
    else {
    String queryParamString = queryParamsToString(queryParams);

        return sendRequestAndParseResponseBodyAsUTF8Text(
              () -> newPostRequest(urlPrefix + "/NationalNetwork" +
                  queryParamString, "application/json", HttpRequest.
                  BodyPublishers.ofString(input), headers),
              ClientHelper::parseJsonObject
        );
    }
}
```

- Σενάρια ελέγχου για τον παραγόμενο κώδικα

  Στα σενάρια ελέγχου που παράγονται και αφορούν τον κώδικα του REST API Client αυτή τη φορά έχουμε τελικό σημείο διαπίστευσης που δέχεται τα δεδομένα σε μορφή

JSON. Δηλώνοντάς το αυτό στον Groovy Builder, το αντίστοιχο σενάριο ελέγχου είναι το εξής:

```
def "POST to users_authenticate with headers and queryParams"() {

   given:
   String requestBody = new JSONObject()
      .put("username", "bodyParamValue")
      .put("password", "bodyParamValue")
      .toString();
   ObjectMapper responseMapper = new ObjectMapper()
   JsonNode resultJSON = responseMapper.readTree("{\"value\":\"ok\"}")
   wms.givenThat(
         post(urlMatching("/api/users/authenticate\\\\?.*"))
         .withRequestBody(containing(requestBody))
         .willReturn(aResponse()
               .withStatus(201)
               .withJsonBody(resultJSON)
         )
   )
   Map<String, String> headers = new HashMap<>()
   Map<String, List<String>> queryParams = new HashMap<>()

   when:
   Map<String, Object> result = caller.
       post_to_users_authenticate_with_headers_and_queryParams(requestBody,
       headers, queryParams)

   then:
   result.get("body").toString().matches("[\\{\\[].*[\\}\\]]")
}
```

- Σενάρια ελέγχου για το RESTful API

  Όμοια με παραπάνω, το σενάριο ελέγχου για την διαπίστευση των χρηστών προσαρμόζεται στα δεδομένα εισόδου και γίνεται το ακόλουθο:

```
def "POST to users_authenticate with headers and queryParams"() {

   given:
   String requestBody = new JSONObject()
      .put("username", "bodyParamValue")
      .put("password", "bodyParamValue")
      .toString();
   Map<String, String> headers = new HashMap<>()
```

```
    Map<String, List<Object>> queryParams = new HashMap<>()

    when:
    Map<String, Object> result = caller.
        post_to_users_authenticate_with_headers_and_queryParams(requestBody,
        headers, queryParams)
    Map<String, Object> resultHeaders = result.get("headers") as Map<String,
        Object>

    then:
    println("Body:")
    println(result.get("body"))
    println("Headers:")
    println(resultHeaders)

    result.toString().matches("[\\{\\[].*[\\}\\]]")
}
```

Αυτή τη φορά μπορούμε να τροποποιήσουμε το σενάριο ελέγχου αλλάζοντας τις τυχαίες τιμές των παραμέτρων username και password στο JSON αρχείο αίτησης με ένα ζεύγος έγκυρων δεδομένων.

Το authentication token που θα τυπωθεί στην κονσόλα μπορούμε να το αντιγράψουμε στην τιμή της αντίστοιχης κεφαλίδας πιστοποίησης των σεναρίων ελέγχου που την απαιτούν.

Έτσι, για παράδειγμα, μπορούμε να αντικαταστήσουμε με αυτό το 'headerValue' στο σενάριο ελέγχου της δημιουργίας νέου παρόχου μέσω μεθόθου POST, το οποίο παράγεται αυτόματα ως εξής:

```
def "POST to NationalNetwork with headers and queryParams"() {

    given:
    String requestBody = new JSONObject()
        .put("name", "bodyParamValue")
        .put("location", "bodyParamValue")
        .put("email", "bodyParamValue")
        .toString();
    Map<String, String> headers = new HashMap<>()
    headers.put("Authorization", 'headerValue')
    Map<String, List<Object>> queryParams = new HashMap<>()

    when:
    Map<String, Object> result = caller.
        post_to_NationalNetwork_with_headers_and_queryParams(requestBody, headers
```

```
        , queryParams)
    Map<String, Object> resultHeaders = result.get("headers") as Map<String,
        Object>

    then:
    println("Body:")
    println(result.get("body"))
    println("Headers:")
    println(resultHeaders)

    result.toString().matches("[\\{\\[].*[\\}\\]]")
}
```

Κεφάλαιο 6

Σχετικές Εργασίες

Η μελέτη των προγραμματιστικών διεπαφών τύπου REST και του ελέγχου τους, παρότι είναι ένας τομέας που αξιοποιείται σημαντικά σε επιχειρησιακό επίπεδο, παρουσιάζει και μεγάλο ερευνητικό ενδιαφέρον σήμερα.

Οι προγραμματιστικές διεπαφές τύπου REST μελετώνται παράλληλα με τις σύγχρονες τεχνολογικές τάσεις, όπως είναι για παράδειγμα η ανάπτυξη της νέας γενιάς ασύρματων δικτύων 5G [29]. Η νέα γενιά αυτή αναμένεται να επιφέρει ραγδαίες εξελίξεις ως προς τη μετάβαση στο διαδίκτυο των πραγμάτων (IoT) [30], μία κατάσταση στην οποία μηχανές και συσκευές καθημερινής χρήσης αλληλεπιδρούν μεταξύ τους μέσω ενος διεθνούς δικτύου [31].

Με την έκρηξη της μηχανικής μάθησης και των νευρωνικών δικτύων, έχουν υπάρξει προσπάθειες αξιοποίησης της τεχνητής νοημοσύνης στον κλάδο των προγραμματιστικών διεπαφών. Αυτές περιλαμβάνουν τόσο τη σχεδίαση νέων μορφών διεπαφών, προσαρμοσμένων στις απαιτήσεις της μηχανικής μάθησης [32][33] και την σύγκριση των δημοφιλέστερων [34], όσο και τον έλεγχο των διεπαφών με την αυτοματοποιημένη παραγωγή μεγάλου όγκου σεναρίων ελέγχου που προκύπτουν από την ανάλυση χιλιάδων άλλων προγραμμάτων [35].

Στο παρόν κεφάλαιο περιγράφονται τα παρακάτω πιο δημοφιλή εργαλεία που χρησιμοποιούνται σήμερα για την μοντελοποίηση και τον έλεγχο προγραμματιστικών διεπαφών τύπου REST:

- OpenAPI Specification [36]
- Swagger Inspector [37]
- Postman [38]

6.1 OpenAPI Specification

Το OpenAPI Specification είναι μία προδιαγραφή μοντελοποίησης για RESTful διεπαφές προγραμματισμού που χρησιμοποιείται για την περιγραφή, την ανάπτυξη, τη χρήση και την αναπαράστασή τους.

Η προδιαγραφή OpenAPI είναι ανεξάρτητη από τη γλώσσα προγραμματισμού και επιτρέπει τόσο στους προγραμματιστές να περιγράψουν εύκολα και κατανοητά μια διεπαφή, όσο και σε εφαρμογές να αλληλεπιδράσουν μαζί της.

Η μορφή που έχει είναι ένα αρχείο JSON που περιλαμβάνει τα υποστηριζόμενα πεδία με τις τιμές τους. Οι τιμές αυτές μπορεί να δηλώνουν είτε μία σταθερά είτε ένα σχήμα δεδομένων.

Οι τύποι δεδομένων που υποστηρίζονται είναι οι εξής:

- Αριθμοί (integer, long, float, double)
- Συμβολοσειρές (συμπεριλαμβανομένων bytes και bits)
- Τύποι δεδομένων Αληθείας (Boolean)
- Ειδικοί τύποι δεδομένων, όπως ημερομηνίες και κωδικοί πρόσβασης

Το αρχείο JSON της προδιαγραφής μπορεί να περιλαμβάνει τα ακόλουθα πεδία:

- openapi

  Υποχρεωτικό πεδίο, περιλαμβάνει την έκδοση της προδιαγραφής OpenAPI που χρησιμοποιείται.

- info

  Υποχρεωτικό πεδίο, περιλαμβάνει απαραίτητες πληροφορίες για την διεπαφή.

- servers

  Περιλαμβάνει τους διακομιστές με τους οποίους επικοινωνεί η διεπαφή.

- paths

  Υποχρεωτικό πεδίο, περιλαμβάνει τα μονοπάτια των τελικών σημείων και των λειτουργιών τους.

- components

  Περιλαμβάνει όλα τα δομικά στοιχεία της διεπαφής.

- security

  Περιλαμβάνει περιγραφές των μηχανισμών ασφαλείας της διεπαφής.

- tags

  Περιλαμβάνει ετικέτες με μεταδεδομένα (metadata) για την διεπαφή.

- externalDocs

  Περιλαμβάνει εξωτερικά έγγραφα τεκμηρίωσης.

Το κομμάτι που παρουσιάζει ενδιαφέρον για την παρούσα εργασία είναι εκείνο των δομικών στοιχείων (components). Όπως φαίνεται, χρησιμοποιώντας την προδιαγραφή OpenAPI, μπορεί κανείς να ορίσει μία προγραμματιστική διεπαφή τύπου REST ως ένα σύνολο από τα ακόλουθα συστατικά:

- schemas

  Περιγραφές για τα δεδομένα εισόδου ή εξόδου. Αυτές μπορεί να είναι μία κανονική έκφραση (regex) του περιεχομένου, μέγιστο και ελάχιστο μήκος ή πλήθος αντικειμένων κλπ.

- responses

  Λίστα με τις αναμενόμενες απαντήσεις με βάση την κωδικό κατάστασης (HTTP status code).

  Κάθε απάντηση περιλαμβάνει υποχρεωτικά περιγραφική (description), ενώ προαιρετικά πληροφορίες για τις κεφαλίδες και το περιεχόμενό της, όπως το σχήμα των δεδομένων.

- parameters

  Η προδιαγραφή OpenAPI ορίζει τέσσερις τύπους παραμέτρων:

  1. path - μονοπατιού διεύθυνσης
  2. query - επιθέματος διεύθυνσης
  3. header - κεφαλίδας
  4. cookie - δεδομένων cookies

- examples

  Παραδείγματα επαλήθευσης με τιμές που στηρίζονται στο σχήμα δεδομένων του αντικειμένου.

- requestBodies

  Αναπαράσταση των πιθανών σωμάτων μιας αίτησης, με τις περιγραφές τους και πληροφορίες για το αν είναι προαιρετικές ή υποχρεωτικές και τι περιεχόμενο υποστηρίζουν.

- headers

  Αναπαράσταση μιας κεφαλίδας, με δομή παρόμοια με αυτή των παραμέτρων.

- securitySchemes

  Πληροφορίες για τις δυνατότητες ασφαλείας, με υποστηριζόμενες τις εξής:

  1. Πιστοποίηση HTTP
  2. Κλειδί API (μέσω header, query ή cookie)
  3. OAuth2
  4. OpenID

- links

  Διευθύνσεις που αναπαριστούν πιθανές αιτήσεις της διεπαφής, με πληροφορίες για τις παραμέτρους τους, το σώμα τους και τον διακομιστή που επικοινωνούν.

- callbacks

  Αναπαράσταση των πιθανών απαντήσεων της διεπαφής, με βάση τα χαρακτηριστικά των αιτήσεων.

Σε αντιπαραβολή με την παρούσα εργασία, παρατηρούμε πως η μοντελοποίηση RESTful API που σχεδιάστηκε υποστηρίζει περιορισμένους τύπους δεδομένων σε σχέση με το OpenAPI. Για το ίδιο το RESTful API βλέπουμε πως εδώ υποστηρίζονται περισσότερα χαρακτηριστικά, όπως πολλαπλοί διακομιστές και επιλογές ασφαλείας. Όσο για τα δομικά στοιχεία της διεπαφής, αξίζει να σημειωθούν η επιλογή για cookies, η πληθώρα μεθόδων και τύπων περιεχομένου (content-type) των αιτήσεων και η μεγαλύτερη ποικιλία σε τύπους μεταβλητών, όπως για παράδειγμα ημερομηνίες και κωδικοί πρόσβασης.

6.2 Swagger Inspector

Το Swagger είναι ένα ανοικτό πλαίσιο εργαλείων που βοηθούν στον σχεδιασμό, την ανάπτυξη, την τεκμηρίωση και την χρήστη προγραμματιστικών διεπαφών τύπου REST.

Ένα από τα εργαλεία αυτά είναι το Swagger Inspector που επιτρέπει τον έλεγχο διεπαφών. Ο έλεγχος πραγματοποιείται μέσα από την αλληλεπίδραση με τα τελικά σημεία μιας διεπαφής.

Το Swagger Inspector επιτρέπει την αποστολή αιτημάτων σε ένα τελικό σημείο χρησιμοποιώντας κάποιες βασικές HTTP μεθόδους. Τα αιτήματα αυτά μπορεί να περιλαμβάνουν παραμέτρους, κεφαλίδες και σώμα. Ακόμα υποστηρίζεται η διαπίστευση μέσω username και password σε authentication header και μέσω JSON Web Token.

Με την πραγματοποίηση της αίτησης, εμφανίζεται η απάντηση που επιστρέφεται, μαζί με τον κωδικό κατάστασης, τον χρόνο που χρειάστηκε και τις κεφαλίδες που περιλαμβάνει.

Τέλος, υποστηρίζεται η εισαγωγή προδιαγραφής OpenAPI. Αυτόματα δημιουργείται μια λίστα με τα υποστηριζόμενα τελικά σημεία και για καθένα από αυτά προκαθορισμένες αιτήσεις με τις αντίστοιχες μεθόδους. Σε κάθε αίτηση περιλαμβάνονται οι παράμετροι που περιγράφονται στην προδιαγραφή με τιμές που μπορεί να συμπληρώσει ο χρήστης.

Βλέπουμε πως το Swagger Inspector δεν παράγει αυτόματα σενάρια ελέγχου, όμως σε αντίθεση με την παρούσα εργασία, για κάθε χειροκίνητο σενάριο ελέγχου που εκτελείται, ενημερώνει τον χρήστη για τον χρόνο που χρειάστηκε.

6.3 Postman

Ένα άλλο πολύ δημοφιλές εργαλείο ελέγχου ΑΠΙς είναι το Postman με 13 εκατομμύρια ενεργούς χρήστες και 500 χιλιάδες συνεργαζόμενες εταιρείες.

Πέρα από την πραγματοποίηση αιτημάτων σε τελικά σημεία και τη λήψη απαντήσεων, έχει επιπλέον τη δυνατότητα συγγραφής και εκτέλεσης σεναρίων ελέγχου. Αυτά έχουν τη μορφή τμημάτων κώδικα JavaScript και χρησιμοποιούν δυναμικές μεταβλητές για την αξιολόγηση των περιεχομένων της απάντησης και την ανταλλαγή δεδομένων μεταξύ αιτήσεων.

Για παράδειγμα, ένας έλεγχος που είναι έγκυρος όταν η απάντηση έχει κωδικό κατάστασης 201 γράφεται σε JavaScript ως εξής:

```
pm.test("Status test", function () {
   pm.response.to.have.status(201);
});
```

Παρόμοια θα γράφαμε ένα σενάριο ελέγχου για το περιεχόμενο της απάντησης. Αν θέλαμε να ελέγξουμε ότι η απάντηση είναι ένα αρχείο JSON και δεν δηλώνει κάποιο πρόβλημα, θα συντάσσαμε τον ακόλουθο κώδικα:

```
pm.test("response should be okay to process", function () {
   pm.response.to.not.be.error;
   pm.response.to.have.jsonBody("");
   pm.response.to.not.have.jsonBody("error");
});
```

Το Postman τέλος υποστηρίζει τη δημιουργία συλλογών και φακέλων με πλήθος σεναρίων ελέγχου. Αυτά μπορεί να εκτελούνται είτε μετά από κάθε αίτηση, είτε μετά από ένα σύνολο αιτήσεων, ενώ μετά την εκτέλεσή τους παρουσιάζονται συνοπτικά πληροφορίες για κάθε αίτηση και κάθε σενάριο ελέγχου.

Κεφάλαιο 7

Συμπεράσματα Και Μελλοντικές Επεκτάσεις

Βασική επιδίωξη της εργασίας είναι η αυτόματη παραγωγή σεναρίων ελέγχου για προγραμματιστικές διεπαφές τύπου REST. Με τη δηλωτική γλώσσα ορισμού RESTful API μοντέλων που σχεδιάστηκε μπορεί οποιοσδήποτε να περιγράψει μία διεπαφή προγραμματισμού. Ο Κατασκευαστής Groovy διαβάζει τη γλώσσα αυτή και παράγει Java κλάσεις που αναπαριστούν τη διεπαφή. Έπειτα η Γεννήτρια Κώδικα δέχεται στην είσοδό της τις παραπάνω κλάσεις και δημιουργεί σενάρια ελέγχου για το αρχικό RESTful API μοντέλο, μαζί με τον απαραίτητο Πελάτη (Client) και σενάρια ελέγχου για τον παραγόμενο κώδικα. Όπως φαίνεται από τον σχεδιασμό και την υλοποίηση της εργασίας , καθώς και της επαλήθευσης της ορθής λειτουργίας της μέσω δύο διαδικτυακών RESTful APIs, με τη δηλωτική περιγραφή μιας διεπαφής προγραμματισμού μπορούμε εύκολα να παράγουμε σενάρια ελέγχου που καλύπτουν ένα μεγάλο φάσμα των δυνατοτήτων της και των χαρακτηριστικών της.

Στο πλαίσιο της εργασίας αποφασίστηκαν κάποιες παραδοχές ως προς την εμβάθυνση που θα γινόταν στο σύνολο των δεδομένων. Ως εκ τούτου, υπάρχει περιθώριο μελλοντικών επεκτάσεων για την παραγωγή αναλυτικότερων και πιο συγκεκριμένων σεναρίων ελέγχου. Για παράδειγμα, θα μπορούσε να υπάρξει υποστήριξη σε περισσότερους τύπους περιεχομένου (content-type) των αιτήσεων, περισσότερων HTTP μεθόδων και μεγαλύτερη ποικιλία τύπων μεταβλητών. Επιπλέον, χρήσιμη θα ήταν μία εξειδίκευση σε τελικά σημεία, όπως για παράδειγμα αυτά της διαπίστευσης και αποσύνδεσης χρηστών, που θα επέτρεπε πιο στοχευμένα σενάρια ελέγχου. Τέλος, μία δυνατότητα επέκτασης της εργασίας αφορά την παροχή περισσότερων ειδών ελέγχου των προγραμματιστικών διεπαφών, όπως για παράδειγμα έλεγχος της απόδοσής της.

Βιβλιογραφία

[1] *Information Management: A Proposal*. https://www.w3.org/History/1989/proposal.html. Ημερομηνία πρόσβασης: 29-11-2020.

[2] C. Date και E. F. Codd. *The relational and network approaches: Comparison of the application programming interfaces*. *SIGFIDET '74*, 1975.

[3] Glenford J Myers. *A controlled experiment in program testing and code walkthroughs/inspections*. *Communications of the ACM*, 21(9):760–768, 1978.

[4] Brian A Wichmann, AA Canning, DL Clutterbuck, LA Winsborrow, NJ Ward και DWR Marsh. *Industrial perspective on static analysis*. *Software Engineering Journal*, 10(2):69–75, 1995.

[5] Thomas Ball. *The concept of dynamic analysis*. *Software Engineering—ESEC/FSE'99*, σελίδες 216–234. Springer, 1999.

[6] Glenford J Myers, Tom Badgett, Todd M Thomas και Corey Sandler. *The art of software testing*, τόμος 2. Wiley Online Library, 2004.

[7] Jussi Kasurinen, Ossi Taipale και Kari Smolander. *Software Test Automation in Practice: Empirical Observations*. *Advances in Software Engineering*, 2010, 2010.

[8] Tim A. Majchrzak. *Improving the Technical Aspects of Software Testing in Enterprises*. *International Journal of Advanced Computer Science and Applications*, 1, 2010.

[9] Jiantao Pan. *Software testing*. *Dependable Embedded Systems*, 5:2006, 1999.

[10] Connie U Smith. *Performance engineering of software systems*. Addison-Wesley Longman Publishing Co., Inc., 1990.

[11] Filippos I Vokolos και Elaine J Weyuker. *Performance testing of software systems*. *Proceedings of the 1st International Workshop on Software and Performance*, σελίδες 80–87, 1998.

[12] Dick Hamlet. *Foundations of software testing: dependability theory*. *Proceedings of the 2nd ACM SIGSOFT Symposium on Foundations of software engineering*, σελίδες 128–139, 1994.

[13] S Pradeep και Yogesh Kumar Sharma. *A pragmatic evaluation of stress and performance testing technologies for web based applications*. *2019 Amity International Conference on Artificial Intelligence (AICAI)*, σελίδες 399–403. IEEE, 2019.

[14] Bruce Potter και Gary McGraw. *Software security testing*. *IEEE Security & Privacy*, 2(5):81–85, 2004.

[15] Rasneet Kaur Chauhan και Iqbal Singh. *Latest research and development on software testing techniques and tools*. *International Journal of Current Engineering and Technology*, 4(4):2368–2372, 2014.

[16] Elfriede Dustin, Jeff Rashka και John Paul. *Automated Software Testing: Introduction, Management, and Performance: Introduction, Management, and Performance*. Addison-Wesley Professional, 1999.

[17] Antonia Bertolino. *Software testing research: Achievements, challenges, dreams*. *Future of Software Engineering (FOSE'07)*, σελίδες 85–103. IEEE, 2007.

[18] Karuturi Sneha και Gowda M Malle. *Research on software testing techniques and software automation testing tools*. *2017 International Conference on Energy, Communication, Data Analytics and Soft Computing (ICECDS)*, σελίδες 77–81. IEEE, 2017.

[19] Roy Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. Διδακτορική Διατριβή, 2000.

[20] Brenda Jin, Saurabh Sahni και Amir Shevat. *Designing Web APIs: Building APIs That Developers Love*. O'Reilly Media, Incorporated, 2018.

[21] Michael Amundsen, Leonard Richardson και S Ruby. *RESTful Web APIs*. O'Reilly Media, Incorporated, 2013.

[22] Mark Massé. *REST API design rulebook: designing consistent RESTful Web Service Interfaces*. O'Reilly Media, Incorporated, Beijing, 2012.

[23] Roy Fielding, Jim Gettys, Jeffrey Mogul, Henrik Frystyk, Larry Masinter, Paul Leach και Tim Berners-Lee. *Hypertext transfer protocol-HTTP/1.1*, 1999.

[24] Patrick Niemeyer και Jonathan Knudsen. *Learning java*. " O'Reilly Media, Inc.", 2005.

[25] James Gosling και Henry McGilton. *The Java language environment*. *Sun Microsystems Computer Company*, 2550:38, 1995.

[26] Ken Arnold, James Gosling, David Holmes και David Holmes. *The Java programming language*, τόμος 2. Addison-wesley Reading, 2000.

[27] Erich Gamma. *Design patterns: elements of reusable object-oriented software*. Pearson Education India, 1995.

[28] Konstantinos Kapelonis. *Java testing with Spock*. Manning Publications Shelter Island, New York, 2016.

[29] Georg Mayer. *RESTful APIs for the 5G service based architecture*. *Journal of ICT Standardization*, 6(1):101–116, 2018.

[30] Dan Wang, Dong Chen, Bin Song, Nadra Guizani, Xiaoyan Yu και Xiaojiang Du. *From IoT to 5G I-IoT: The next generation IoT-based intelligent algorithms and 5G technologies*. *IEEE Communications Magazine*, 56(10):114–120, 2018.

[31] Jayavardhana Gubbi, Rajkumar Buyya, Slaven Marusic και Marimuthu Palaniswami. *Internet of Things (IoT): A vision, architectural elements, and future directions*. *Future generation computer systems*, 29(7):1645–1660, 2013.

[32] Álvaro López García. *DEEPaaS API: A REST API for machine learning and deep learning models*. *Journal of Open Source Software*, 4(42):1517, 2019.

[33] Jeremy Howard και Sylvain Gugger. *Fastai: A layered API for deep learning*. *Information*, 11(2):108, 2020.

[34] Adam Kubany, Shimon Ben Ishay, Ruben Sacha Ohayon, Armin Shmilovici, Lior Rokach και Tomer Doitshman. *Comparison of state-of-the-art deep learning APIs for image multi-label classification using semantic metrics*. *Expert Systems with Applications*, 161:113656, 2020.

[35] Alberto Martin-Lopez. *AI-driven web API testing*. *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Companion Proceedings*, σελίδες 202–205, 2020.

[36] *OpenAPI-Specification*. https://github.com/OAI/OpenAPI-Specification. Ημερομηνία πρόσβασης: 22-1-2021.

[37] *About Swagger Specification*. https://swagger.io/docs/specification/about/. Ημερομηνία πρόσβασης: 22-1-2021.

[38] *Postman | The Collaboration Platform for API Development*. https://www.postman.com/. Ημερομηνία πρόσβασης: 22-1-2021.