ΔΙΕΘΝΕΣ ΠΑΝΕΠΙΣΤΗΜΙΟ ΤΗΣ ΕΛΛΑΔΟΣ

ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΗΛΕΚΤΡΟΝΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Δημιουργία πλατφόρμας marketing με τεχνικές βελτιστοποίησης

Των φοιτητών:
Οικονόμου Ιωάννη
Αρ. Μητρώου: 144187
Τσαντούλη Αναστάσιου
Αρ. Μητρώου: 164762

Επιβλέπων
Τζέκης Παναγιώτης
Καθηγητής

Ημερομηνία: 22/08/2025

Τίτλος: Δημιουργία πλατφόρμας marketing με τεχνικές βελτιστοποίησης
Κωδικός: 24298
Ονοματεπώνυμο φοιτητή: Οικονόμου Ιωάννης
Ονοματεπώνυμο φοιτητή: Τσαντούλης Αναστάσιος
Ονοματεπώνυμο εισηγητή: Τζέκης Παναγιώτης
Ημερομηνία ανάληψης: 31/10/2024
Ημερομηνία περάτωσης: 22/08/2025

Βεβαιώνουμε ότι είμαστε οι συγγραφείς αυτής της εργασίας και ότι κάθε βοήθεια την οποία είχαμε για την προετοιμασία της είναι πλήρως αναγνωρισμένη και αναφέρεται στην εργασία. Επίσης, έχουμε καταγράψει τις όποιες πηγές από τις οποίες κάναμε χρήση δεδομένων, ιδεών, εικόνων και κειμένου, είτε αυτές αναφέρονται ακριβώς είτε παραφρασμένες. Επιπλέον, βεβαιώνουμε ότι αυτή η εργασία προετοιμάστηκε από εμάς προσωπικά, ειδικά ως διπλωματική εργασία, στο Τμήμα Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων του ΔΙ.ΠΑ.Ε.

«Στις οικογένειες, τους φίλους και συναδέλφους μας
και όλα τα άτομα που ήταν εκεί
για την στήριξη που μας παρείχαν,
τον φίλο Χρήστο για τις συμβουλές στο frontend,
καθώς και όλους αυτούς που δοκίμασαν την εφαρμογή μας.»

Πρόλογος

Επιλέξαμε την υλοποίηση της συγκεκριμένης πτυχιακής εργασίας, διότι μας έδωσε την ευκαιρία να χρησιμοποιήσουμε πολλές και σύγχρονες τεχνολογίες, και να εφαρμόσουμε τεχνικές μηχανικής λογισμικού, ώστε να αναπτύξουμε ένα σύστημα το οποίο να έχει καλή απόδοση, να είναι κλιμακούμενο και εύκολα επεκτάσιμο. Επίσης, επειδή η πτυχιακή εργασία αφορά και την βελτιστοποίηση περιεχομένου με την χρήση γλωσσικών μοντέλων, είχαμε την δυνατότητα να πειραματιστούμε με διάφορα γλωσσικά μοντέλα, να δημιουργήσουμε τα κατάλληλα prompts και να και να αξιολογήσουμε την επίδρασή τους στην ποιότητα και την αποτελεσματικότητα των παραγόμενων κειμένων. Το αποτέλεσμα είναι μια πλατφόρμα που συνδυάζει πολλές καινοτόμες τεχνολογίες, ακολουθεί τις σωστές πρακτικές μηχανικής λογισμικού και προσφέρει στους χρήστες ένα σύγχρονο και αξιόπιστο εργαλείο για τη μαζική αποστολή και βελτιστοποίηση email.

Περίληψη

Η εργασία αυτή παρουσιάζει την ανάπτυξη της εφαρμογής Mail-Sender, μιας διαδικτυακής πλατφόρμας μαζικής αποστολής email με έμφαση στη βελτιστοποίηση του περιεχομένου μέσω τεχνολογιών μηχανικής μάθησης. Η αρχιτεκτονική του συστήματος είναι βασισμένη σε μικροϋπηρεσίες και χρησιμοποιείται message broker για κλιμακωσιμότητα και ευελιξία. Για την βελτιστοποίηση των email γίνεται χρήση ενός γλωσσικού μοντέλου, καθώς και η υπηρεσία Spamassassin, για την αξιολόγηση του Spam-score των email. Η υλοποίηση του front-end είναι βασισμένη στο framework Vue.js/Quasar. Η πλατφόρμα προσφέρει όλα τα απαραίτητα εργαλεία στους χρήστες, όπως στατιστικά καμπανιών και εργαλεία βελτίωσης περιεχομένου, με στόχο την αύξηση της απόδοσης των καμπανιών και τη μείωση του spam-score των email. Στα επόμενα κεφάλαια αναλύονται αναλυτικά οι τεχνολογίες που χρησιμοποιήθηκαν, η βάση δεδομένων, τα microservices, τα μοντέλα βελτιστοποίησης, η σχεδίαση του front-end, το deployment και η τελική εμπειρία χρήσης.

Creating a marketing platform with optimization techniques

Oikonomou Ioannis & Tsantoulis Anastasios

Abstract

This thesis presents the development of the Mail Sender application, a web-based platform for mass email delivery with a strong focus on content optimization through machine learning technologies. The system architecture is based on microservices and utilizes a message broker to ensure scalability and flexibility. Email optimization is performed using a language model, as well as the SpamAssassin service for evaluating the spam score of emails. The frontend is implemented using the Vue.js/Quasar framework. The platform provides users with all the necessary tools, such as campaign statistics and content improvement features, aiming to increase campaign effectiveness and reduce email spam scores. The following chapters provide a detailed analysis of the technologies used, the database structure, the microservices, the optimization models, the frontend design, the deployment process, and the final user experience.

Ευχαριστίες

Θα θέλαμε να ευχαριστήσουμε τους φίλους, την οικογένεια και τους συναδέλφους μας, που βοήθησαν και μας στήριξαν με διάφορους τρόπους κατά την διάρκεια αυτής της εργασίας. Επίσης θα θέλαμε να ευχαριστήσουμε και τον Κ. Τζέκη, που μας έδωσε την δυνατότητα να αναλάβουμε την εργασία αυτή και μας βοήθησε να την ολοκληρώσουμε.

Περιεχόμενα

Πρόλογος v
Περίληψη vi
Abstract vii
Ευχαριστίες viii
Περιεχόμενα ix
Κατάλογος Εικόνων xiv
Κατάλογος Πινάκων xiv
Συντομογραφίες xvi
Κεφάλαιο 1ο: Εισαγωγή 1
1.1 Εισαγωγή 1
1.2 Αρχιτεκτονική συστήματος 1
1.3 Επόμενα κεφάλαια 2
Κεφάλαιο 2ο: Τεχνολογίες 3
2.1 Εισαγωγή 3
2.2 Backend γλώσσα προγραμματισμού 3
2.2.1 Διαφορές Kotlin με Java 3
2.3 Βάση δεδομένων 5
2.3.1 MySQL 5
2.3.2 Μη σχεσιακές βάσης δεδομένων 5
2.3.3 Επιλογη σχεσιακής βασης ή μη σχεσιακης βάσης 5
2.3.4 Ανάλυση σχεσιακών βάσεων 6
2.3.5 Επιλογή της βάσης δεδομένων 7
2.4 Εργαλεία build για το backend 7
2.4.1 Εισαγωγή 7
2.4.2 Ant 7
2.4.3 Apache Maven 8
2.4.4 Gradle 8
2.4.5 Επιλογή Build Tool 8
2.5 Message Queue System 8
2.5.1 Εισαγωγή 8
2.5.2 RabbitMQ και Apache Kafka 9
2.5.3 RabbitMQ 9

2.5.4 Apache Kafka 9
2.5.5 Επιλογή μεταξύ Apache Kafka και RabbitMQ 10
2.6 Version Control System (VCS) 10
2.6.1 Εισαγωγή 10
2.6.2 Διαθέσιμα VCS συστήματα 11
2.6.3 Git 11
2.6.4 SVN 11
2.6.5 Επιλογή Git 11
2.6.6 Χρήση GitLab Docker Image Registry 12
2.7 Αρχιτεκτονική Back-end εφαρμογής 12
2.7.1 Εισαγωγή 12
2.7.2 Προβλήματα αναλογων υπηρεσιων 12
2.7.3 Πολυεπίπεδη αρχιτεκτονική 12
2.7.4 Αρχιτεκτονική εξάγωνου 13
2.7.5 Επιλογή αρχιτεκτονικής 14
2.8 Συστήμα αξιολόγησης ηλεκτρονικών μηνυμάτων 14
2.8.1 SpamAssassin 14
2.8.2 Πως λειτουργεί το SpamAssassin 14
2.9 Τεχνολογίες frontend 14
2.9.1 Εισαγωγή 14
2.9.2 React 14
2.9.3 Angular 15
2.9.4 Vue 15
2.9.5 Επιλογή framework 16
2.9.6 Quasar 16
2.10 Ασφάλεια και αυθεντικοποίηση χρηστών 17
2.10.1 Εισαγωγη 17
2.10.2 Αυθεντικοποίηση βάση συνεδρίας 17
2.10.3 Stateless αυθεντικοποίηση 17
2.10.4 Βήματα αυθεντικοποίησης του χρήστη με JWT 18
2.10.5 Spring Security 19
Κεφάλαιο 3ο: Βάση δεδομένων 20
3.1 Διαχείριση του Σχήματος Βάσης με Liquibase 20
3.2 Πίνακες βάσης δεδομένων 21

3.2.1 Πίνακας user 21
3.2.2 Πίνακας user_password_reset 22
3.2.3 Πίνακας user_account_confirmation 22
3.2.4 Πίνακας subscriber 23
3.2.5 Πίνακας _group 23
3.2.6 Πίνακας subscriber_group 24
3.2.7 Πίνακας campaign 24
3.2.8 Πίνακας campaign_group 24
3.2.9 Πίνακας campaign_subscriber 25
3.2.10 Πίνακας mail_content 25
3.2.11 Πίνακας email_outgoing_event 25
3.2.12 Πίνακας email_tracking 26
3.2.13 Πίνακας campaign_spam 26
3.2.14 Πίνακας mail_server 26
3.3 Indexes βάσης δεδομένων 27
3.4 Κανονικοποίηση βάσης δεδομένων 28
Κεφάλαιο 4ο: Υλοποίηση Back-end 29
4.1 Core module 29
4.1.1 Δομή του core module 29
4.1.2 Το API της εφαρμογής 30
4.1.3 Αλγόριθμος βελτιστοποίησης Email 37
4.1.4 Minio object storage 40
4.2 Executor module 41
4.2.1 Δομή του Executor module 41
4.2.2 Διεργασία αρχικοποίησης καμπάνιας 42
4.2.3 Διεργασία δημιουργίας και αποστολής email 43
4.2.4 Διεργασία τερματισμού καμπάνιας 45
4.2.5 Διεργασία αποστολής email κατά την εγγραφή χρήστη 45
4.2.6 Διεργασία ενημέρωσης κατάστασης παράδοσης email 46
4.3 Publisher module 47
4.3.1 Δομή του Publisher module 47
4.3.2 Αλληλεπίδραση με το RabbitMQ 48
4.3.3 Διαδικασία επεξεργασίας μηνυμάτων 49
4.4 Αρχιτεκτονική συστήματος 49
4.4.1 Αρχιτεκτονική υψηλού επιπέδου 51

4.4.2 Κλιμακωσιμότητα του συστήματος 53
4.4.3 Email servers 56
4.4.4 Βάση δεδομένων 57
Κεφάλαιο 5ο: Υλοποίηση front-end 58
5.1 Οργάνωση κώδικα 58
5.2 Layout System 59
5.2.1 Authentication Layout 59
5.2.2 Main Layout 59
5.2.3 Page Components 59
5.2.4 Index Page 59
5.2.5 Editor Page 60
5.2.6 Contacts Page 60
5.2.7 Campaign Page 60
5.2.8 Analytics Page 60
5.2.9 Profile Page 60
5.2.10 Login Page 61
5.2.11 Register Page 61
5.3 Styling 61
5.3.1 Dark mode 62
5.4 Api Integration 62
5.4.1 Παραμετροποίηση 62
5.4.2 Αυθεντικοποίηση 62
5.4.3 Διαχείριση καμπάνιας 62
5.4.4 Βελτιστοποίηση Email 63
Κεφάλαιο 6ο: Deployment εφαρμογής 65
6.1 Εγκατάσταση ως service 65
6.1.1 Εγκατάσταση των jar 65
6.1.2 Εγκατάσταση περιφερειακών προγραμμάτων 65
6.1.3 Εγκατάσταση RabbitMQ 66
6.1.4 Εγκατάσταση του MinIO 66
6.1.5 Εγκατάσταση SpamAssassin 67
6.1.6 Εγκατάσταση Nginx Proxy για το LM Studio 67
6.1.7 Εκτέλεση Microservices 68
6.2 Docker 69

6.2.1 Προ απαιτούμενα ... 69
6.2.2 Ανάπτυξη ... 70
6.2.3 Πρόσβαση στις Υπηρεσίες ... 70
6.3 Kubernetes ... 71
6.3.1 Προ απαιτούμενα ... 71
6.3.2 Συμπεράσματα ... 73
Κεφάλαιο 7ο: Χρήση εφαρμογής ... 74
7.1 Σελίδα εγγραφής χρήστη ... 74
7.2 Σελίδα εισόδου χρήστη ... 77
7.3 Αρχική σελίδα (Homepage) ... 78
7.4 Σελίδα διαχείρισης επαφών ... 79
7.5 Σελίδα δημιουργίας καμπάνιας ... 81
7.5.1 Δημιουργία περιεχομένου ... 82
7.5.2 Βελτιστοποίηση περιεχομένου email ... 83
7.5.3 Επιλογή παραληπτών καμπάνιας ... 84
7.6 Σελίδα σύνοψης καμπάνιας ... 85
7.7 Σελίδα συνολικών στατιστικών ... 87
7.8 Σελίδα προφίλ χρήστη ... 88
Κεφάλαιο 8ο: Συμπεράσματα και προτάσεις βελτίωσης ... 90
8.1 Συμπεράσματα ... 90
8.2 Προτάσεις βελτίωσης ... 90
ΒΙΒΛΙΟΓΡΑΦΙΑ ... 92

Κατάλογος Εικόνων

Εικόνα 1: Πίνακας DATABASECHANGELOG 21
Εικόνα 2: Μοντέλο Οντοτήτων-Συσχετίσεων βάσης δεδομένων 21
Εικόνα 3: Πλήρης αρχιτεκτονική συστήματος 52
Εικόνα 4: Παράδειγμα κλιμάκωσης 1 54
Εικόνα 5: Παράδειγμα κλιμάκωσης 2 55
Εικόνα 6: Παράδειγμα πολλαπλών mail server 56
Εικόνα 7: Παράδειγμα οριζόντιας κλιμάκωση ανάγνωσης 57
Εικόνα 8: Δομή φακέλων front-end 58
Εικόνα 9: Σελίδα εγγραφής χρήστη 74
Εικόνα 10: Επικύρωση στοιχείων στην σελίδα εγγραφής 75
Εικόνα 11: Επιτυχής εγγραφή 76
Εικόνα 12: Σελίδα εισόδου χρήστη 77
Εικόνα 13: Αρχική σελίδα εφαρμογής 78
Εικόνα 14: Σελίδα εισαγωγής επαφών 79
Εικόνα 15: Εισαγωγή CSV με επαφές 80
Εικόνα 16: Επιλογή ομάδας 81
Εικόνα 17: Δημιουργία καμπάνιας (Δημιουργία email) 83
Εικόνα 18: Δημιουργία καμπάνιας (Βελτιστοποίηση email) 84
Εικόνα 19: Δημιουργία καμπάνιας (Προγραμματισμός καμπάνιας) 85
Εικόνα 20: Λίστα καμπανιών χρήστη 85
Εικόνα 21: Σύνοψη καμπάνιας σε εκτέλεση 86
Εικόνα 22: Σύνοψη παραληπτών ολοκληρωμένης καμπάνιας 87
Εικόνα 23: Σύνοψη στατιστικών όλων των καμπανιών 88
Εικόνα 24: Προφίλ χρήστη 89

Κατάλογος Πινάκων

Πίνακας 1: Liquibase Changeset 20
Πίνακας 2: Endpoints /user 30
Πίνακας 3: Endpoints /subscriber 31
Πίνακας 4: Endpoints /group 32
Πίνακας 5: Endpoints /campaign 33
Πίνακας 6: Endpoints /resources 34
Πίνακας 7: Endpoints /track 34
Πίνακας 8: Endpoints /campaign-statistics 34
Πίνακας 9: Endpoints /openrate 35
Πίνακας 10: Χρήση @UserID annotation 36
Πίνακας 11: Συνάρτηση υπολογισμού temperature 39
Πίνακας 12: System prompt βελτιστοποίησης email 40
Πίνακας 13: Promted rules βελτιστοποίησης email 40
Πίνακας 14: MinIO URL 41
Πίνακας 15: Ψευδοκώδικας initializeCampaignExecution 43
Πίνακας 16: Nginx configuration 68

Πίνακας 17: Βασικές ιδιότητες ρύθμισης στο application.properties 69

Συντομογραφίες

API	Application Programming Interface
JVM	Java Virtual Machine
HTTP	Hyper Text Transfer Protocol
ORM	Object-Relational Mapping

Κεφάλαιο 1ο: Εισαγωγή

1.1 Εισαγωγή

Η εφαρμογή Mail-sender είναι μια διαδικτυακή πλατφόρμα μαζικής αποστολής email. Δίνει την δυνατότητα στους χρήστες να εισάγουν και διαχειρίζονται ομάδες συνδρομητών, να δημιουργούν καμπάνιες email, και να τις αποστέλλουν εύκολα στις επιλεγμένες ομάδες συνδρομητών τους. Επίσης η πλατφόρμα προσφέρει αναλυτικά στατιστικά στοιχεία σχετικά με τις καμπάνιες χρηστών, όπως τα ποσοστά των επιτυχώς απεσταλμένων email, τα ποσοστά των email που ανοίχτηκαν από τους παραλήπτες, καθώς και το ποιοι συνδρομητές συγκεκριμένα άνοιξαν το email που παρέλαβαν από την πλατφόρμα.

Το ιδιαίτερο χαρακτηριστικό της πλατφόρμας είναι ότι προσφέρει την δυνατότητα στους χρήστες να βελτιστοποιούν το περιεχόμενο των καμπανιών email, με την χρήση μοντέλων μηχανικής μάθησης. Η βελτιστοποίηση στοχεύει κατά κύριο λόγο στην μείωση του spam-score των email, ώστε να μεγιστοποιηθεί η πιθανότητα να φτάσει το email στον τελικό παραλήπτη, και να μην κοπεί από τα διάφορα φίλτρα spam των παρόχων email. Επίσης, μπορεί να βελτιώσει και το email όσον αφορά τα ορθογραφικά λάθη, τη δομή και τη σαφήνεια του κειμένου, ενισχύοντας έτσι τη συνολική αναγνωσιμότητα και την αποτελεσματικότητα του μηνύματος.

1.2 Αρχιτεκτονική συστήματος

Η αρχιτεκτονική του συστήματος έχει σχεδιαστεί με γνώμονα την κλιμακωσιμότητα, ώστε να μπορεί να διαχειρίζεται αυξανόμενο όγκο χρηστών και αποστολών email χωρίς μείωση της απόδοσης. Τα επιμέρους υποσυστήματα είναι διαχωρισμένα σύμφωνα με την αρχή του separation of concerns, επιτρέποντας ευκολότερη συντήρηση, ανεξάρτητη ανάπτυξη και απομόνωση σφαλμάτων.

Πιο συγκεκριμένα, το backend είναι χωρισμένο σε τρία microservices, το core microservice που αφορά την υλοποίηση του API της εφαρμογής, το executor microservice που αφορά την εκτέλεση των καμπανιών χρήστη και την δημιουργία των email που θα αποσταλούν, και το publisher microservice που είναι υπεύθυνο για την επικοινωνία με τους mail server και την αποστολή των email. Και τα τρία microservices χρησιμοποιούν μια κοινή σχεσιακή βάση δεδομένων, και κάθε microservice έχει πρόσβαση σε συγκεκριμένους πίνακες. Για την επικοινωνία μεταξύ των microservices χρησιμοποιείται ένας message broker, που διασφαλίζει την ασύγχρονη και αξιόπιστη ανταλλαγή μηνυμάτων μεταξύ τους, εξασφαλίζοντας χαμηλό coupling και υψηλή επεκτασιμότητα. Τέλος, βασικό κομμάτι της backend αρχιτεκτονικής είναι και το γλωσσικό μοντέλο που χρησιμοποιείται για την βελτιστοποίηση των email, καθώς και το λογισμικό SpamAssasin που χρησιμοποιείται για την αξιολόγηση του spam-score των email.

Όσον αφορά το frontend, η υλοποίηση του έχει γίνει με το framework Vue.js και την βιβλιοθήκη Quasar. Η επικοινωνία με το backend γίνεται αποκλειστικά με την χρήση των REST APIs που υλοποιεί το core microservice. Έχει δοθεί ιδιαίτερη σημασία στον σχεδιασμό της διεπαφής χρήστη ώστε να είναι απλή, ευχάριστη και λειτουργική, διευκολύνοντας την πλοήγηση και τη χρήση της πλατφόρμας ακόμη και από μη τεχνικούς χρήστες.

1.3 Επόμενα κεφάλαια

Στα επόμενα κεφάλαια γίνεται αναλυτική επεξήγηση του συστήματος. Πιο συγκεκριμένα, στην αρχή γίνεται μια περιγραφή όλων των τεχνολογιών που χρησιμοποιήθηκαν, και ο λόγος που επιλέχθηκαν οι συγκεκριμένες τεχνολογίες. Στην συνέχεια περιγράφεται αναλυτικά η βάση δεδομένων, η μορφή της και το σκεπτικό πίσω από την σχεδίαση της. Έπειτα, γίνεται επεξήγηση του backend, εμβαθύνοντας στο κάθε microservice της εφαρμογής, στην επικοινωνία μεταξύ των microservices, καθώς και τρόπους κλιμακωσιμότητας τους. Επίσης αναλύεται η διαδικασία βελτιστοποίησης με την χρήση των γλωσσικών μοντέλων και του λογισμικού SpamAssasin. Στην συνέχεια ακολουθεί η ανάλυση του front-end και της σχεδίασης του. Το deployment της εφαρμογής είναι το επόμενο κεφάλαιο, στο οποίο γίνεται επεξήγηση του τρόπου με τον οποίο η εφαρμογή μπορεί να εγκατασταθεί και να ρυθμιστεί, ώστε να είναι έτοιμη προς χρήση. Τέλος, γίνεται μια βήμα προς βήμα ανάλυση της χρήσης της εφαρμογής, παρουσιάζοντας όλα τα βασικά σενάρια χρήσης, από την εγγραφή και εισαγωγή συνδρομητών, μέχρι τη δημιουργία, αποστολή και παρακολούθηση καμπανιών. Μέσα από αυτό το κεφάλαιο ο αναγνώστης αποκτά πλήρη εικόνα για τη λειτουργικότητα της πλατφόρμας, καθώς και για τα διαθέσιμα εργαλεία που προσφέρονται για τη βελτιστοποίηση των email καμπανιών τους.

Κεφάλαιο 2ο: Τεχνολογίες

2.1 Εισαγωγή

Σε αυτό το κεφάλαιο θα ασχοληθούμε με τις τεχνολογίες που χρησιμοποιήθηκαν για την δημιουργία της front-end εφαρμογής, της backend εφαρμογής καθως και τις περιφερειακές υπηρεσίες/services που χρησιμοποιήθηκαν.

2.2 Backend γλώσσα προγραμματισμού

Η Kotlin[1] αποτελεί μια σχετικά καινούργια γλώσσα που κάνει χρήση του JVM. Εμφανίστηκε για πρώτη φορά το 2011 και από τότε έχει καταφέρει να πάρει ένα αρκετά μεγάλο μερίδιο χρήσης στην χώρο της πληροφορικής. Σαν γλώσσα πλέον μπορεί να αντικαταστήσει πλήρως την Java που πλέον δείχνει την ηλικία της σαν γλώσσα.

2.2.1 Διαφορές Kotlin με Java

Ένα από τα μεγαλύτερα πλεονεκτήματα της Kotlin ενάντι της Java είναι η δυνατότητα να υπάρχουν τιμές null χωρίς να χρειάζεται κάποιο εξτρά wrap όπως το Optional που έχει η java μετρά την έκδοση 8. Αντίθετα η Kotlin ενσωματώνει την τιμή null και δίνει κατάλληλες μεθόδους και εργαλεία για την διαχείριση της χωρίς ο προγραμματιστής να έχει ένα αντικείμενο optional ή συνέχεια να πρέπει να γραφεί try/catch για την αποφυγή exception.

Επιπρόσθετα, η Kotlin υποστηρίζει πραγματικούς τύπους συναρτήσεων πράγμα που διορθώνει ένα μεγάλο αρνητικό της Java που πετύχαινε παρόμοιο functionality με την χρήση SAM (Single Abstract Method)[2] που πρακτικά μετατρέπουν μια λάμδα έκφραση σε ένα interface με μια αφηρημένη μέθοδο. Αντίθετα οι λάμδα συναρτήσεις αντιμετωπίζονται ως πραγματικές τιμές πράγμα που σε συνδυασμό με τα coroutines και της inline συναρτήσεις επιτρέπει πολύ πιο readable και resource efficient αποτέλεσμα.

Η υποστήριξη του invariance στους πίνακες της Kotlin μας βοηθάει να αποτρέπουμε σφάλματα τύπου κατά την εκτέλεση. Πρακτικά η Kotlin επιτρέπει μόνο έναν συγκεκριμένο τύπο/κλάση μέσα σε έναν πίνακα ακόμα και εάν αυτός αποτελεί «πατερά» ή «παιδί» κάπου αλλού τύπου/κλάσης. Αυτό εξασφαλίζει ότι ο τύπος του πίνακα θα είναι πάντα ένας σταθερός, σε αντίθεση με την Java που το επιτρέπει και μπορεί να οδηγήσει σε runtime exceptions.

Μια πολύ σημαντική διαφορά μεταξύ των 2 γλωσσών είναι ότι η Kotlin δεν υποστηρίζει checked exceptions. Ενώ με την java κάποιος πρέπει να ορίσει στην υπογραφή της μεθόδου εάν κάνει throw κάποιο exception ή να γίνει αναγκαστική διαχείριση του με την try-catch στην Kotlin δεν υπάρχει αυτή η ανάγκη. Σε συνεργασία με το nullability των μεταβλητών καθώς και με την διαφορετική νοοτροπία διαχειρίσεις σφαλμάτων της Kotlin με την χρήση κλειστών δομών (sealed classes) επιτρέπει στον προγραμματιστή να γράψει κώδικα ο οποίος είναι πιο εκφραστικός και λιγότερα δυσνόητος.

Στα υπερ. της Kotlin επίσης εντάσσετε και η διαφοροποίηση δομών δεδομένων ανάλογα με το εάν είναι ευμετάβλητες ή όχι. Σε αντίθεση με την Java, η Kotlin παρέχει διαφορετικός τύπους όπως πχ για λίστες η οποίες μπορεί ή μπορεί και να μην είναι διαθέσιμες για επεξεργασία μετρά την δημιουργία τους. Για παράδειγμα val z = listOf(1,2,3) η μεταβλητή z δεν μπορεί να πάρει νέα δεδομένα μετρά την δημιουργία της. Αντίθετα, η Val z = mutableListOf(1,2,3) παρέχει την δυνατότητα να προσθέσει κάποιος

καινούργια δεδομένα. Αυτή είναι τέλειος διαφορετική λογική από αυτή που έχει η Java οπού εκεί μπορεί ο προγραμματιστής να προσθέσει η να αφαιρέσει δεδομένα ανάπαυσα στιγμή από μια δομή δεδομένων.

Αρκετά μεγάλη διαφορά επίσης εμφανίζεται στους visibility modifiers που υπάρχουν στις δυο γλώσσες. Για παράδειγμα οι διαθέσιμοι τύποι για την Kotlin είναι οι εξής: private, protected, internal, και public με default τον public. Η java από την άλλη μερια παρέχει τους εξής: Default, Private, Protected, Public. Οι διαφορές εμφανίζονται στον Default που επιλεγεί η κάθε γλώσσα καθώς και την προσθήκη του internal.

Αρκετά δυνατό πλεονέκτημα επίσης δίνει στην Kotlin τα coroutines[3] που κάνουν πολύ απλή την δημιουργία ασύγχρονου κώδικα ενσωματώνοντας μέσα σε μια μέθοδο που είναι ορισμένη ως suspend το πεδίο του continuity. Αντίθετα το ίδιο πράγμα με την java αποτελεί μια πολύ δύσκολη διαδικασία η οποία προϋποθέτει πάρα πολύ μεγάλη εξοικειώσει του χρήστη με τα threads.

Ένα επίσης πάρα πολύ σημαντικό πλεονέκτημα αποτελεί το γεγονός ότι η Kotlin είναι σχεδιασμένη να λειτουργεί ως Object Oriented[4] γλώσσα αλλά και ως functional programming γλώσσα[5]. Με αυτό τον τρόπο είναι δυνατόν να σχεδιαστεί μια εφαρμογή φεύγοντας από το κλασσικό τρόπο που ορίζεται από τον Object Oriented προγραμματισμό, αλλά και ακολουθεί το functional programming που έχει βάσεις κυρίως στα μαθηματικά. Ενώ ο functional προγραμματισμός αποτελεί πιο δυσνόητη έννοια στην αρχή για να ξεκινήσει κάποιος να γράφει κώδικα επιτρέπει μια πολύ πιο απλή δομή κώδικα καθώς δεν υπάρχουν αντικείμενα και κρυφές καταστάσεις αντικειμένων αλλά μόνο συναρτήσεις οι οποίες μπορούν να περαστούν ως ορίσματα σε άλλες συναρτήσεις ή να επιστραφούν σαν αποτελέσματα. Αυτή την υβριδική δυνατότητα δεν την παρέχει η Java στον βαθμό που η Kotlin προσπάθησε να συνεχίσει από προηγούμενες προσπαθείς στο JVM όπως με την Scala.

Εν συντομία η Kotlin παρέχει τα εξής πλεονεκτήματα όσο αφορά την Java:

- *Λάμδα εκφράσεις + inline συναρτήσεις = αποδοτικές προσαρμοσμένες δομές ελέγχου*
- *Συναρτήσεις επέκτασης (extension functions)*
- *Ασφάλεια ως προς το null (null-safety)*
- *Έξυπνα cast τύπων (smart casts) (Java 16: Pattern Matching για instaceof)*
- *Πρότυπα συμβολοσειρών (string templates) (Java 21: String Templates - Preview)*
- *Ιδιότητες (properties) αντί για get/set μεθόδους*
- *Primary constructors*
- *Ανάθεση πρώτης κλάσης (first-class delegation)*
- *Τύποι μεταβλητών με αυτόματη αναγνώριση (type inference) (Java 10: var)*
- *Μονοσύνολα (singletons) object*
- *Declaration-site variance και type projections*
- *Εκφράσεις εύρους (range expressions) (π.χ. 1..10)*
- *Υπερφόρτωση τελεστών (operator overloading)*
- *Συνοδευτικά αντικείμενα (companion objects)*
- *Κλάσεις δεδομένων (data classes) για αυτόματη δημιουργία equals, hashCode, toString, κ.λπ.*
- *Coroutines για ασύγχρονο και μη μπλοκαρισμένο κώδικα*
- *Συναρτήσεις κορυφαίου επιπέδου (top-level functions), έξω από κλάσεις*
- *Προεπιλεγμένες τιμές παραμέτρων (default arguments)*
- *Ονομαστικές παράμετροι (named parameters)*
- *Συναρτήσεις σε μορφή εντολών (infix functions) για πιο φυσική σύνταξη*
- *Δηλώσεις expect και actual για κοινό κώδικα μεταξύ platform (multiplatform)*
- *Λειτουργία "Explicit API" και καλύτερος έλεγχος του δημόσιου API*

2.3 Βάση δεδομένων

Για την μόνιμη αποθήκευση των δεδομένων αποφασίστηκε να χρησιμοποιηθεί η MySQL.

2.3.1 MySQL

Η MySQL αποτελεί μια σχεσιακή βάση δεδομένων ανοιχτού κώδικα που χρησιμοποιεί SQL για την δημιουργία και την διαχειρίσεις βάσεων δεδομένων. Η MySQL χρησιμοποιεί για την αποθήκευση δεδομένων πίνακες με γραμμές και στήλες που οργανώνονται σε σχήματα. Κάθε σχήμα χαρακτηρίζει τον τρόπο με τον οποίον είναι οργανωμένα τα δεδομένα καθώς, πως αυτά αποθηκεύονται και περιγραφή την σχέση μεταξύ των διάφορων πινάκων. Τα πεδία μπορεί να έχουν πλήθους τύπος όπως χαρακτήρες, νούμερα, ημερομηνίες, καθώς και πλέον JSON.

2.3.2 Μη σχεσιακές βάσης δεδομένων

Μια εναλλακτική στα σχεσιακά συστήματα βάσεων δεδομένων αποτελούν τα μη σχεσιακά συστήματα βάσεων δεδομένων. Με τα πιο γνωστά να είναι η MongoDB, Cassandra και το Neo4j αποτελούν συστήματα τα οποία δεν έχουν κάποια αυστηρή δόμηση των δεδομένων σε αντίθεση με τα σχεσιακά τα οποία έχουν αυστηρό τρόπο πως θα υπάρχει αποθηκευμένο ένα entity μέσα στην βάση. Ένα πολύ ακόμα μεγάλο θετικό αποτελεί το γεγονός ότι συνήθως χαρακτηρίζονται από πολύ μεγαλύτερη ταχύτητα ανάγνωσης και εγγραφής των δεδομένων. Παράλληλα κάποια από τα συστήματα έχουν συγκεκριμένο scope χρήσης πράγμα που τα κάνουν πολύ καλύτερες εναλλακτικές συγκριτικά από τα κλασικές σχεσιακές βάσεις (όπως για παράδειγμα το Neo4J που αποτελεί μια βάση για αποθήκευση διανυσμάτων και εύρεση σχέσεων μεταξύ αντικειμένων).

Παρόλα αυτά υπάρχουν και αρνητικά όσο αφορά τις μη σχεσιακές βάσεις δεδομένων. Οι σχεσιακές βάσεις δεδομένων προσφέρουν αυστηρότητα για το πως αποθηκεύεται ένα entity μέσα σε μια βάση δεδομένων, πράγμα που βοηθάει σε περίπτωση που θέλουμε έναν συγκεκριμένο τρόπο υπάρξεις κάποιον οντοτήτων. Παράλληλα η ύπαρξη του ACID (Atomicity, Consistency, Isolation, Durability)[6] μας εγγυάται ότι οι συναλλαγές στην βάση δεδομένων λειτουργούν αξιόπιστα. Ποιο συγκεκριμένα η ατομικότητα εγγυάται ότι όλες οι ενέργειες θα εκτελεστούν επιτυχώς ή καμία δεν θα εφαρμοστεί και κατά συνέπεια θα έχουμε το γνωστό rollback. Η συνέπεια μας έγκειται ότι οι κανόνες τις βάσεις θα τηρηθούν, για παράδειγμα τα constraints θα τηρηθούν καθώς και η ύπαρξη ξένων κλειδιών. Η απομόνωση μας δίνει την δυνατότητα να μπορούμε να γλυτώσουμε από προβλήματα όπως να διαβάσει κάποιος μη οριστικοποιημένες αλλαγές δεδομένων ή να χαθούν ενημερώσεις λόγω ταυτοχρόνων ενεργειών, θέτει λοιπόν ότι σε παράλληλες εκτελέσεις συναλλαγών δεν θα χαθεί κάποια. Τέλος η μονιμότητα μας εξασφαλίζει ότι οποίο δεδομένο έχει γραφτεί στην βάση τα δεδομένα δεν θα χαθούν. Αυτό επιτυγχάνετε με την χρήση αρχείων logs ή της μεθόδου journaling (παρόμοια που υπάρχει και στα λειτουργικά συστήματα Linux). Τέλος, οι σχεσιακές βάσεις δεδομένων με την χρήση της SQL δίνει την δυνατότητα στον προγραμματιστή να μπορεί ευκολά να φτιάξει πολύπλοκα ερωτήματα μεταξύ πολλών πινάκων και να τα συσχέτιση μεταξύ τους με σχετική ευκολία.

2.3.3 Επιλογη σχεσιακής βασης ή μη σχεσιακης βάσης

Για την εργασία αποφασίστηκε να χρησιμοποιηθεί ένα σχεσιακό σύστημα βάσεων δεδομένων. Αυτή η απόφαση πάρθηκε καθώς γνωρίζουμε ότι θα υπάρχει μεγάλος όγκος δεδομένων, υπάρχει ανάγκη για αυστηρή δομή των δεδομένων χωρίς αυτά να χρειάζεται να έχουν αλλαγές μεταξύ τους, υπάρχει ανάγκη για συσχέτιση πολλών δεδομένων μεταξύ πινάκων καθώς και ανάγκη ύπαρξης του ACID.

2.3.4 Ανάλυση σχεσιακών βάσεων

Στην αγορά υπάρχουν αρκετές σχεσιακές βάσεις δεδομένων για να μπορεί κάποιος να διαλέξει. Κάποιες από αυτές που θα αναλυθούν παρακάτω είναι η MySQL, PostgreSQL, SQL server, oracle db., maria db. και η SQL lite.

2.3.4.1 MySQL

Η MySQL[7] αποτελεί μια βάση δεδομένων πλέον της oracle η οποία διαμοιράζεται και με Open Source(GPL) άδεια χρήσης. Αποτελεί μια από της πιο δημοφιλής βάσεις με αρκετά καλή απόδοση αι ταχύτητα. Με την έκδοση 8.xx η MySQL δίνει και την δυνατότητα να αποθηκεύει δεδομένα και με μορφή Json πράγμα που στην περίπτωση που χρειαζόταν ο προγραμματιστής δυο βάσεις δεδομένων, μια σχεσιακή και μια μη σχεσιακή πλέον μπορεί να το κάνει μέσα από την MySQL. Ταυτόχρονα η εύκολη δυνατότητα για clustering που παρέχει η MySQL με την χρήση ενός MySQL router και MySQL nodes δίνει την δυνατότητα για την διατηρήσει των δεδομένων σε πολλά nodes καθώς και την ευελιξία για καταμερισμό του φόρτου εργασίας.

2.3.4.2 PostgreSQL

Η PostgreSQL[8] αποτελεί μια αντικειμενοστραφής και πλήρωνες συμβατή με SQL βάση δεδομένων ανοιχτού κώδικα. Αποτελεί μια από τις γνωστότερες σχεσιακές βάσεις δεδομένων με μεγάλη υποστήριξη από την κοινότητα ανοιχτού κώδικα. Υποστηρίζει πολύπλοκα ερωτήματα, JSON τύπους δεδομένων, καθολικά και παραμετρικά indexes, CTEs, window functions, stored procedures και πλήρη ACID συμμόρφωση. Χάρη στην προηγμένη της υποστήριξη για semi-structured δεδομένα μέσω JSONB, μπορεί να καλύψει περιπτώσεις χρήσης που απαιτούν και SQL και NoSQL λειτουργίες. Είναι ιδανική επιλογή για εφαρμογές που απαιτούν ακρίβεια, αξιοπιστία, και δυνατότητα παραμετροποίησης. Παράλληλα, υποστηρίζει replication, partitioning, και κλιμάκωση, καθιστώντας την κατάλληλη για enterprise ή επιστημονικές εφαρμογές.

2.3.4.3 Microsoft SQL Server

Η Microsoft SQL Server[9] αποτελεί μια εμπορική σχεσιακή βάση δεδομένων που αναπτύχθηκε από την Microsoft κυρίως για εταιρικά περιβάλλοντα και κυρίως για χρήση στο οικοσύστημα της .NET . Υποστηρίζει την επέκταση της SQL, T-SQL και προσφέρει δυνατότητες όπως triggers, stored procedures, views, in-memory tables και ενσωμάτωση με τα Active Directories της Microsoft. Δεν αποτελεί μια λύση ανοιχτού κώδικα αλλά προσφέρει μια δωρεάν έκδοση της την SQL Server Express.

2.3.4.4 Oracle DB

Η Oracle Db[10] αποτελεί και αυτή μια εμπορική σχεσιακή βάση δεδομένων αναπτυγμένη από την Oracle. Χρησιμοποιείτε κύριος σε τραπεζικά και ERP συστήματα. Υποστηρίζει το πρότυπο SQL καθώς και επεκτείνει της δυνατότητες της με την χρήση της γλωσσάς PL/SQL μιας αρκετά δυνατής scripting γλώσσας που προσφέρει ευκολότερη, γρηγορότερη και παραμετροποίηση ανάκτηση δεδομένων. Επίσης παρέχει δυνατότητες όπως rationing, metalized views, advanced replication, compression καθώς και παράλληλη εκτέλεση ερωτημάτων. Διαθέτει προηγμένα χαρακτηριστικά για ασφάλεια, ανάκτηση δεδομένων, και διαχείριση μεγάλου όγκου πληροφορίας, και υποστηρίζει JSON, XML, και Special δεδομένα. Χάρη στο Oracle RAC (Real Application Clusters), επιτρέπει την ταυτόχρονη πρόσβαση πολλών servers σε μία βάση για υψηλή διαθεσιμότητα.

2.3.4.5 MariaDB

Η MariaDB[11] είναι fork της MySQL, που δημιουργήθηκε από τους αρχικούς δημιουργούς της MySQL όταν η Oracle την αγόρασε. Είναι πλήρως ανοιχτού κώδικα, πλήρως ως συμβατή με MySQL. Υποστηρίζει SQL και περιλαμβάνει πολλά χαρακτηριστικά που δεν υπάρχουν στην MySQL, όπως Aria και ColumnStore storage engines, βελτιωμένη ασφάλεια, γρηγορότερα indexes, καθώς και υποστήριξη για JSON, GIS και dynamic columns.

2.3.4.6 SQLite

Τέλος, η SQLite αποτελεί μια πολύ ελαφριά σχεσιακή βάση δεδομένων που μπορεί να ενσωματωθεί μέσα στην εφαρμογή και δεν χρειάζεται ξεχωριστό server. Αυτό την κάνει ιδανική για mobile εφαρμογές καθώς και για εφαρμογές desktop οπού δεν υπάρχει μεγάλη πολυπλοκότητα και τα δεδομένα που πρέπει να αποθηκευτούν αποτελούν συνήθως μόνο user data.

2.3.5 Επιλογή της βάσης δεδομένων

Για την πτυχιακή εργασία κρίθηκαν 3 από αυτές τις βάσεις χρήσιμες η MySQL, PostgreSQL και η MariaDB. Από αυτές αποφασίστηκε να χρησιμοποιηθεί η MySQL λόγω μεγάλης εξοικειώσεις και όσο αφορά το κόμματοί της SQL, της παραμετροποίησης καθώς και την εύκολη δημιουργία clustering με αυτή. Αυτό βέβαια δεν σημαίνει ότι η PostgreSQL και η MariaDB δεν θα μπορούσαν να αντικαταστήσουν την MySQL. Αντίθετα, ειδικά για την PostgreSQL,l θα προσέφερε πολύ μεγαλύτερη ταχύτητα και ευκολία καθώς η προτεινομένη σχεσιακή βάση δεδομένων για χρήση με το Spring boot Web Flux και με το R2DBC είναι η PostgreSQL. Στο μέλλον θα μπορούσε να αλλάξει από την μια στην άλλη χωρίς κάποια ιδιαίτερη δυσκολία.

2.4 Εργαλεία build για το backend

2.4.1 Εισαγωγή

Για την δημιουργία εφαρμογών Spring boot Java χρειαζόμαστε κάποιο εργαλείο το οποίο θα μας κάνει compile τον κώδικα και θα μας πράξει το τελικό εκτελέσιμο πρόγραμμα. Στην παρούσα κατάσταση εφόσον χρησιμοποιούμε Kotlin σε JVM μπορούμε να χρησιμοποιήσουμε οποιοδήποτε build tool είναι διαθέσιμο για την Java. Τρία από αυτά ξεχωρίζουν βάση και ιστορικότητας, το Ant, το Maven και το Gradle τα οποία θα αναλυθούν παρακάτω το κάθε ένα ξεχωριστά.

2.4.2 Ant

Η αδυναμία του παραδοσιακού Make να προσαρμοστεί στις ανάγκες του οικοσυστήματος μιας και προοριζόταν κυρίως για την γλώσσα C έφερε στο προσκήνιο το Apache Ant (Another Neat Tool)[12]. Το Ant είναι μια βιβλιοθήκη της Java το οποίο προσφέρει αυτόματες διαδικασίες build για τις εφαρμογές Java και όχι μόνο. Ήταν κομμάτι του Apache Tomcat μέχρι το 2000 που έγινε αυτόνομο. Το Ant χρησιμοποιεί XML αρχεία για να γίνει η παραμετροποίηση του και αποθηκεύει τις διαδικασίες που κάνουν build το project με ένα αρχείο που βάση συμβάσης ονομάζεται build.xml. Επειδή το Ant όμως δεν επιβάλει με ποιον τρόπο θα είναι δομημένο το project είχε σαν αποτέλεσμα ο προγραμματιστής να πρέπει να γράψει μονός του όλες τις διαδικασίες που θα έπρεπε το ant να εκτελέσει. Αυτό είχε σαν αποτέλεσμα τα xml αρχεία να μεγαλώνουν παράλληλα με τον κώδικα πράγμα που από ένα σημείο και μετά καθιστά αρκετά δύσκολα στην συντήρηση τους. Επίσης ένα μεγάλο αρνητικό που έκανε το Ant να αντικατασταθεί εν τελεί από το Maven ήταν η έλλειψη μηχανισμού Dependency Management. Αυτό

ναι μεν προστέθηκε αργότερα με το Apache Ivy αλλά δεν απέκτησε ποτέ ξανά το μερίδιο το οποίο έχασε από το Maven.

2.4.3 Apache Maven

Το Apache Maven[13] αποτελεί τον βασικό αντικαταστάτη του Apache Ant. Είναι ένα dependency management και build automation εργαλείο το οποίο χρησιμοποιείται από εφαρμογές Java και Kotlin αντίστοιχα. Χρησιμοποιεί και αυτό XML αρχεία όπως το Ant αλλά με έναν πολύ πιο εύκολο τρόπο γραφής και κατ' επέκτασης κατανόησης του. Αυτό επιτυγχάνετε κυρίως μέσα από ισχυρές συμβάσης που το Maven χρησιμοποιεί. Κάποιες από αυτές τις συμβάσεις για παράδειγμα είναι το πως πρέπει να είναι δομημένος ο κώδικας με τα πακέτα καθώς και το που πρέπει να βρίσκονται τα tests. Έτσι, το maven μπορεί να παρέχει προκαθορισμένες εντολές και ενεργείς που χρειάζεται το project που τα ονομάζει goals. Παράλληλα, η υποστηρίξει για το Dependency management, τα plugin που μπορεί να προσθέσει κάποιος για να κάνει μια συγκεκριμένη δουλειά (να δημιουργήσει κάποιος ένα docker image) καθώς και η ευκολία στην χρήση του το έκαναν αρκετά δελεαστικό στην χρήση του ενάτη του Ant. Βέβαια, όλη αυτή απλούστευση ήρθε με το τίμημα ότι συγκριτικά με το Ant, το Maven, είναι αρκετά πιο δύσκολο να παραμετροποιηθεί, καθώς και το γεγονός ότι ναι μεν τα αρχεία παραμετροποίησης είναι πιο μικρά από αυτά του Ant παραμένουν μεγάλα και δύσκολα στην διαχείριση τους.

2.4.4 Gradle

Το Gradle[14] αποτελεί επίσης ένα εργαλείο για διαχείριση Dependency και την αυτοματοποίηση της διαδικασίας build. Η πρώτη μεγάλη ουσιαστική διαφορά είναι ότι δεν χρησιμοποιεί αρχεία XML όπως το Ant και το Maven αλλά κάτι γνωστό ως DSL (Domain Specific Language). Το Gradle λοιπόν προσφέρει ένα DSL το οποίο είναι προσαρμοσμένο στις ανάγκες του ως build automation tool και δεν προσπαθεί να προσαρμόσει το XML στις ανάγκες του. Διαθέσιμες είναι δυο γλώσσες στις οποίες το DSL είναι διαθέσιμο, Groovy και Kotlin. Για αυτό το λόγω είναι πολύ πιο εύκολο το Gradle να παρέχει δυνατότητες για αυτόματη συμπλήρωση καθώς και για έλεγχο λαθών. Παρόμοια με το maven, το Gradle λειτουργεί κυρίως μέσα από plugins για τις διάφορες ενεργείς που θέλουμε να συμπεριληφθούν σε αυτό.

2.4.5 Επιλογή Build Tool

Λόγο της επιλογής μας της Kotlin ως γλώσσα για την εργασία μας αποφασίσαμε να χρησιμοποιήσουμε το Gradle σαν build και dependency management εργαλείο. Η δυνατότητα παραμετροποίησης του Gradle με Kotlin μας αφήνει να έχουμε μια ομοιομορφία μεταξύ του αρχικού μας κώδικα και των task που έχουμε θέσει στο Gradle. Με αυτό τον τρόπο γίνεται πιο ευκολά η συντήρηση του ενώ ταυτόχρονα κρατάμε το μέγεθος του και την πολυπλοκότητα του χαμηλά.

2.5 Message Queue System

2.5.1 Εισαγωγή

Καθώς η εργασία μας βασίζεται στην αρχιτεκτονική των microservices, θα πρέπει να υπάρχει και κάποιος τρόπος να αποθηκεύονται προσωρινά τα δεδομένα όταν φεύγουν από την υπηρεσία Α για να παραληφθούν από την υπηρεσία Β. Για αυτό τον λόγω χρειάστηκε να χρησιμοποιηθεί ένα σύστημα ουρών ή αλλιώς message broker.

2.5.2 RabbitMQ και Apache Kafka

Υπάρχουν δυο βασικές λύσεις σε αυτό το ζήτημα, το RabbitMQ και το Apache Kafka προσφέροντας ο καθένας ξεχωριστά αρχιτεκτονικά παραδείγματα και σύνολα χαρακτηριστικών. Αυτή η ενότητα παρέχει μια συνοπτική επισκόπηση των αντίστοιχων πλεονεκτημάτων και μειονεκτημάτων τους, θέτοντας τις βάσεις για την κατανόηση της εφαρμοσιμότητάς τους σε διάφορες αρχιτεκτονικές συστημάτων.

2.5.3 RabbitMQ

Το RabbitMQ αποτελεί έναν open-source message broker που βασίζεται στο πρωτόκολλο Advanced Message Queuing Protocol ή αλλιώς AMQP. Υποστηρίζει σημείο-προς-σημείο αποστολή μηνυμάτων καθώς και publish/subscribe. Παρέχει μεγάλη αξιοπιστία στην παράδοση μηνυμάτων.

2.5.3.1 Πλεονεκτήματα του RabbitMQ

Κάποια από τα πλεονεκτήματα του RabbitMQ είναι:

- *Το RabbitMQ υπερέχει σε περίπλοκα σενάρια δρομολόγησης μηνυμάτων μέσω των τύπων ανταλλαγών του (direct, fanout, topic, headers) και των μηχανισμών δέσμευσης. Αυτό επιτρέπει εξαιρετικά λεπτομερή έλεγχο της διανομής μηνυμάτων σε συγκεκριμένες ουρές βάσει κλειδιών δρομολόγησης και χαρακτηριστικών μηνυμάτων.*
- *Ως μια ευρέως διαδεδομένη λύση, το RabbitMQ διαθέτει εκτενέστατο documentation και υποστήριξη για πολλαπλά πρωτόκολλα μηνυμάτων πέρα από το AMQP, συμπεριλαμβανομένων των STOMP και MQTT, επιτρέποντας ευρείες δυνατότητες ενσωμάτωσης.*
- *Το RabbitMQ παρέχει εγγυήσεις για την παράδοση μεμονωμένων μηνυμάτων, συμπεριλαμβανομένων των επιβεβαιώσεων μηνυμάτων (επιβεβαιώσεις producer, επιβεβαιώσεις consumer) και των persistent μηνυμάτων, διασφαλίζοντας ότι τα μηνύματα δεν χάνονται ακόμη και σε περίπτωση αποτυχίας του broker. Αυτό το καθιστά κατάλληλο για ουρές εργασιών όπου κάθε μήνυμα πρέπει να επεξεργαστεί και να μην χαθεί σε κάποιο downtime κάτω από μεγάλο φόρτο εργασίας.*
- *Το RabbitMQ είναι σχετικά απλό στην εγκατάσταση και διαχείριση. Επιπλέον, το περιβάλλον διαχείρισής του προσφέρει εύκολη παρακολούθηση των queue και διαμόρφωση αυτόν.*

2.5.3.2 Αρνητικά του RabbitMQ

Ταυτόχρονα όμως υπάρχουν και οι αρνητικές πτυχές του RabbitMQ που θα αναλυθούν παρακάτω.

- *Ενώ μπορεί να γίνει ευκολά scalable, το RabbitMQ επειδή πρέπει να κάνει acknowledge κάθε ένα μήνυμα ώστε να σιγουρευτεί ότι δεν θα χαθεί κάποιο είναι εύκολο να δημιουργηθεί bottleneck σε εκείνο το σημείο όταν πρέπει να γίνει επεξεργασία πολλών χιλιάδων ή εκατομμύριων μηνυμάτων το δευτερόλεπτο.*
- *Επειδή το RabbitMQ είναι από την φύση του εφήμερο, δεν κρατάει κάποιο ιστορικό ενεργειών, εάν ο consumer αποτύχει να επεξεργαστεί τα μηνύματα αφού τα έχει παραλάβει από τον Message Broker δεν υπάρχει κάποιος τρόπος διορθώσεις της κατάστασης.*
- *Για πολύ μεγάλους αριθμούς ουρών και ταυτόχρονων συνδέσεων, το RabbitMQ μπορεί να καταναλώσει σημαντικούς πόρους μνήμης και CPU*

2.5.4 Apache Kafka

Το Apache Kafka είναι μια κατανεμημένη πλατφόρμα ροής σχεδιασμένη για τη δημιουργία data pipeline πραγματικού χρόνου και streaming υπηρεσίες. Σε αντίθεση με τις παραδοσιακές ουρές μηνυμάτων, το Kafka λειτουργεί ως ένα κατανεμημένο Committee Logic, αντιμετωπίζοντας τα μηνύματα ως αμετάβλητα αρχεία που προσαρτώνται σε θέματα (topics) σε πραγματικό χρόνο.

2.5.4.1 Πλεονεκτήματα του Apache Kafka

Κάποια από τα θετικά χαρακτηριστικά του είναι τα εξής:

- *Το Kafka έχει σχεδιαστεί για εξαιρετικά υψηλή απόδοση και οριζόντια επεκτασιμότητα, ικανό να χειριστεί εκατομμύρια μηνύματα ανά δευτερόλεπτο σε ένα Cluster5. Η αρχιτεκτονική του που βασίζεται σε διαμερίσματα (partitions) επιτρέπει μαζική παράλληλη επεξεργασία ροών δεδομένων.*
- *Τα μηνύματα στο Kafka αποθηκεύονται μόνιμα στο δίσκο για κάποια περίοδο, επιτρέποντας την ανθεκτική αποθήκευση ροών συμβάντων. Η κατανεμημένη και αναπαραγόμενη αρχιτεκτονική καταγραφής του παρέχει ανοχή σφαλμάτων, καθιστώντας το εξαιρετικά ανθεκτικό σε αστοχίες κόμβων σε αντίθεση με το RabbitMQ που είναι πιο επιρρεπής σε τέτοια σφάλματα.*
- *Το μοντέλο του Kafka που βασίζεται σε Logic επιτρέπει στους καταναλωτές να διαβάζουν από οποιοδήποτε σημείο στην ιστορία ενός topic (offsets), επιτρέποντας δυνατότητες όπως το event sourcing, η επεξεργασία ροών και η επανάληψη ιστορικών δεδομένων για αναλυτικούς σκοπούς ή εντοπισμό σφαλμάτων.*
- *Το Kafka διαθέτει ένα πλούσιο και ταχέως εξελισσόμενο ecosystem, συμπεριλαμβανομένων των Kafka Streams για τη δημιουργία εφαρμογών επεξεργασίας ροής, του Kafka Connect για ενσωμάτωση με άλλα συστήματα και του KSQL για ερωτήματα που μοιάζουν με SQL σε ροές.*

2.5.4.2 Αρνητικά του Apache Kafka

Ταυτόχρονα όμως υπάρχουν και τα αρνητικά τμήματα του Apache Kafka.

- *Πολυπλοκότητα όταν έχει να γίνει απλή ανταλλαγή μηνυμάτων. Όταν απλά υπάρχει επικοινωνία point-to-point ή απλές ουρές εργασιών το Kafka εισάγει περιττή λειτουργική πολυπλοκότητα λόγω της κατανεμημένες αρχιτεκτονικής που ακολουθεί.*
- *Η σειρά τον μηνυμάτων μπορεί να είναι διαθέσιμη μόνο σε ένα τμήμα (partition)*
- *Ενώ το Kafka υπερέχει στην συνολική απόδοση, η καθυστέρηση για μεμονωμένα μηνύματα, ειδικά όταν περιμένει επιβεβαιώσεις, μπορεί μερικές φορές να είναι υψηλότερη από ό,τι στο RabbitMQ για σενάρια πολύ χαμηλής καθυστέρησης, "fide-and-former".*

2.5.5 Επιλογή μεταξύ Apache Kafka και RabbitMQ

Για την εργασία προτιμήθηκε η χρήση του RabbitMQ λόγω της υποστήριξης που παρέχει το ίδιο το Spring Boot. Επιπρόσθετα επιλέχθηκε καθώς μιας και τα μηνύματα είναι single event τα οποία μπορεί να παραδοθούν εν τελεί ή όχι, στην περίπτωση που δεν παραδοθούν μπορούν απλά να ξανά δημιουργηθούν μέσα σε κάποιο προκαθορισμένο χρονικό οροί που θέτει ο producer μας. Επίσης πολύ σημαντική είναι και αξιοποίηση των ουρών που χρησιμοποιεί το RabbitMQ καθώς μπορούμε δυναμικά να φτιάξουμε μια ουρά για κάθε έναν server producer μας που υπάρχει και δυναμικά να μπορεί ο executor μας να διαβάζει αυτές από το RabbitMQ και να αποστέλλει το κάθε μήνυμα στην σωστή ουρά[15].

2.6 Version Control System (VCS)

2.6.1 Εισαγωγή

Για την οργάνωση, την αποθήκευση και την τήρηση ιστορικού του κώδικα χρησιμοποιούνται συστήματα Version Control (VCS).Με την χρήση τέτοιον συστημάτων δίνεται η δυνατότητα πολύ προγραμματιστές να μπορούν να συνεργαστούν στην ιδιά βάση κώδικα, να τηρείτε ιστορικό αλλαγών καθώς και να μπορεί κάποιος να γυρίσει ευκολά σε μια παλιότερη έκδοση.

2.6.2 Διαθέσιμα VCS συστήματα

Υπάρχουν αρκετά εργαλεία που μας δίνουν τέτοιες δυνατότητες όπως για παράδειγμα το Git και το Subversion (SVN). Παρακάτω θα αναλυθούν αυτά τα συστήματα καθώς και τα υπερ. και τα κατά τους [16].

2.6.3 Git

Το git[17] αποτελεί ένα από τα πιο σημαντικά και γνωστά VCS. Αναπτυγμένο αρχικά για χρήση του στον πυρήνα τον Linux αποτελεί μια αρκετά αποδοτική λύση η οποία υποστηρίζει branches καθώς και merging αυτόν μεταξύ τους. Αποτελεί ένα κατανεμημένο σύστημα ελέγχου έκδοσης (DVCS) το οποίο βασίζεται στην λογική του commit-push-pull-update. Αυτό σημαίνει ότι για να γίνει μια αλλαγή κάποιος πρέπει να συνθέσει ένα commit, να το κάνει push στο remote και για να μπορεί κάποιος να δει την νέα αλλαγή θα πρέπει να κάνουν pull τις αλλαγές από το remote και μετρά να μπορούν να κάνουν αλλαγές επιώ αυτού. Το git αποτελεί την βάση για συστήματα όπως το GitHub, GitLab και το Bitbuckets.

2.6.4 SVN

Το subversion (svn)[18] αποτελεί ένα κεντρικό σύστημα ελέγχου εκδόσεων (CVCS). Το SVN δεν χρησιμοποιήσετε τόσο συχνά από προτζεκτ ανοιχτού κώδικα αλλά κύριος από επιχρίσεις λόγω της απλότητάς που προσφέρει. Η βασική διαφορά του με το git αποτελεί το ότι υπάρχει μόνο ένα κεντρικό repository του κώδικα. Αυτό έχει ως σκοπό το να χρειάζεται συνεχόμενα σύνδεση πάνω σε αυτό το repository ώστε να μπορούν να γίνουν commit οι αλλαγές, σε αντίθεση με το git που μπορεί να δουλέψει και offline σε τοπικό επίπεδο. Επίσης πολύ διαφορετικός είναι ο τρόπος που οργανώνονται τα branches σχετικά με το git. Ενώ τα branches στο git είναι pointers σε κάποιο commit, το sun ορίζει τα branches ως φακέλους με την κατάληξη .svn μέσα στο κεντρικό filesystem. Η γενική δομή και ιδέα του παρόλα αυτά είναι πολύ πιο απλή σε σχέση με τον τρόπο που λειτουργεί το git, πράγμα που το κάνει φιλικό προς νέους χρήστες σε αντίθεση με το git το οποίο έχει πολύ μεγαλύτερο επίπεδο δυσκολίας.

2.6.5 Επιλογή Git

Για την πτυχιακή εργασία χρησιμοποιήθηκε το git για την οργάνωση του κώδικα καθώς και την εύκολη συνεργασία μεταξύ των μελών. Ποιο συγκεκριμένα, χρησιμοποιήθηκε το GitLab ένα σύστημα git ανοιχτού κώδικα αντί για το GitHub. Και τα δυο συστήματα αποτελούν μια πάρα πολύ καλή λύση καθώς προσφέρουν τις ιδίες λειτουργίες. Κάποια από τα βασικά πράγματα είναι η εύκολη δημιουργία Issues που μπορούν να εκφράσουν μέσω ενός αρχείου .md το τι πρέπει να γίνει (πχ περιγραφή ενός προβλήματος που παρατηρήθηκε, περιγραφή ενός σφάλματος κλπ.), την δημιουργία Merge Request ή αλλιώς Pull Request για την ενσωμάτωση του κώδικα από ένα branch σε ένα άλλο καθώς και την ύπαρξη του GitLab Runner[19] και του GitHub Actions[20] που δίνει την δυνατότητα με την χρήση συγκεκριμένου συντακτικού να υποστηριχτεί CI/CD.

Ο τρόπος που δομήσαμε την ροή εργασίας μας ήταν ο εξής:

- *Ύπαρξη Issues για το τι θέλαμε να φτιάξουμε.*
- *Δυο ξεχωριστά repositories, ένα για το frontend κομμάτι και ένα για το backend.*
- *Κάθε νέο feature ή bug βρισκόταν στο δικό του branch*
- *Επιπρόσθετα από αυτά τα branch υπήρχαν και τα:*
- *Development: Εδώ υπήρχαν features που ακόμα δεν έχουν γίνει release ακόμα. Πολλές φορές θα πρέπει να μαζέψουμε αρκετά features για να μπορούν να περάσουν στο επόμενο στάδιο της έκδοσης.*

- *Main: Αποθηκεύεται ο stable κώδικας με κάθε release του κώδικα. Δεν γίνονται απευθείας commit πράγματα από κάποιο feature branch σε αυτό αλλά μόνο από το develop.*
- *Και τα δυο αυτά branch αποτελούν protected branches που σημαίνει ότι δεν μπορούν να σβηστούν και δεν έχουν όλοι πρόσβαση να κάνουν commit σε αυτά.*

2.6.6 Χρήση GitLab Docker Image Registry

Τέλος, χρησιμοποιήθηκε και το docker registry που προσφέρει το GitLab για να μπορούμε να αποθηκεύομε πάνω τα Docker images τα οποία δημιουργούμε ώστε να μπορούμε να δώσουμε την δυνατότητα κάποιος να μπορεί να τα κατεβάσει και να τα χρησιμοποιήσει.

2.7 Αρχιτεκτονική Back-end εφαρμογής

2.7.1 Εισαγωγή

Λόγω του μεγέθους και της πολυπλοκότητας της πτυχιακής εργασίας καθώς και ότι το back end αποτελείται από 3 τμήματα, ο κώδικας πρέπει να είναι οργανωμένος με τον ίδιο τρόπο παντού. Για αυτό το λόγο έπρεπε από νωρίς να αποφασιστεί με τη αρχιτεκτονική θα δομηθεί η εργασία.

2.7.2 Προβλήματα αναλογων υπηρεσιων

Καθοριστικό παράγοντα στην επιλογή αρχιτεκτονικής αποτέλεσε η αδυναμία που είχαμε παρατηρήσει σε ανάλογα συστήματα τα οποία ήταν μονολιθικά δηλαδή υπήρχε μια εφαρμογή για το σύνολο των ενεργειών που κάνει αυτή. Σε αυτήν την περίπτωση παρατηρούνταν ότι όταν υπήρχε μεγάλος όγκος δεδομένων που έπρεπε να σταλθούν από την εφαρμογή προς τον mail server λειτουργίες όπως τα στατιστικά καθυστερούσαν να δώσουν απάντηση πίσω στην εφαρμογή. Για αυτό το λόγο αποφασίστηκε να δομηθεί η εφαρμογή ως μικρό-υπηρεσίες (microservices). Με αυτό τον τρόπο η εφαρμογή μπορεί να αποτελείται από πολλές περισσότερες μικρές εφαρμογές οι οποίες επικοινωνούν μεταξύ τους άλλα τρέχουν διαφορετικά και δεν επηρεάζει είναι η εκτέλεση της μιας στην άλλη.

Με αυτόν τον τρόπο μας μένουν 2 βασικές αρχιτεκτονικές που θα μπορούσαμε να οργανώσουμε τον κώδικά μας Την πολυεπίπεδη αρχιτεκτονική και την αρχιτεκτονική εξάγωνου. Παρακάτω θα αναλυθούν αυτές οι δυο.

2.7.3 Πολυεπίπεδη αρχιτεκτονική

Στην πολυεπίπεδη αρχιτεκτονική (layered architecture)[21] ο κώδικας οργανώνεται μεταξύ 3 βασικών επιπέδων. Τα επίπεδα αυτά είναι: οι Controller, τα Services και τα Repositories.

- *Στο επίπεδο των Controller υπάρχει η λογική η οποία είναι υπεύθυνη για το πως επικοινωνεί η εφαρμογή με άλλες εφαρμογές. Οι controller στην περίπτωση αυτή εμπεριέχουν την λογική του REST API καθώς και τις λήψης μηνυμάτων από το message broker (RabbitMQ). Εμπεριέχουν αυστηρά την ελάχιστη δυνατή λογική μέσα τους όπου αυτή περιορίζεται συνήθως στο να καλέσει κάποιο Service και μετά να στείλει με σωστό τρόπο την απάντηση πίσω.*
- *Στο δεύτερο επίπεδο υπάρχουν τα Services. Τα Services αποτελούν τον εγκέφαλο της εφαρμογής καθώς αυτά είναι υπεύθυνα για οποιαδήποτε ενέργεια πρέπει να γίνει πάνω στα δεδομένα που έχουν ληφθεί από τους Controllers καθώς και για το τι και πως θα αποθηκευτεί ή να ανακτηθεί από το επόμενο επίπεδο που είναι το Repositories. Για να μπορεί να ακολουθηθούν οι αρχές του SOLID τα services κρύβουν τις λεπτομερείς λειτουργεία τους με την χρήση interfaces και implementations αυτών των interfaces. Έτσι για παράδειγμα υπάρχει το UserService το οποίο δίνει δυνατότητες όπως πχ registerUser ή changeUserPassword ενώ η λογική και οι ενέργειες που η κάθε συνάρτηση κάνει βρίσκονται μέσα στο UserServiceImpl το οποίο εφαρμόζει το*

UserService. Ο Controller δεν χρειάζεται να ξέρει το τι υπάρχει μέσα στο UserServiceImpl πάρα μόνο το πως είναι δομημένο το συμβόλαιο που χαρακτηρίζεται από το interface UserService.

- *Τέλος στο τελευταίο επίπεδο υπάρχει το Repository. Στο συγκεκριμένο επίπεδο είναι συγκεντρωμένη όλη η λογική η οποία έχει να κάνει συνήθως με αποθήκευση δεδομένων σε κάποια βάση δεδομένων ή με την αποστολή δεδομένων προς κάποιον message broker.*
- *Η συγκεκριμένη αρχιτεκτονική έχει το καλό ότι κάθε επίπεδο χρειάζεται να γνωρίζει μόνο το επίπεδο κάτω από αυτό και δεν τον ενδιαφέρει για το επίπεδο από πάνω του. Ταυτόχρονα στην layer αρχιτεκτονική συναντιούνται δύο ακόμα πακέτα τα οποία εμπεριέχουν τις οντότητες (entities) καθώς και τα αντικείμενα μεταφοράς τομέα (Domain Transfer Object). Με αυτό τον τρόπο κάθε τμήμα της εφαρμογής βλέπει μόνο τα δεδομένα που χρειάζεται να ξέρει χωρίς να υπάρχει κίνδυνος λόγω αλλαγής ενός πεδίου να εκτεθεί πληροφορία που δεν πρέπει στον τελικό χρήστη. Το επίπεδο του Controller χρησιμοποιεί αυστηρά μόνο τα domain transfer object, το επίπεδο του repository τα entities ενώ το επίπεδο των services είναι υπεύθυνο για την μετατροπή του ενός στο άλλο ανάλογα με το σε πιο επίπεδο πρέπει να πάνε τα δεδομένα. Αυτό βοηθάει για παράδειγμα στην περίπτωση της οντότητας του User, για να γίνει εγγραφή ενός χρηστεί χρειάζεται το username, password και το email. Αυτά τα δεδομένα τα λαμβάνει ο controller τα δίνει στο ανάλογο service, τα μετατρέπει σε entity και προσθέτει κάποια πεδία τα οποία θα αποθηκευτούν στην βάση δεδομένων που τα χρειάζεται η εφαρμογή όπως το ποτέ έγινε η εγράφη του χρήστη, πότε έκανε τελευταία φορά σύνδεση στο σύστημα, χωρά εγράφης κλπ. Τα παραπάνω αποτελούν πληροφορία που δεν χρειάζεται και πολλές φορές δεν πρέπει να βγει εκτός εφαρμογής όπως πχ το αποτέλεσμα μιας συνάρτησης (hash) του password που είναι αποθηκευμένο στην βάση και χρησιμοποιείτε για επαλήθευση του χρήστη.*

Αυτό μαζί με το γεγονός ότι είναι ευκολά το να διαχωριστεί το Business Logic από το υπολιπόν κομμάτι της εφαρμογής το κάνει ιδανικό για την χρήση σε αυτή την εργασία αν και συνήθως συναντάνε σε μονολιθικές εφαρμογές.

2.7.4 Αρχιτεκτονική εξάγωνου

Ένας εναλλακτικός τρόπος δόμησης της αρχιτεκτονικής αποτελεί η εξάγωνη αρχιτεκτονική (hexagonal architecture)[22]. Σε αυτή την περίπτωση η κεντρική ιδέα είναι ότι η εφαρμογή χτίζεται με βάση κυρίως την επιχειρησιακή λογική (core business logic) και γύρο της προστατεύεται από εξωτερικής τεχνολογίες όπως βάσεις δεδομένων, HTTP interfaces, Rest Apis.

Σε αντίθεση με την αρχιτεκτονική σε επίπεδα δεν έχει controllers, services και repositories αλλά ports, adapters και το domain.

- *Το domain είναι υπεύθυνο για το τι κάνει η εφαρμογή, δηλαδή τους κανόνες, τις οντότητες καθώς και κρατάει την επιχειρησιακή λογική.*
- *Το ports αποτελούν το κομμάτι που εμπεριέχει τα interfaces που εξηγούν τι χρειάζεται το core όπως για παράδειγμα η εγγραφή ενός χρήστη.*
- *Το adapters αποτελεί την εφαρμογή τον ports, δηλαδή εμπεριέχει την λογική του με ποιον τρόπο θα γίνουν πραγματικά οι ενεργείς που χαρακτηρίζονται στα interfaces.*

Με αυτό τον τρόπο κάθε τμήμα της εφαρμογής είναι πραγματικά ανεξάρτητο από τα υπόληπτα και κάτ. επέκταση εύκολο στο να αντικατασταθεί από κάποιο άλλο. Για παράδειγμα είναι αρκετά εύκολο να αλλάξει το κομμάτι που επικοινωνεί με το RabbitMQ με κάποιο άλλο το οποίο θα στέλνει πλέον τα δεδομένα με REST API. Παράλληλα, προσφέρει πολύ πιο εύκολο έλεγχο (unit testing) δεδομένου ότι κάθε τμήμα είναι αυτόνομο αρά και μπορεί να ελεγχθεί αυτόνομα και να εντοπιστούν προβλήματα από νωρίς. Αφού το domain είναι ανεξάρτητο από τα υπόλοιπα τμήματα αυτό δίνει την δυνατότητα να μπορεί να είναι ανεξάρτητο από τις επιμέρους τεχνολογίες που χρησιμοποιούνται στην εφαρμογή καθώς και το τμήμα αυτό κρατάει αυστηρά και μόνο καθαρή την επιχειρησιακή λογική.

Η αρχιτεκτονική εξάγωνου είναι φτιαγμένη κύριος για συστήματα με microservices λόγω αυτών τον ευελιξιών που δίνει στους προγραμματιστές.

2.7.5 Επιλογή αρχιτεκτονικής

Η εφαρμογή έχει κάνει χρήση της επίπεδης αρχιτεκτονικής και οργανώνει τον κώδικα βάση αυτής. Η απόφαση αυτή παρθικέ καθώς το spring boot αποτελεί μια τεχνολογία που είναι βασισμένη σε αυτή την αρχιτεκτονική αρά και είναι πιο εύκολη και πιο γρήγορη η υλοποίηση της. Παρόλα αυτά, η εφαρμογή θα μπορούσε αρκετά ευκολά να αλλάξει αρχιτεκτονική και να υλοποίηση την εξάγωνη αρχιτεκτονική.

2.8 Συστήμα αξιολόγησης ηλεκτρονικών μηνυμάτων

2.8.1 SpamAssassin

Για την αξιολόγηση των ηλεκτρονικών μηνυμάτων χρησιμοποιήθηκε το SpamAssassin. Το SpamAssassin[23] αποτελεί ένα εργαλείο ανοιχτού κώδικα, από την Apache, το οποίο βοηθάει στο φιλτράρισμα και την αξιολόγηση ανεπιθύμητων email. Λειτουργεί εφαρμόζοντας μια σειρά από τεστ και ελέγχους κανόνων πάνω σε ένα email και αξιολογεί αναλογώ με ένα τελικό σκορ, εάν αυτός είναι κάτω από ένα προκαθορισμένο όριο το μήνυμα χαρακτηρίζεται ως spam και υπο κανονικές συνθήκες προσθέτει ένα ειδικό header για να μετακινηθεί στον φάκελο spam. Στην περίπτωση αυτής της εργασίας το SpamAssassin λαμβάνει ένα μήνυμα και επιστρέφει στο backend το score καθώς και λεπτομερείς για ποιο λόγω βγήκε αυτό το score.

2.8.2 Πως λειτουργεί το SpamAssassin

Η βασική αρχή λειτουργίας του SpamAssassin είναι η βαθμολόγηση κάθε email με βάση μια σειρά από διαφορετικές τεχνικές και τεστ. Τα βασικά συστατικά και χαρακτηριστικά του είναι τα εξής:

- *Φίλτρο Bayesian: Το SpamAssassin μπορεί να μάθει να αναγνωρίζει spam και μη mail (ham) με βάση προηγουμένη εμπειρίας του. Χρησιμοποιώντας στατιστική ανάλυση με πιθανότητες Bayesian, αξιολογεί την πιθανότητα να είναι ένα μήνυμα spam συγκρίνοντας το περιεχόμενο και την δομή των email με μηνύματα που έχουν μαρκαριστεί ως spam παλιότερα.*
- *DNS-based: Βάση της διεύθυνσης IP του αποστολέα μπορεί να ελέγξει εάν ο αποστολέας βρίσκεται σε γνωστές λίστες με spam servers.*
- *Έλεγχος SPF, DKIM και DMARC: Μπορεί να ελέγξει εάν ο αποστολέας είναι εξουσιοδοτημένος από το domain να στείλει email.*
- *Συνδυασμός των παραπάνω.*

Στην περίπτωση της εργασίας μας κυρίως αξιοποιήσαμε το σύστημα κανόνων καθώς και το φίλτρο Bayesian. Έχοντας θέση το SpamAssassin σε κατάσταση εκμάθησης σε συνεργασία με το μοντέλο μηχανικής μάθησης η εφαρμογή μπορεί όσο περνάει ο καιρός με το πλήθος των δεδομένων να δίνει καλύτερες προβλέψεις και συνολικές βαθμολογίες.

2.9 Τεχνολογίες frontend

2.9.1 Εισαγωγή

Για την δημιουργία της διεσπάρης του χρήστη υπάρχουν διάφορες τεχνολογίες που μπορούν να χρησιμοποιηθούν. Σε αυτό το τμήμα θα αναλυθούν κάποιες από τις διαδεδομένες τεχνολογίες που υπάρχουν, ποια επιλέχθηκε και για ποιο λόγο.

2.9.2 React

Ένα από τα πιο γνωστά library για την ανάπτυξη του frontend αποτελεί η React[24]. Αναπτύσσεται από την Meta και είναι φτιαγμένο να δουλεύει με TypeScript ή/και JavaScript. Η βασική ιδέα του βασίζεται

στην δημιουργία component τα οποία είναι κομμάτια κώδικα επαναχρησιμοποιούμενα ειδικού σκοπού. Χρησιμοποιεί το virtual DOM, δηλαδή την αναπαράσταση του DOM δέντρου στην μνήμη οπού το κάθε element αποτελεί ένα κόμβο. Με αυτό τον τρόπο η React μπορεί να κάνει γρήγορες αλλαγές στο UI κομμάτι της εφαρμογής χωρίς να χρειάζεται να τροποποιεί το DOM. Επίσης, πολύ θετικό αποτελεί το γεγονός της ύπαρξης μεγάλης υποστήριξης από την κοινότητα από βιβλιοθήκες. Κάποια από τα όρια της React όμως αποτελούν το ότι δεν είναι ένα ολοκληρωμένο framework αλλά βιβλιοθήκη, αυτό έχει σαν αποτέλεσμα να χρειάζεται επιπρόσθετες βιβλιοθήκες για την υποστήριξη παραπάνω ιδιοτήτων όπως για παράδειγμα routing, διατήρηση form καθώς και για την διαχείριση του state.

2.9.3 Angular

Η Angular[25] αποτελεί ένα αρκετά εκτενές και δογματικό front-end framework από την Google. Αποτελεί μια εκδοχή της αρχικής AngularJS φτιαγμένη από την αρχή με την χρήση TypeScript για χρήση σε εφαρμογές μεγάλης κλίμακας.

Σε αντίθεση με την React που είναι component base, η Angular χρησιμοποιεί την MVVM (Model-View-View Model) αρχιτεκτονική βασιζομένη στο dependency injection, τα δηλωτικά πρότυπα καθώς και την ασύγχρονη εκδοχή της JS την RxJS. Επίσης υποστηρίζει το bidirectional binding το οποίο σημαίνει ότι το κομμάτι του μοντέλου μπορεί να λάβει και να δώσει δεδομένα στο επίπεδο του view, Αυτό μειώνει τον κώδικα που πρέπει να επαναλαμβάνεται μέσα στην εφαρμογή (boilerplate code).

Κάποια από τα θετικά στοιχεία της είναι:

- *Αποτελεί μια πλήρης λύση. Δεδομένου ότι είναι ένα framework και όχι μια βιβλιοθήκη προσφέρει στον προγραμματιστή πολλά παραπάνω διαθέσιμα εργαλεία χωρίς να χρειάζεται να προσθέσει κάτι παραπάνω για βασικές λειτουργίες.*
- *Κλιμακωσιμότητα. Η modular αρχιτεκτονική του καθώς και ο περιορισμός στα σφάλματα που προσφέρει το strong typing το κάνει κατάλληλο για enterprise εφαρμογές.*

Λόγω της ευελιξίας που δίνει στον προγραμματιστή βέβαια αποτελεί ένα δύσκολο framework στην κατανόηση του και του πως να δομηθεί σωστά. Αυτό έχει σαν αποτέλεσμα κάποιος ο οποίος είναι νέος στην χρήση του μπορεί να κάνει λάθη τα οποία να μετριάσουν την αποτελεσματικότητα του.

2.9.4 Vue

Η Vue[26] αποτελεί ένα JavaScript/TypeScript framework για την κατασκευή διεπαφών. Δημιουργήθηκε από τον Evan You το 2014 και πλέον βρίσκεται στην έκδοση 3. Αποτελεί ένα αρκετά εύχρηστο και ευέλικτο framework το οποίο δίνει την δυνατότητα και σταδιακό integration του σε υπάρχουσες εφαρμογές.

Η Vue βασίζεται σε ένα reactive data-binding σύστημα σε συνδυασμό με το Virtual Dom. Η βασική του ιδέα βασίζεται στα Components, όπως και η react, τα οποία μπορούν να είναι είτε JavaScript είτε με ειδική σύνταξη που παρέχει η Vue σε αρχεία με κατάληξη .vue τα οποία περιλαμβάνουν την HTML, τα CSS καθώς και την JavaScript/Typescript σε ένα αρχείο. Ακολουθεί την αρχιτεκτονική MVVM (Model-View-ViewModel) όπως η Angular αλλά χωρίς να απαιτείται χειρισμός του DOM μιας και υποστηρίζει virtual DOM.

Κάποια από τα πλεονεκτήματα της είναι τα εξής:

- *Εύκολη εκμάθηση*
- *Καθαρή και κατανοητή σύνταξη με την χρήση single file component*
- *Προοδευτική υιοθέτηση σε περίπτωση μετάπτωσης από ένα framework/library στην vue*
- *Πολύ καλή και πλήρης τεκμηρίωση.*

Περιορισμοί της Vue:

- *Περιορισμένη υποστήριξη σε μεγάλες εφαρμογές σε αντίθεση με Angular ή React.*
- *Μεγάλη διαφορά μεταξύ εκδόσεων (η έκδοση 2 δεν είναι συμβατή με την 3)*

Μια διαφορά μεταξύ των framework/library όπως η vue με την react αποτελεί με ποιον τρόπο και πως λειτουργούν τα components. Τα components αποτελούν μια αυτόνομη και επαναχρησιμοποιούμενη μονάδα διεπαφής που εμπεριέχει HTML, CSS καθώς και την λογική συνήθως σε JavaScript ή όπως στην περίπτωση μας TypeScript. Αυτά χωρίζονται σε δυο μεγάλες κατηγορίες τα functional components και τα class components.

Τα functional components είναι απλές συναρτήσεις που επιστρέφουν μια αναπαράσταση της διεπαφής. Έχουν απλή σύνταξη, και λόγω το ότι είναι συναρτήσεις είναι εκ φύσεως πιο ελαφριά. Ένα ακόμα θετικό αποτελεί το γεγονός ότι επειδή δεν είναι κλάση και δεν διαχειρίζεται αντικείμενα (όπως στον αντικειμενοστραφή προγραμματισμό) δεν έχει state δηλαδή εσωτερική κατάσταση. Τα συγκεκριμένα ενδείκνυται κυρίως για απεικόνιση δεδομένων, πράγμα που είναι κατάλληλο για την χρήση σε αυτή την εργασία.

Τα class components χρησιμοποιούν την ιεραρχία τον κλάσεων της JavaScript και της TypeScript. Είναι πιο βαριά και χρησιμοποιούνται για να ορίσουν functions που θα έχουν την πλήρης λογική. Αυτά σε αντίθεση με τα functional components υποστηρίζουν state καθώς και lifecycle, δηλαδή γνωρίζουν ανά πάσα στιγμή την κατάσταση της εφαρμογής και όταν κάτι αλλάξει πράττουν αυτόματα ενεργείς που έχουν οριστεί.

Η Vue είναι βασισμένη αυστηρά σε functional components[27] και δεν δίνει την δυνατότητα επιλογής μεταξύ των δυο. Σε αντίθεση με την React που υποστηρίζει και τα δυο καθώς είναι backwards compatible με τις παλαιότερες έκδοσης της που ήταν βασισμένη κυρίως στα class components. Λόγω της χρήσης της εφαρμογής, του σχεδιασμού της καθώς και της απλότητας που προσφέρουν τα functional components κρίθηκε ότι δεν υπάρχει λογίως για την ύπαρξη class components. Αυτή η απόφαση ενισχύθηκε περαιτέρω από την προσπάθεια της react στις τελευταίες έκδοσης της να αποσύρει, με αργούς βέβαια ρυθμός, την χρήση των class components.

2.9.5 Επιλογή framework

Για την χρήση της πτυχιακής εργασίας κρίθηκε σκόπιμο να χρησιμοποιηθεί η Vue ενάτη της React και της Angular. Αυτό έγινε καθώς χρειάστηκε ένα framework το οποίο θα ήταν εύκολο στην εκμάθηση του καθώς κανένας από τους δυο που εργαστήκαμε στην εργασία δεν είχε γνώση μέχρι πρότινος κάποιου αλλού framework. Αυτό έκανε την Vue εξαιρετικά ελκυστική σε συνάρτηση με την πλούσια τεκμηρίωση και παραδείγματα που προσφέρει. Η Angular αποτέλεσε μια ακόμα λύση αλλά λόγω της πολυπλοκότητας που φέρει σαν framework ήταν πιο δύσκολο να δικαιολογηθεί η επιλογή της.

2.9.6 Quasar

Με σκοπό ένα τελικά όμορφο και όσο γίνεται πιο απλό αποτέλεσμα μαζί με την Vue3 χρησιμοποιήθηκε επιπρόσθετα το framework με ονομα Quasar[28]. Η Quasar αποτελεί ένα framework το οποίο είναι χτισμένο πάνω στην Vue και προσφέρει την απλότητα της Vue μαζί με κάποια έτοιμα αλλά και παραμετροποίηση component που κάνουν την ανάπτυξη της διεπαφής πολύ πιο εύκολη και γρήγορη καθώς και την δυνατότητα να γίνει build για διαφορετικούς στόχους όπως Android, iOS καθώς και εφαρμογή electron.

Το Quasar προσφέρει έτοιμα component όπως κουμπιά, κάρτες, διάλογους, μπάρες εργαλείων, editors, πίνακες και αλλά. Αυτά είναι έτοιμα να χρησιμοποιηθούν κατευθείαν με πολύ εύκολη ενσωμάτωση στην εφαρμογή. Παράλληλα, με την χρήση του έχουμε απευθείας υποστήριξη για responsive design πράγμα που απλοποιεί την χρήση της εφαρμογής σε κινητές συσκευές και γενικότερα συσκευές με διαφορετική ανάλυση οθόνης από αυτήν που έγινε αρχικά η ανάπτυξη της. Παρέχει εύκολη μετάφραση μέσω της υποστηρίξεις i18n, δηλαδή μπορούμε να έχουμε παράλληλα Ελληνικά, Αγγλικά ή οποιαδήποτε άλλη γλώσσα μπορεί να μεταφραστεί το περιεχόμενο της εφαρμογής σε αυτή. Τέλος, με την χρήση του Quasar Layout System είναι πολύ εύκολο να δημιουργηθούν πολύπλοκες διατάξεις με ένα πολύ απλό και ευέλικτο τρόπο πράγμα που το κάνει ακόμα πιο ελκυστικό σε προγραμματιστές που δεν έχουν πολύ μεγάλη εμπειρία με την ανάπτυξη frontend εφαρμογών.

2.10 Ασφάλεια και αυθεντικοποίηση χρηστών

2.10.1 Εισαγωγη

Για την διασφάλιση της ασφάλειας της εφαρμογής μας, για την σωστή επικοινωνία μεταξύ του frontend με το backend καθώς και για την διασφάλιση προστασίας των δεδομένων των χρηστών πρέπει να επιλεχθεί ένα μέσω ασφάλειας και αποδείξεις της ταυτότητας του χρήστη. Για αυτόν τον σκοπό υπάρχουν δυο μεγάλες κατηγορίες η αυθεντικοποίηση μέσω συνεδρίας καθώς και η αυθεντικοποίηση χωρίς αυτήν. Στο κεφάλαιο αυτό θα αναλυθούν κάποια μέσα αυθεντικοποίησης, τα υπερ, τα κατά τους καθώς και κάποιες λεπτομερείς σχετικά με αυτά και στο τέλος ποιο χρησιμοποιήθηκε.

2.10.2 Αυθεντικοποίηση βάση συνεδρίας

Η αυθεντικοποίηση που βασίζεται στην συνέδρια αποτελεί τον πιο γνωστό και παραδοσιακό τρόπο για την αυθεντικοποίηση του χρήστη σε μια εφαρμογή[29]. Ο χρήστης αφού δώσει το όνομα χρήστη και τον κωδικό του μέσω μιας φόρμας σύνδεσης τα αποστέλλει στο backend. Το backend με την σειρά του εξακριβώνει την εγκυρότητα των στοιχείων και εάν αυτά αντιστοιχούν σε κάποιον εγγεγραμμένο χρήστη, όταν αυτό εξακριβωθεί δημιουργεί ένα session και το αποθηκεύει είτε στην μνήμη, είτε στην βάση δεδομένων. Έπειτα δημιουργεί ένα cookie με όνομα JSESSIONID το οποίο εμπεριέχει το session id του χρήστη. Το frontend αφού το λάβει θα πρέπει για κάθε επόμενο request που θα κάνει προς την υπηρεσία να το συμπεριλαμβάνει (αν και συνήθως γίνεται αυτόματα από τον browser). Αποτελεί έναν από τους πιο εύκολους τρόπους αυθεντικοποίησης. Βέβαια, ένα πολύ μεγάλο μειονέκτημα του αποτελεί το γεγονός ότι αφού μιας και πρέπει να αποθήκευση in-memory ή σε σπάνιες περιπτώσεις στην βάση κάποια δεδομένα που αφορούν το session αντενδείκνυνται για χρήση σε περιβάλλοντα με microservices εκτός εάν υπάρχει το λεγόμενο session replication. Επίσης, λόγω ότι εξαρτάται αυστηρά από τα cookies είναι ιδανικό για εφαρμογές όπως αυτή που διαθέτουν REST APIs.

2.10.3 Stateless αυθεντικοποίηση

Μια διαφορετική λύση στο πρόβλημα της αυθεντικοποίηση χρήστη έρχεται να δώσει η stateless αυθεντικοποίηση[29]. Σε αυτή την περίπτωση αντί να υπάρχει κάποιο cookie και το backend να κρατάει κάποιο session αποθηκευμένο στην μνήμη του μπορεί να χρησιμοποιεί ένα token το οποίο είναι μοναδικό και απευθύνεται σε έναν και μόνο χρήστη. Αυτό αποθηκεύεται στην βάση και έχει ορισμένο χρόνο λήξης (TTL). Η διαφορά επίσης βρίσκεται ότι σε αυτή την περίπτωση το frontend πρέπει να αποθήκευση μόνο του αυτή την πληροφορία σε κάποιο local Storage καθώς και να φροντίσει να το συμπεριλαμβάνει στο Authorization header του κάθε αιτήματος που κάνει στο

backend. Με αυτό τον τρόπο δεν απαιτείται κάποιο server side session και επειδή δεν χρειάζεται κάποιο feature του browser (cookies management) είναι ιδανικό για REST APIs.

Υπάρχουν διάφοροι τρόποι να δομηθεί αυτό το token που χαρακτηρίζει το stateless authentication. Ένας από τους πιο γνωστούς αποτελεί το JWT ή αλλιώς JSON Web Token[30]. Το JWT αποτελεί μια εξειδικευμένη μορφή token-based authentication, είναι ένα ψηφιακά υπογεγραμμένο token το οποίο περιέχει πληροφορίες για τον χρήστη που χρειάζεται. Αποτελείται από τρία μέρη τα οποία είναι κωδικοποιημένα σε μορφή Base64URL και χωρίζονται με τελείς (.).

- *Το πρώτο τμήμα αποτελεί τον header του token. Σε αυτό εμπεριέχεται ο τύπος του token (σε αυτή την περίπτωση το JWT) καθώς και ο αλγόριθμος κρυπτογράφησης που χρησιμοποιήθηκε όπως HMAC SHA256 ή RSA.*
- *Το δεύτερο τμήμα αποτελεί τα claims. Σε αυτό το τμήμα ο προγραμματιστής μπορεί να θέσει τα πεδία που θα θέλει να είναι προστασία από το frontend, όπως για παράδειγμα χάρης το username, το user tier, expiration date και αλλά. Τα claims χωρίζονται σε 3 υποκατηγορίες*
- *Τα public τα οποία είναι κάποια σταθερά claims τα οποία ορίζονται από το IANA JSON Web Token Registry και χρησιμοποιούνται μόνο για τους σκοπούς που περιγράφει το RFC[32].*
- *Η δεύτερη κατηγορία ονομάζεται private claims και αυτά αποτελούν πληροφορία που έχει συμφωνήσει ότι θα χρησιμοποιεί το frontend και το backend.*
- *Και η τρίτη κατηγορία κάποια προκαθορισμένα claims τα οποία δεν είναι αναγκαστικά στην χρήση τους αλλά παρέχουν κάποιες χρήσιμες πληροφορίες όπως το: is (issuer), exp (expiration time) και αλλά.*
- *Τέλος το τελευταίο τμήμα ονομάζεται Signature. Σε αυτό εμπεριέχεται η ψηφιακή υπογραφή του εγγράφου. Για την δημιουργία αυτού το κομματιού πρέπει να πάρουμε το τμήμα του κωδικοποιημένου header, το κωδικοποιημένο payload (τα claims), ένα private key (secret), τον αλγόριθμο που ορίσαμε στον header και να υπογραφθούν αυτά. Για παράδειγμα: HMACSHA256(base64UrlEncode(header) + "." + base64UrlEncode(payload), secret)*

2.10.4 Βήματα αυθεντικοποίησης του χρήστη με JWT

Η αυθεντικοποίηση πλέον του χρήστη γίνεται με τον εξής τρόπο στην περίπτωση του JWT. Ο χρήστης κάνει login και στέλνει τα δεδομένα του στο backend. Το backend αφού επαλήθευση την ταυτότητα του χρήστη του εκδίδει ένα JWT token που εμπεριέχει βασικές πληροφορίες στο κομμάτι των claims που αφορούν τον user και χρειάζεται το frontend. Επιπρόσθετα αποστέλλει μαζί με αυτό και εάν refresh JWT που σε περίπτωση λήξης του JWT token μπορεί το frontend να χρησιμοποιήσει για την έκδοση νέου JWT token χωρίς να χρειαστεί το backend να ξανά ελέγξει την ταυτότητα του χρήστη. Το frontend αποθηκεύει την πληροφορία αυτή στο local Storage, επαληθεύει την προέλευση του token και εξάγει τις χρήσιμες για αυτό πληροφορίες που εμπεριέχονται στα claims.

Για να κάνει κάποιο request πλέον το frontend μετρά την παραπάνω διαδικασία το μόνο που έχει να κάνει είναι περνάει στον Authorization header το Bearer ακολουθούμενο από το JWT Token. Όταν το backend λάβει αυτό το token επαληθεύει ότι το έχει εκδώσει το ίδιο και εξάγει τα στοιχεία του user που κάνει κάποιο request. Με αυτό τον τρόπο δεν χρειάζεται κάθε φορά το backend service να κάνει κάποια κλήση σε βάση δεδομένων με πιθανών χιλιάδες εγγραφές για να φέρει τα στοιχεία του χρήστη εξοικονομώντας πολύτιμο χρόνο και πόρους που αλλιώς θα χρειαζόντουσαν.

Επειδή όμως ακριβώς δεν αποθηκεύεται κάπου αυτό το JWT token η χρήση του εμπεριέχει μερικούς κινδύνους. Ένας από αυτούς αποτελεί το γεγονός ότι δεν είναι εύκολη η ακύρωση κάποιο από τα JWT token πριν την λήξη τους. Ένας τρόπος καταπολέμησης αυτού του προβλήματος αποτελεί το να κρατάει ο server ιστορικό για τα blacklisted tokens που ενώ αποτελεί ξανά πιο γρήγορη διαδικασία από την αυθεντικοποίηση βασισμένη σε sessions παραμένει μια δραστηριότητα που χρειάζεται επικοινωνία με

την βάση. Ταυτόχρονα η εφαρμογή του στον server αποτελεί πιο δύσκολη διαδικασία από το απλό stateful authentication.

Η ευκολία όμως που παρέχει στο frontend καθώς και η μη ανάγκη συνεχομένης κλήσης της βάσης δεδομένων για την αυθεντικοποίηση ενός αιτήματος από έναν χρήστη κάνει την λύση του JWT αρκετά ελκυστική για την χρήση της εφαρμογής. Καθώς η εφαρμογή έχει σημαντική και μεγάλη σε όγκο δεδομένων επικοινωνία με εξωτερικά συστήματα όπως βάσης δεδομένων, RabbitMQ και mail server το να μπορεί να γλυτώσει κάποιες επαναλαμβανόμενες κλήσεις στην βάση δεδομένων αποτελεί ένα πολύ σημαντικό πλεονέκτημα το οποίο δεν μπορεί να ισορροπιστή με κάποια άλλη μέθοδο.

2.10.5 Spring Security

Για τον server χρησιμοποιήθηκε το Spring Security[31] το οποίο δίνει κάποιο σταθερό κώδικα (boilerplate) για την αυθεντικοποίηση. Το springbok είναι σε θέση να κάνει ταυτοποίηση χρηστών, να ελέγχει δικαιώματα πρόσβασης, να προστατεύει URL από μη εξουσιοδοτημένα άτομα καθώς και να προστατεύει από γνωστές επίθεσης όπως CSRF (Cross site request forgery). Δυστυχώς ενώ έχει υποστήριξη για αυθεντικοποίηση με χρήση JWT αυτή δεν αποτελεί μια plug-and-play λύση. Για την υποστήριξη του έπρεπε να δημιουργηθούν services υπεύθυνα για την εξαγωγή και δημιουργία των claims από το payload, την ψηφιακή υπογραφή που παρουσιάζεται στο πεδίο signature καθώς και τον μηχανισμό παροχής refresh Token. Με την δημιουργία αυτών δόθηκε η δυνατότητα στο backend να υποστηρίξει αυθεντικοποίηση μέσω JWT.

Κεφάλαιο 3ο: Βάση δεδομένων

Η βάση δεδομένων που επιλέξαμε να χρησιμοποιήσουμε είναι η MySQL. Επιλέξαμε να χρησιμοποιήσουμε μια σχεσιακή βάση δεδομένων γιατί η φύση των δεδομένων της εφαρμογής είναι δομημένη, με ξεκάθαρες σχέσεις μεταξύ οντοτήτων όπως χρήστες, καμπάνιες, συνδρομητές και ομάδες. Επίσης η σχεσιακή προσέγγιση μας προσφέρει την δυνατότητα θέσουμε περιορισμούς στα δεδομένα (UNIQUE, NOT NULL, FOREIGN KEY) ώστε να έχουμε ένα επιπλέον επίπεδο ασφάλειας, όσον αφορά την ακεραιότητα των δεδομένων.

Όσον αφορά την MySQL, την επιλέξαμε διότι είναι μια ευρέως χρησιμοποιούμενη σχεσιακή βάση δεδομένων, η οποία προσφέρει σταθερότητα, καλή απόδοση και πλήρη υποστήριξη ACID συναλλαγών. Παρέχει συμβατότητα με το οικοσύστημα του Spring Boot και του Hibernate μέσω JDBC, διευκολύνοντας την ενσωμάτωσή της στο backend της εφαρμογής. Επιπλέον, διαθέτει ισχυρά χαρακτηριστικά όπως constraints και indexes, τα οποία είναι απαραίτητα για την ακεραιότητα και την αποδοτική αναζήτηση των δεδομένων. Τέλος, η χρήση της σε συνδυασμό με το εργαλείο Liquibase επιτρέπει τον ασφαλή και ελεγχόμενο χειρισμό των αλλαγών στο σχήμα της βάσης.

3.1 Διαχείριση του Σχήματος Βάσης με Liquibase

Το Liquibase[33] βασίζεται σε ένα αρχείο που ονομάζεται changelog. Αυτό το αρχείο περιέχει μια σειρά από changesets, τα οποία περιγράφουν βήμα προς βήμα τις αλλαγές που πρέπει να εφαρμοστούν στη βάση δεδομένων. Κάθε changeset αντιπροσωπεύει μία και μόνο μία αλλαγή στο schema, για παράδειγμα, τη δημιουργία ενός πίνακα, την προσθήκη ενός πεδίου ή τη δημιουργία ενός index. Κάθε changeset φέρει ένα μοναδικό αναγνωριστικό, το όνομα του συντάκτη, και το μονοπάτι του αρχείου από όπου προέρχεται. Αυτά τα τρία πεδία μαζί ορίζουν την ταυτότητα της αλλαγής και χρησιμοποιούνται για να διασφαλιστεί ότι η κάθε αλλαγή εφαρμόζεται μόνο μία φορά.

```
-- changeset a.tsantoulis:9 logicalFilePath:classpath:db/changelog.mysql.sql

ALTER TABLE `mail_server` MODIFY COLUMN `spam_threshold` DOUBLE NOT NULL;
```

Πίνακας 1: Liquibase Changeset

Επίσης το Liquibase χρησιμοποιεί δύο πίνακες στη βάση δεδομένων για να παρακολουθεί την κατάσταση των αλλαγών και να διαχειρίζεται την ασφαλή εκτέλεση τους, τους πίνακες **DATABASECHANGELOG** και **DATABASECHANGELOGLOCK**.

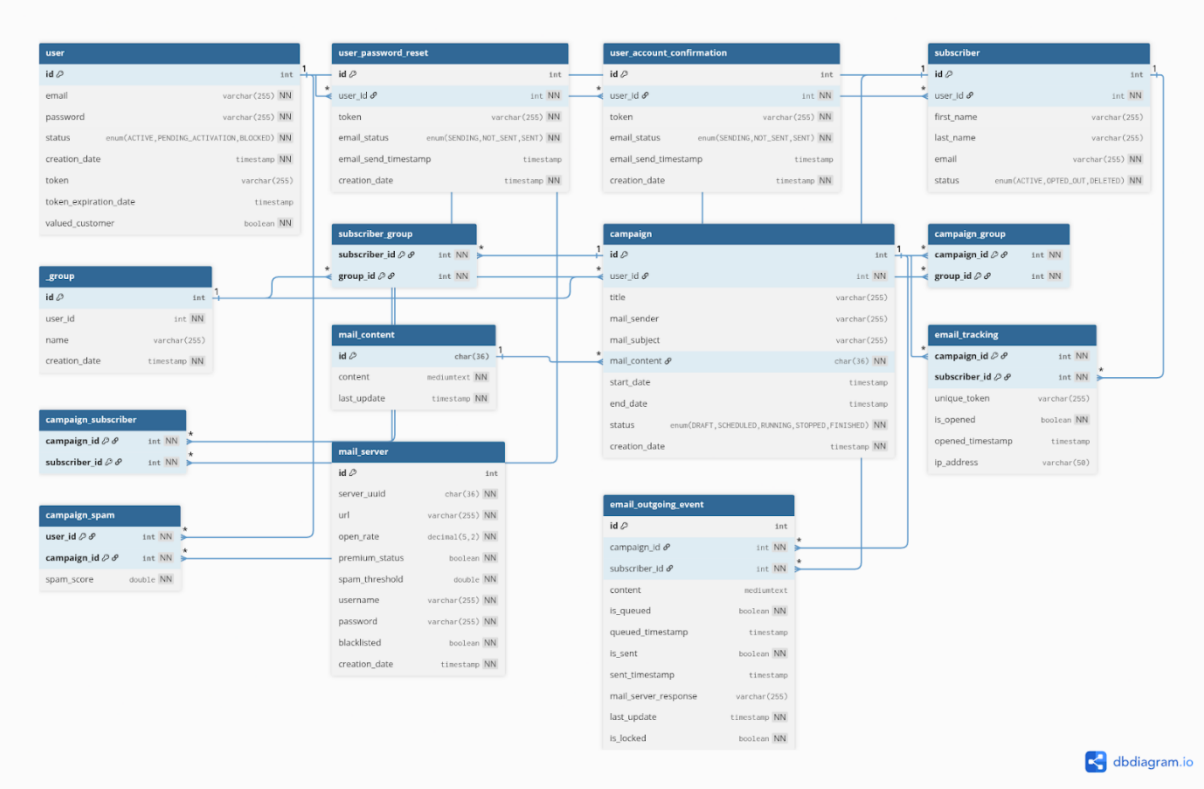
Ο πίνακας DATABASECHANGELOG καταγράφει κάθε changeset που έχει εκτελεστεί. Για κάθε εγγραφή αποθηκεύονται πληροφορίες όπως το ID, το AUTHOR, το FILENAME, την ημερομηνία εκτέλεσης, το checksum του changeset (MD5SUM) και η σειρά εκτέλεσης, Με βάση αυτά τα δεδομένα, το Liquibase μπορεί να αναγνωρίζει ποιες αλλαγές έχουν ήδη εφαρμοστεί και να αποφεύγει την επανεκτέλεση τους. Επιπλέον, το checksum χρησιμεύει στον εντοπισμό μη εξουσιοδοτημένων ή εσφαλμένων τροποποιήσεων σε αλλαγές που έχουν ήδη εφαρμοστεί.

Ο δεύτερος πίνακας, DATABASECHANGELOGLOCK, χρησιμεύει για να αποτρέψει την ταυτόχρονη εκτέλεση των αλλαγών από πολλαπλά instances της εφαρμογής. Περιλαμβάνει ένα μόνο row, με πεδίο LOCKED που δηλώνει αν υπάρχει ενεργό lock, και πληροφορίες όπως η ώρα και ο host που το απέκτησε. Πριν ξεκινήσει οποιαδήποτε εκτέλεση αλλαγών, το Liquibase αποκτά αυτό το lock. Αν το lock είναι ήδη ενεργό, η διαδικασία περιμένει ή αποτυγχάνει, αποτρέποντας έτσι race conditions ή διπλές εγγραφές.

	ID	AUTHOR	FILENAME	DATEEXECUTED	ORDEREXECUTED	EXECTYPE
1	1	i.oikonomou	db/changelog.mysql.sql	2025-04-06 15:33:30	1	EXECUTED
2	2	i.oikonomou	db/changelog.mysql.sql	2025-04-06 15:33:30	2	EXECUTED
3	3	i.oikonomou	db/changelog.mysql.sql	2025-04-06 15:33:30	3	EXECUTED
4	4	i.oikonomou	db/changelog.mysql.sql	2025-04-06 15:33:30	4	EXECUTED
5	5	i.oikonomou	db/changelog.mysql.sql	2025-04-06 15:33:30	5	EXECUTED
6	6	a.tsantoulis	db/changelog.mysql.sql	2025-04-17 15:40:06	6	EXECUTED

Εικόνα 1: Πίνακας DATABASECHANGELOG

3.2 Πίνακες βάσης δεδομένων



Εικόνα 2: Μοντέλο Οντοτήτων-Συσχετίσεων βάσης δεδομένων

3.2.1 Πίνακας user

Στον πίνακα user υπάρχουν όλα τα στοιχεία που αφορούν τους χρήστες της εφαρμογής. Κάθε γραμμή του πίνακα αναπαριστά έναν ξεχωριστό χρήστη. Οι στήλες του πίνακα είναι οι εξής:

- ***id*** *: Μοναδικό αναγνωριστικό του χρήστη, που χρησιμοποιείται εσωτερικά στο backend για αναφορά στον χρήστη. Είναι το primary key του πίνακα και αυξάνεται αυτόματα (AUTO_INCREMENT) με κάθε νέο χρήστη που δημιουργείται.*
- ***email****: Είναι το email το οποίο χρησιμοποίησε ο χρήστης για να εγγραφεί στην εφαρμογή, και το email που χρησιμοποιεί κατά το login. Αυτή η στήλη έχει τον περιορισμό UNIQUE ώστε να αποτρέψει την εγγραφή δεύτερου λογαριασμού με το ίδιο email.*
- ***password****: Σε αυτή την στήλη αποθηκεύεται το hash του κωδικού του χρήστη. Το hash προκύπτει από τον plaintext κωδικό και ένα προκαθορισμένο salt.*
- ***status****: Περιέχει την κατάσταση του λογαριασμού του χρήστη. Μπορεί να έχει τις τιμές ACTIVE, PENDING_ACTIVATION, BLOCKED. Η κατάσταση PENDING_ACTIVATION σημαίνει ότι ο χρήστης δεν έχει επιβεβαιώσει τον λογαριασμό του μέσω του email επιβεβαίωσης, ενώ BLOCKED είναι μια κατάσταση που μας επιτρέπει να αποτρέψουμε κακόβουλους χρήστες στο να κάνουν login. Η κατάσταση ACTIVE είναι η κατάσταση που βρίσκεται ένας λογαριασμός που είναι επιβεβαιωμένος και ο χρήστης μπορεί να τον χρησιμοποιήσει κανονικά.*
- ***creation_date****: Περιέχει ένα timestamp το οποίο δημιουργείται κατά την δημιουργία ενός λογαριασμού χρήστη.*

3.2.2 Πίνακας user_password_reset

Αυτός ο πίνακας περιέχει πληροφορίες που αφορούν την διαδικασία επαναφοράς κωδικού, ενός λογαριασμού χρήστη. Κάθε γραμμή του πίνακα αναπαριστά μια ενεργή διαδικασία επαναφοράς, και περιέχει τις κατάλληλες πληροφορίες που χρειάζονται για την ασφαλή επαναφορά κωδικού.

- ***id****: Το μοναδικό αναγνωριστικό κάθε σειράς το οποίο είναι το κύριο κλειδί του πίνακα και αυξάνεται αυτόματα με κάθε νέα εγγραφή.*
- ***user_id****: Ξένο κλειδί το οποίο αναφέρεται στον πίνακα user και συνδέει τη διαδικασία επαναφοράς με τον αντίστοιχο χρήστη.*
- ***token****: Ένα μοναδικό string που δημιουργείται κατά την έναρξη της διαδικασίας επαναφοράς και αποστέλλεται στο χρήστη μέσω email. Χρησιμοποιείται για να ταυτοποιήσει το αίτημα επαναφοράς.*
- ***email_status****: Κατάσταση της αποστολής του email επαναφοράς κωδικού. Επιτρεπόμενες τιμές είναι οι SENDING, NOT_SENT, SENT. Αυτή η στήλη χρησιμοποιείται από την διεργασία αποστολής email για να καταλάβει ποια email εκκρεμεί η αποστολή τους.*
- ***email_send_timestamp****: Η χρονική στιγμή κατά την οποία το email αποστολής token στάλθηκε στον χρήστη. Χρησιμοποιείται για τον καθορισμό χρονικού ορίου ισχύος του token επαναφοράς κωδικού.*
- ***creation_date****: Η χρονική στιγμή δημιουργίας της εγγραφής, ορίζεται αυτόματα και χρησιμοποιείται για monitoring και debugging προβλημάτων αποστολής.*

3.2.3 Πίνακας user_account_confirmation

Αυτός ο πίνακας περιέχει πληροφορίες που αφορούν την διαδικασία επιβεβαίωσης εγγραφής των χρηστών μέσω email. Κάθε εγγραφή αντιστοιχεί σε ένα token ενεργοποίησης που έχει παραχθεί για έναν νέο χρήστη. Μετά την επιβεβαίωση χρήστη η εγγραφή αφαιρείται από αυτόν τον πίνακα.

- ***id****: Πρωτεύον κλειδί της εγγραφής. Είναι μοναδικό και παράγεται αυτόματα με χρήση AUTO_INCREMENT.*
- ***user_id****: Ξένο κλειδί προς τον πίνακα user. Καθορίζει ποιος χρήστης σχετίζεται με την επιβεβαίωση λογαριασμού.*
- ***token****: Ένα μοναδικό string που δημιουργείται κατά την εγγραφή του χρήστη και αποστέλλεται μέσω email. Χρησιμοποιείται στη διαδικασία επιβεβαίωσης για να επαληθευτεί ότι ο παραλήπτης του email είναι και ο κάτοχος του λογαριασμού.*

- ***email_status**: Κατάσταση της αποστολής του email επιβεβαίωσης λογαριασμού. Επιτρεπόμενες τιμές είναι οι SENDING, NOT_SENT, SENT. Αυτή η στήλη χρησιμοποιείται από την διεργασία αποστολής email για να καταλάβει ποια email εκκρεμεί η αποστολή τους.*
- ***email_send_timestamp**: Η χρονική στιγμή κατά την οποία το email επιβεβαίωσης στάλθηκε στον χρήστη. Χρησιμοποιείται για τον καθορισμό χρονικού ορίου ισχύος του token επιβεβαίωσης, και για επαναποστολή του email μετά από ένα χρονικό διάστημα, εφόσον ο χρήστης το ζητήσει.*
- ***creation_date**: Η χρονική στιγμή δημιουργίας της εγγραφής, ορίζεται αυτόματα και χρησιμοποιείται για monitoring και debugging προβλημάτων αποστολής.*

3.2.4 Πίνακας subscriber

Ο πίνακας subscriber περιέχει πληροφορίες για τους συνδρομητές των καμπανιών email που δημιουργούνται από τους χρήστες της εφαρμογής. Κάθε εγγραφή αντιπροσωπεύει έναν μοναδικό συνδρομητή και αφορά μόνο έναν χρήστη της εφαρμογής (user).

- ***id**: Πρωτεύον κλειδί. Μοναδικό αναγνωριστικό της εγγραφής, με αυτόματη αύξηση (AUTO_INCREMENT).*
- ***user_id**: Ξένο κλειδί προς τον πίνακα user. Δηλώνει σε ποιον χρήστη ανήκει ο συνδρομητής. Είναι υποχρεωτικό (NOT NULL) και διασφαλίζει τον διαχωρισμό των συνδρομητών μεταξύ διαφορετικών χρηστών*
- ***first_name**: Το όνομα του συνδρομητή. Χρησιμοποιείται για προσωποποίηση του περιεχομένου των emails. Αυτό το πεδίο μπορεί να είναι κενό.*
- ***last_name**: Το επώνυμο του συνδρομητή. Χρησιμοποιείται για προσωποποίηση του περιεχομένου των emails. Αυτό το πεδίο μπορεί να είναι κενό.*
- ***email**: Η διεύθυνση email του συνδρομητή. Δεν υπάρχει περιορισμός μοναδικότητας στον πίνακα subscriber συνολικά, γιατί μπορεί διαφορετικοί χρήστες να έχουν τον ίδιο συνδρομητή στη δική τους λίστα.*
- ***status**: Καθορίζει την κατάσταση του συνδρομητή. Παίρνει τις τιμές ACTIVE, OPTED_OUT, DELETED. Οι συνδρομητές με κατάσταση ACTIVE μπορούν να συμπεριληφθούν σε καμπάνιες χρηστών και να λάβουν email. Οι OPTED_OUT συνδρομητές δεν μπορούν να συμπεριληφθούν σε καμπάνια, αλλά ο χρήστης μπορεί να τους δει στην λίστα συνδρομητών του. Οι DELETED χρήστες δεν εμφανίζονται στην λίστα συνδρομητών και ούτε λαμβάνουν email. Ο λόγος που ένας συνδρομητής έχει κατάσταση DELETED, και δεν διαγράφεται είναι για να διασφαλιστεί η συνεκτικότητα της βάσης και η ιστορικότητα. Με την ένδειξη DELETED, αποφεύγεται η φυσική διαγραφή της εγγραφής, επιτρέποντας την αναδρομή σε παλαιότερες καμπάνιες, την ανάλυση στατιστικών στοιχείων, αλλά και την αποτροπή επανεγγραφής με το ίδιο email χωρίς συναίνεση.*

3.2.5 Πίνακας _group

Ο πίνακας _group ορίζει τις ομάδες συνδρομητών που δημιουργούν οι χρήστες, ώστε να οργανώνουν καλύτερα τους παραλήπτες τους. Η χρήση του χαρακτήρα underscore στο όνομα έγινε επειδή η λέξη group είναι δεσμευμένη λέξη στη SQL. Κάθε γραμμή του πίνακα αντιπροσωπεύει ένα group ενός χρήστη της εφαρμογής.

- ***id**: Πρωτεύον κλειδί του πίνακα, με αυτόματη αύξηση (AUTO_INCREMENT). Αναγνωρίζει μοναδικά κάθε ομάδα.*
- ***user_id**: Ξένο κλειδί προς τον πίνακα user. Δηλώνει σε ποιον χρήστη ανήκει η ομάδα. Αυτό επιτρέπει σε κάθε χρήστη να έχει τις δικές του ομάδες ανεξάρτητα από άλλους.*
- ***name**: Το όνομα της ομάδας, όπως το δηλώνει ο χρήστης. Χρησιμοποιείται για την αναγνώριση και επιλογή ομάδων κατά τη δημιουργία καμπάνιας.*
- ***creation_date**: Η ημερομηνία δημιουργίας της ομάδας. Ορίζεται αυτόματα με CURRENT_TIMESTAMP.*

3.2.6 Πίνακας subscriber_group

Ο πίνακας subscriber_group υλοποιεί τη σχέση πολλών-προς-πολλών μεταξύ συνδρομητών και ομάδων. Ένας συνδρομητής μπορεί να ανήκει σε πολλές ομάδες και μία ομάδα μπορεί να περιέχει πολλούς συνδρομητές. Κάθε γραμμή αντιπροσωπεύει έναν subscriber που ανήκει σε ένα συγκεκριμένο group.

- ***subscriber_id****: Ξένο κλειδί προς τον πίνακα subscriber. Δηλώνει τον συνδρομητή που ανήκει στην ομάδα.*
- ***group_id****: Ξένο κλειδί προς τον πίνακα _group. Δηλώνει την ομάδα στην οποία έχει ενταχθεί ο συνδρομητής.*

Ορίζει ως σύνθετο πρωτεύον κλειδί τον συνδυασμό των δύο πεδίων (PRIMARY KEY (subscriber_id, group_id)). Αυτό σημαίνει ότι ένας συνδρομητής δεν μπορεί να ενταχθεί στην ίδια ομάδα περισσότερες από μία φορές.

3.2.7 Πίνακας campaign

Ο πίνακας campaign καταγράφει τις καμπάνιες αποστολής email που δημιουργεί κάθε χρήστης. Κάθε καμπάνια περιέχει τα απαραίτητα δεδομένα για να προγραμματιστεί, να σταλεί και να παρακολουθηθεί. Κάθε γραμμή του πίνακα αναπαριστά μια ξεχωριστή καμπάνια ενός χρήστη.

- ***id****: Πρωτεύον κλειδί. Μοναδικό αναγνωριστικό καμπάνιας, με αυτόματη αύξηση (AUTO_INCREMENT).*
- ***user_id****: Ξένο κλειδί προς τον πίνακα user. Δηλώνει τον δημιουργό της καμπάνιας.*
- ***title****: Αφορά τον τίτλο που έδωσε ο χρήστης στην καμπάνια. Χρησιμοποιείται μόνο για εσωτερική χρήση, στο dashboard του χρήστη, και δεν αφορά τα email.*
- ***mail_sender****: Η διεύθυνση αποστολέα που όρισε ο χρήστης. Περιέχει το From πεδίο που θα βλέπει ο παραλήπτης.*
- ***mail_subject****: Το θέμα του email, όπως θα εμφανίζεται στον παραλήπτη.*
- ***mail_content****: Αναφορά (μέσω UUID) στον πίνακα mail_content που περιέχει το πραγματικό σώμα του email.*
- ***start_date****: Ορίζει την ημερομηνία και ώρα έναρξης της καμπάνιας. Χρησιμοποιείται για τον προγραμματισμό αποστολής, ώστε η καμπάνια να ξεκινήσει αυτόματα την καθορισμένη στιγμή.*
- ***end_date****: Αυτό το πεδίο παραμένει αρχικά κενό (NULL) και λαμβάνει τιμή όταν η καμπάνια ολοκληρώνεται ή τερματίζεται χειροκίνητα. Η τιμή αντιστοιχεί στην ημερομηνία και ώρα λήξης της καμπάνιας, ώστε να καταγράφεται το χρονικό σημείο που σταμάτησε η αποστολή των email.*
- ***status****: Δηλώνει την τρέχουσα κατάσταση της καμπάνιας. Επιτρεπόμενες τιμές:*
- *DRAFT: Υπό επεξεργασία, δεν έχει προγραμματιστεί.*
- *SCHEDULED: Έχει προγραμματιστεί για αποστολή σε μελλοντικό χρόνο.*
- *RUNNING: Είναι σε εξέλιξη η αποστολή.*
- *STOPPED: Η αποστολή διακόπηκε χειροκίνητα ή από σφάλμα.*
- *FINISHED: Η αποστολή έχει ολοκληρωθεί.*
- ***creation_date****: Ημερομηνία δημιουργίας της καμπάνιας. Δημιουργείται αυτόματα.*

3.2.8 Πίνακας campaign_group

Ο πίνακας campaign_group υλοποιεί τη σχέση πολλών-προς-πολλούς μεταξύ των καμπανιών (campaign) και των ομάδων συνδρομητών (_group). Επιτρέπει σε κάθε καμπάνια να στοχεύει μία ή περισσότερες ομάδες παραληπτών.

- ***campaign_id**: Ξένο κλειδί που αναφέρεται στην καμπάνια.*
- ***group_id**: Ξένο κλειδί που αναφέρεται στην ομάδα συνδρομητών.*

Ορίζει ως σύνθετο πρωτεύον κλειδί τον συνδυασμό των δύο πεδίων (campaign_id, group_id). Αυτό σημαίνει ότι μία καμπάνια δεν μπορεί να στοχεύσει το ίδιο group δύο φορές, και να αποσταλούν διπλά email.

3.2.9 Πίνακας campaign_subscriber

Ο πίνακας campaign_subscriber υλοποιεί τη σχέση πολλών-προς-πολλούς μεταξύ καμπανιών (campaign) και συνδρομητών (subscriber). Επιτρέπει την άμεση συσχέτιση συγκεκριμένων παραληπτών με μία καμπάνια, ανεξάρτητα από ομάδες.

- ***campaign_id**: Ξένο κλειδί προς τον πίνακα campaign. Αναφέρεται στην καμπάνια που αποστέλλεται το email.*
- ***subscriber_id**: Ξένο κλειδί προς τον πίνακα subscriber. Δηλώνει τον παραλήπτη που περιλαμβάνεται στην καμπάνια.*

Ορίζει ως σύνθετο πρωτεύον κλειδί τον συνδυασμό των δύο πεδίων (campaign_id, subscriber_id). Αυτό σημαίνει ότι μία καμπάνια δεν μπορεί να στοχεύσει το ίδιο subscriber δύο φορές.

3.2.10 Πίνακας mail_content

Ο πίνακας mail_content αποθηκεύει το πραγματικό περιεχόμενο των email που αποστέλλονται μέσω των καμπανιών. Η διαχωρισμένη αποθήκευση του περιεχομένου από τον πίνακα campaign επιτρέπει επαναχρησιμοποίηση και αποτελεσματική διαχείριση μεγάλων κειμένων.

- ***id**: Μοναδικό αναγνωριστικό τύπου UUID , το οποίο χρησιμοποιείται ως πρωτεύον κλειδί.*
- ***content**: Το σώμα του email, αποθηκευμένο ως MEDIUMTEXT, επιτρέποντας μεγέθη έως 16MB. Περιέχει κώδικα HTML, CSS ή απλό κείμενο που αποτελεί το μήνυμα προς τον παραλήπτη.*
- ***last_update**: Timestamp της τελευταίας τροποποίησης του περιεχομένου.*

3.2.11 Πίνακας email_outgoing_event

Ο πίνακας email_outgoing_event καταγράφει κάθε μεμονωμένο email που έχει δημιουργηθεί προς αποστολή, και χρησιμοποιείται για την αποθήκευση κατάστασης αποστολής των email. Σε αυτό τον πίνακα βασίζονται διάφορες διεργασίες, για να διαπιστώσουν ποια email χρειάζονται αποστολή, ποια δεν στάλθηκαν επιτυχώς και ποια επιτυχώς. Επίσης αυτός ο πίνακας χρησιμοποιείται και για την εξαγωγή στατιστικών κάθε καμπάνιας, ώστε να τα βλέπει ο χρήστης.

- ***id**: Πρωτεύον κλειδί με αυτόματη αύξηση (AUTO_INCREMENT), μοναδικό για κάθε εγγραφή.*
- ***campaign_id**: Ξένο κλειδί προς τον πίνακα campaign. Συσχετίζει το email με την καμπάνια από την οποία προήλθε.*
- ***subscriber_id**: Ξένο κλειδί προς τον πίνακα subscriber. Υποδεικνύει τον παραλήπτη του συγκεκριμένου email.*
- ***content**: Αποθηκεύει το ακριβές προσωποποιημένο περιεχόμενο του email που θα σταλεί, σε μορφή MEDIUMTEXT, επιτρέποντας μεγάλες ποσότητες δεδομένων.*
- ***is_queued**: Boolean flag που δείχνει αν το email βρίσκεται ήδη στην ουρά αποστολής.*
- ***queued_timestamp**: Χρονική σήμανση κατά την οποία το email μπήκε στην ουρά.*

- ***is_sent****: Boolean flag που υποδηλώνει αν το email έχει ήδη αποσταλεί.*
- ***sent_timestamp****: Χρονική σήμανση αποστολής του email.*
- ***mail_server_response****: Κείμενο που αποθηκεύει την απάντηση του mail server μετά την αποστολή.*
- ***last_update****: Ημερομηνία και ώρα τελευταίας ενημέρωσης της εγγραφής.*
- ***is_locked****: Boolean flag που αποτρέπει ταυτόχρονες προσπάθειες αποστολής του ίδιου email από πολλαπλές διεργασίες.*
-

3.2.12 Πίνακας email_tracking

Ο πίνακας email_tracking χρησιμοποιείται για την παρακολούθηση ανοίγματος των απεσταλμένων email. Για κάθε email που αποστέλλεται δημιουργείται μια εγγραφή σε αυτόν το πίνακα. Κάθε φορά που ένας χρήστης ανοίγει ένα email, τροποποιείται η εγγραφή ώστε να καταγράψει το άνοιγμα του email. Συνεπώς, ο πίνακας καταγράφει μόνο αν έχει ανοιχτεί ένα email, και όχι πόσες φορές.

- ***campaign_id****: Ξένο κλειδί προς τον πίνακα campaign. Καθορίζει την καμπάνια στην οποία ανήκει το email.*
- ***subscriber_id****: Ξένο κλειδί προς τον πίνακα subscriber. Είναι το id του subscriber που άνοιξε το email.*
- ***unique_token****: Μοναδικό αναγνωριστικό token που εισάγεται στα email ως μέρος του link που χρησιμοποιείται για την παρακολούθηση.*
- ***is_opened****: Boolean flag που δηλώνει αν το email έχει ανοιχτεί.*
- ***opened_timestamp****: Χρονική σήμανση που καταγράφει πότε το email άνοιξε για πρώτη φορά.*
- ***ip_address****: Η IP διεύθυνση από την οποία έγινε το άνοιγμα.*

3.2.13 Πίνακας campaign_spam

Ο πίνακας αυτός χρησιμοποιείται για την καταγραφή στατιστικών σχετικά με την διαδικασία spam score. Κάθε φορά που ο χρήστης προγραμματίζει μια καμπάνια για εκτέλεση, σε αυτόν τον πίνακα αποθηκεύεται το spam score της καμπάνιας. Ο λόγος που χρειάζεται το spam score των καμπανιών ενός χρήστη είναι ώστε να μπορεί να διαπιστωθεί ποιος είναι ο κατάλληλος mail server για να στείλει μελλοντικές καμπάνιες. Όταν ένας χρήστης έχει ιστορικό καμπανιών με υψηλό spam score, τότε ο αλγόριθμος στέλνει τις μελλοντικές καμπάνιες του με χαμηλής αξιοπιστίας mail server. Αντίθετα, αν ο χρήστης έχει ιστορικό με χαμηλό spam score, ο αλγόριθμος προτιμά mail server με υψηλή αξιοπιστία.

- ***user_id****: Ξένο κλειδί προς τον πίνακα user. Αναφέρεται στον δημιουργό της καμπάνιας.*
- ***campaign_id****: Ξένο κλειδί προς τον πίνακα campaign. Καθορίζει την καμπάνια που αξιολογείται.*
- ***spam_score****: Αριθμός που αποθηκεύει το συνολικό σκορ spam, όπως προκύπτει από τον έλεγχο με την χρήση spamAssasin*

Το πρωτεύον κλειδί του πίνακα campaign_spam είναι σύνθετο και αποτελείται από τα πεδία user_id και campaign_id. Αυτό σημαίνει ότι για κάθε συνδυασμό χρήστη και καμπάνιας υπάρχει μία και μόνο εγγραφή, αποτρέποντας διπλοεγγραφές.

3.2.14 Πίνακας mail_server

Ο πίνακας mail_server περιέχει όλους τους διαθέσιμους mail server που έχει στην διάθεση της η εφαρμογή, για αποστολή email. Κάθε εγγραφή περιέχει επίσης στοιχεία που αφορούν την αξιοπιστία ενός mail server. id: Πρωτεύον κλειδί με αυτόματη αύξηση (AUTO_INCREMENT).

- ***server_uuid**: Μοναδικός αναγνωριστής UUID για τον κάθε mail server.*
- ***url**: Η διεύθυνση του mail server (SMTP endpoint).*
- ***open_rate**: Ποσοστό ανοίγματος email που έχουν αποσταλεί μέσω αυτού του server.*
- ***premium_status**: Boolean που δηλώνει αν ο server έχει χαρακτηριστεί ως premium (Υψηλή αξιοπιστία).*
- ***spam_threshold**: Αριθμητική τιμή που καθορίζει το όριο αποδοχής spam για τον server. Για να σταλεί μια καμπάνια με έναν mail server, ο mail server πρέπει να έχει spam_threshold μεγαλύτερο η ίσο με το spam score της καμπάνιας.*
- ***username / password**: Διαπιστευτήρια σύνδεσης για τον mail server.*
- ***blacklisted**: Boolean flag που δείχνει αν ο server έχει μπει σε λίστα αποκλεισμού (blacklist) και πρέπει να αποφεύγεται.*
- ***creation_date**: Ημερομηνία και ώρα δημιουργίας της εγγραφής, με προεπιλεγμένη τιμή CURRENT_TIMESTAMP.*

3.3 Indexes βάσης δεδομένων

Ένα index στη βάση δεδομένων είναι μια δομή δεδομένων που επιταχύνει την αναζήτηση και ανάκτηση εγγράφων σε έναν πίνακα. Αντί η βάση να σαρώνει σειριακά ολόκληρο τον πίνακα για να βρει τα δεδομένα που ζητούνται, το ευρετήριο λειτουργεί σαν ένα γρήγορο “πίνακα περιεχομένων” που οδηγεί άμεσα στη θέση των εγγραφών. Αυτό βελτιώνει σημαντικά την απόδοση των ερωτημάτων, ειδικά όταν οι πίνακες έχουν μεγάλο όγκο δεδομένων ή όταν γίνονται συχνές αναζητήσεις με συγκεκριμένα κριτήρια.

Το αρνητικό των indexes είναι ότι επιβραδύνουν τις λειτουργίες εγγραφής στην βάση, γιατί για κάθε εγγραφή πρέπει να ανανεώνεται και η δομή δεδομένων του index. Για τον λόγο αυτό στην βάση δεδομένων έχουν χρησιμοποιηθεί μόνο τα απαραίτητα indexes, για ερωτήματα που είτε τρέχουν πολύ συχνά, είτε ερωτήματα που αφορούν πολύ μεγάλους πίνακες, και καθυστερούσαν πολύ.

Τα indexes που χρησιμοποιήθηκαν είναι τα εξής:

- ***Συνδυαστικό index στο subscriber (email, user_id)**: Για κάθε εισαγωγή νέου subscriber, πρέπει πρώτα να ελεγχθεί αν το email υπάρχει ήδη στην λίστα των subscriber ενός χρήστη. Αυτό το συνδυαστικό index κάνει την διαδικασία πολύ γρήγορη και βελτιώνει κατα πολύ τον χρόνο εισαγωγής subscribers.*
- ***index στο campaign (user_id)**: Βοηθά στην ταχεία ανάκτηση όλων των καμπανιών ενός χρήστη, για εμφάνιση στο dashboard.*
- ***Συνδυαστικό index στο campaign (status, start_date)**: Η διεργασία που είναι υπεύθυνη για την εκτέλεση των καμπανιών, κάνει ανά τακτά χρονικά διαστήματα ερωτήματα στην βάση, για να πάρει καμπάνιες που έχουν status ως SCHEDULED και start_date προγενέστερη του τώρα. Αυτό το index κάνει πολύ πιο γρήγορη αυτή την διεργασία, και είναι απαραίτητο, γιατί αυτή η διεργασία κάνει το συγκεκριμένο ερώτημα ανά ένα λεπτό. Αν δεν υπήρχε το index θα προκαλούσε αχρείαστο φόρτο στην βάση δεδομένων και καθυστερήσεις.*
- ***index στο email_outgoing_event (campaign_id)**: Ο πίνακας email_outgoing_event είναι δυνητικά ένας πολύ μεγάλος πίνακας που περιέχει ότι email έχει σταλεί από την εφαρμογή. Αυτός ο πίνακας χρησιμοποιείται για να βλέπει ο χρήστης στατιστικά αποστολής email για κάθε καμπάνια του. Δεδομένου λοιπόν ότι ο πίνακας μπορεί να είναι πολύ μεγάλος είναι αναγκαίο να υπάρχει index για να επιταχύνει την ανάκτηση δεδομένων για μία συγκεκριμένη καμπάνια.*
- ***index στο email_outgoing_event (is_locked)**: Η διεργασία που είναι υπεύθυνη για την δημιουργία email, κάνει συνεχώς ερωτήματα στην βάση για να λάβει email που χρειάζονται δημιουργία. Σε αυτά τα ερωτήματα χρησιμοποιείται το πεδίο is_locked για να διασφαλιστεί ότι οι εγγραφές που θα επιλεχθούν δεν είναι ήδη υπό επεξεργασία από άλλη διεργασία. Για αυτό τον λόγο έχει δημιουργηθεί αυτό το index, για την επιτάχυνση του ερωτήματος.*

- ***index στο email_tracking (unique token)****: Κάθε φορά που ένας χρήστης ανοίγει ένα email, το backend παίρνει τον μοναδικό κωδικό του tracking link και ενημερώνει την κατάλληλη εγγραφή στον πίνακα email_tracking. Δεδομένου λοιπόν ότι ο πίνακας είναι εν δυνάμει πολύ μεγάλος, γιατί περιέχει μια εγγραφή για κάθε email που έχει σταλεί, και δεδομένου ότι η ενημέρωση γίνεται με βάση το unique_token, είναι απαραίτητο το συγκεκριμένο index, γιατί αλλιώς η όλη διαδικασία θα είχε τρομερή καθυστέρηση.*
- ***index στο email_tracking (is_opened)****: Για την παρουσίαση των στατιστικών ανοίγματος email στον χρήστη, χρησιμοποιείται ο πίνακας email_tracking, και μετρούνται όλες οι εγγραφές που έχουν is_opened ως true. Αυτό το index διασφαλίζει την ταχύτητα της όλης διαδικασίας.*

3.4 Κανονικοποίηση βάσης δεδομένων

Η κανονικοποίηση αποτελεί μια βασική διαδικασία στο σχεδιασμό βάσεων δεδομένων, με σκοπό την οργάνωση των δεδομένων ώστε να ελαχιστοποιηθεί η πλεονασματικότητα και να αποτραπούν ανωμαλίες κατά την εισαγωγή, ενημέρωση ή διαγραφή δεδομένων. Η βάση δεδομένων της εφαρμογής σχεδιάστηκε ώστε να πληροί τα στάδια της κανονικοποίησης, ως εξής:

- ***Πρώτη Κανονική Μορφή (1NF):*** *Η βάση πληροί την 1NF, καθώς κάθε πεδίο σε κάθε πίνακα περιέχει ατομικές, μη διαιρετές τιμές. Δεν υπάρχουν πίνακες με επαναλαμβανόμενα σύνολα δεδομένων ή πολλαπλές τιμές σε μία στήλη. Για παράδειγμα, οι πίνακες user και subscriber αποθηκεύουν ξεκάθαρα διακριτές τιμές ανά πεδίο, χωρίς λίστες ή σύνολα τιμών.*
- ***Δεύτερη Κανονική Μορφή (2NF):*** *Η 2NF επιτυγχάνεται καθώς όλοι οι μη πρωτεύοντες πεδία σε πίνακες με σύνθετο πρωτεύον κλειδί εξαρτώνται πλήρως από το σύνολο του πρωτεύοντος κλειδιού. Πίνακες όπως το subscriber_group, με σύνθετο κλειδί (subscriber_id, group_id), δεν περιέχουν πεδία που να εξαρτώνται μόνο από μέρος του κλειδιού, εξασφαλίζοντας έτσι πλήρη εξάρτηση.*
- ***Τρίτη Κανονική Μορφή (3NF):*** *Η 3NF τηρείται, καθώς δεν υπάρχουν μεταβατικές εξαρτήσεις. Κάθε μη κλειδωμένο πεδίο εξαρτάται άμεσα από το πρωτεύον κλειδί και όχι μέσω άλλου μη κλειδωμένου πεδίου. Για παράδειγμα, στον πίνακα campaign, τα πεδία title, mail_subject και status σχετίζονται άμεσα με το id της καμπάνιας και όχι μεταξύ τους.*
  Τέταρτη Κανονική Μορφή (4NF)*: Η βάση δεδομένων τηρεί την 4NF διαιρώντας ξεκάθαρα τις σχέσεις πολλών-προς-πολλούς σε ξεχωριστούς πίνακες όπως subscriber_group, campaign_group και campaign_subscriber. Αυτό αποτρέπει την ύπαρξη πολυτιμικών εξαρτήσεων, όπου δύο ή περισσότερα ανεξάρτητα σύνολα δεδομένων αποθηκεύονται στον ίδιο πίνακα.*

Κεφάλαιο 4ο: Υλοποίηση Back-end

Σε αυτό το κεφάλαιο θα εξετάσουμε την υλοποίηση του back-end, το οποίο αποτελείται από 3 ξεχωριστά modules. Θα δούμε τις αρμοδιότητες του κάθε module και θα εξηγήσουμε το τρόπο υλοποίησης του.

4.1 Core module

Το core module είναι ο πυρήνας του back-end συστήματος. Η κύρια αρμοδιότητα του είναι να παρέχει την διεπαφή (API) την οποία θα χρησιμοποιούν οι clients (front-end) της εφαρμογής μας. Αυτό το module διαχειρίζεται την εγγραφή και αυθεντικοποίηση χρηστών, την δημιουργία και επεξεργασία καμπανιών μέσω email, την βελτιστοποίηση των email και γενικά ότι αφορά την αλληλεπίδραση του front-end με το back-end.

4.1.1 Δομή του core module

Το core module ακολουθεί το Controller-Service-Repository πρότυπο. Το Controller επίπεδο είναι αποκλειστικά υπεύθυνο για τις λειτουργικότητες οι οποίες χρησιμοποιούνται από εξωτερικές οντότητες και στην ουσία ορίζει τον τρόπο μέσω του οποίου γίνεται η επικοινωνία του backend με το front-end. Συνοπτικά, το Controller επίπεδο περιέχει τους κανόνες που θέτει το API του back-end.

Το επίπεδο Repository περιέχει όλη την λογική που αφορά την επικοινωνία με την βάση δεδομένων. Ο σκοπός του είναι να παρέχει abstraction πάνω από την βάση δεδομένων ώστε τα άλλα επίπεδα, όπως το επίπεδο Service, να χρησιμοποιούν την βάση δεδομένων χωρίς να επηρεάζονται από τις λεπτομέρειες υλοποίησης της βάσης δεδομένων.

Η αρμοδιότητα του Service στρώματος είναι να εφαρμόζει το business logic της εφαρμογής, εκτελώντας του κατάλληλους αλγορίθμους και κάνοντας το κατάλληλο validation, ώστε να φέρει εις πέρας τα αιτήματα των clients.

Ο διαχωρισμός στα στρώματα Controller-Service-Repository επιλέχθηκε ώστε να επιβάλλει ένα καθαρό διαχωρισμό ευθυνών, κάνοντας την ανάπτυξη, την συντήρηση και την επεκτασιμότητα της εφαρμογής πιο εύκολη.

Εκτός από τον σαφή διαχωρισμό της εφαρμογής σε στρώματα, επιδιώξαμε και τον σαφή ορισμό των πληροφοριών που διακινούνται στα διαφορετικά μέρη της εφαρμογής. Για αυτό τον λόγο, στην υλοποίηση του Core module υπάρχει το πακέτο dto, το οποίο περιέχει όλες τις αναπαραστάσεις δεδομένων που χρησιμοποιούνται για την ανταλλαγή πληροφοριών είτε εσωτερικά στην εφαρμογή, είτε με εξωτερικούς clients. Με αυτό τον τρόπο επιτυγχάνεται καλύτερος έλεγχος της ροής των δεδομένων και δημιουργείται ένα σαφές πρότυπο που ακολουθούν όλα τα μέρη της εφαρμογής.

Τέλος, το πακέτο entity είναι υπεύθυνο για τον ορισμό των μοντέλων που αντιστοιχούν απευθείας στους πίνακες της βάσης δεδομένων. Κάθε entity αναπαριστά έναν πίνακα της βάσης και περιλαμβάνει όλα τα απαραίτητα πεδία, καθώς και τις σχέσεις με άλλα entities (όπως one-to-many ή many-to-many). Τα entities χρησιμοποιούνται αποκλειστικά από το Repository επίπεδο για την αποθήκευση και ανάκτηση δεδομένων. Ο διαχωρισμός των entities από τα DTOs επιτρέπει την ανεξαρτησία του persistence layer από τα δεδομένα που εκθέτει το API, προσφέροντας αυξημένη ασφάλεια, καθαρότητα στον κώδικα και ευκολία στη μελλοντική επέκταση ή τροποποίηση της δομής της βάσης.

4.1.2 Το API της εφαρμογής

Το API της εφαρμογής έχει υλοποιηθεί σύμφωνα με τα πρότυπα και τις αρχές που θέτει το REST (Representational State Transfer) πρότυπο. Όλοι οι πόροι στο σύστημα αναπαρίστανται με μοναδικά URLs, και η εκτέλεση λειτουργίας πάνω σε αυτούς τους πόρους γίνεται με την χρήση των κατάλληλων HTTP μεθόδων. Κάθε αίτημα στο API είναι ανεξάρτητο από προηγούμενα αιτήματα, και περιέχει όλες τις κατάλληλες πληροφορίες για την εκτέλεση του. Αυτό καθιστά το API stateless, ακριβώς όπως ορίζει και το πρότυπο REST[34]. Για την ανταλλαγή δεδομένων χρησιμοποιείται η μορφή JSON.

Ο λόγος που επιλέξαμε να ακολουθήσουμε το πρότυπο REST είναι γιατί το χαρακτηριστικό ότι είναι stateless διευκολύνει την κλιμακωσιμότητα της εφαρμογής, καθώς και την επεκτασιμότητα της. Επίσης, το γεγονός ότι είναι ένα δημοφιλές και ευρέως αποδεκτό πρότυπο κάνει την ενσωμάτωση με διάφορα συστήματα και clients πιο απλή και γρήγορη, μειώνοντας την πολυπλοκότητα ανάπτυξης και τη συντήρηση. Τέλος, η χρήση της μορφής JSON για την ανταλλαγή δεδομένων προσφέρει ευκολία στην ανάγνωση και επεξεργασία τόσο από τους προγραμματιστές όσο και από τις εφαρμογές-πελάτες, ενώ υποστηρίζεται ευρέως από σύγχρονες πλατφόρμες και γλώσσες προγραμματισμού, εξασφαλίζοντας γρήγορη και αποδοτική επικοινωνία μεταξύ client και server.

4.1.2.1 Περιγραφή API Endpoints

Η οργάνωση των endpoints έχει γίνει με βάση τους πόρους τους οποίους αφορούν. Τα endpoints κάθε πόρου έχουν υλοποιηθεί σε ξεχωριστή κλάση.

Παρακάτω εξηγούνται συνοπτικά όλα τα endpoints, το URL και η HTTP μέθοδος που χρησιμοποιούνται, καθώς και ο σκοπός του κάθε endpoint.

4.1.2.1.1 Endpoints /user

Τα παρακάτω endpoints χρησιμοποιούνται για λειτουργίες που αφορούν τον User πόρο. Χρησιμοποιούνται για την εγγραφή χρηστών, την αυθεντικοποίηση τους καθώς και την επιβεβαίωση του email τους. Η υλοποίηση τους γίνεται από την κλάση UserController.

POST /user	Το endpoint αυτό χρησιμοποιείται για την εγγραφή νέου χρήστη. Λαμβάνει ως είσοδο ένα αντικείμενο τύπου UserRegistrationDto με τα στοιχεία εγγραφής και επιστρέφει κατάλληλη HTTP απόκριση ανάλογα με το αποτέλεσμα.
GET /user/confirmation	Χρησιμοποιείται για την επιβεβαίωση ενός λογαριασμού μέσω token που έχει σταλεί με email. Λαμβάνει το token ως query parameter.
POST /user/logout	Εκτελεί αποσύνδεση χρήστη, ακυρώνοντας το ενεργό token.
POST /user/login	Χρησιμοποιείται για τη σύνδεση χρήστη. Λαμβάνει ως είσοδο ένα UserLoginDto, και αν η αυθεντικοποίηση είναι επιτυχής, επιστρέφει JWT token σε cookie.

Πίνακας 2: Endpoints /user

4.1.2.1.2 Endpoints /subscriber

Τα παρακάτω endpoints χρησιμοποιούνται για λειτουργίες που αφορούν τον subscriber πόρο. Επιτρέπουν στον client να διαχειριστεί τους subscribers, να προσθέσει νέους, και να τους οργανώσει σε ομάδες. Αυτά τα endpoints χρησιμοποιούνται μόνο από αυθεντικοποιημένους χρήστες. Η υλοποίηση τους γίνεται από την κλάση SubscriberController.

PUT /subscriber	Επιτρέπει την εισαγωγή ή ενημέρωση (upsert) ενός συνδρομητή σε συγκεκριμένο group. Δέχεται το groupId ως query parameter και ένα SubscriberImportDto στο σώμα του αιτήματος, που περιέχει τα στοιχεία του συνδρομητή.
GET /subscriber/id/{subscriberId}	Επιστρέφει τα δεδομένα ενός συνδρομητή βάσει του subscriberId.
GET /subscriber?groupId=1	Επιστρέφει όλους τους συνδρομητές που ανήκουν σε συγκεκριμένο group. Το group ID δίνεται ως query parameter. Περιλαμβάνει έλεγχο ότι ο χρήστης έχει δικαίωμα πρόσβασης στο group.
DELETE /subscriber/id/{id}/group/{groupId}	Αφαιρεί έναν συγκεκριμένο συνδρομητή από ένα group, με έλεγχο ταυτότητας χρήστη και πρόσβασης στο group.
PUT /subscriber/id/{id}/optout	Ορίζει την κατάσταση συνδρομής ενός χρήστη σε OPTED_OUT, σηματοδοτώντας την επιλογή του να εξαιρεθεί από περαιτέρω επικοινωνία. Η λειτουργία εκτελείται μόνο αν ο χρήστης έχει πρόσβαση στον συνδρομητή.
POST /subscriber/upload	Υλοποιεί τη μαζική εισαγωγή συνδρομητών μέσω αρχείου CSV, επιτρέποντας στον χρήστη να ανεβάσει δεδομένα απευθείας σε συγκεκριμένη ομάδα (group). Το αίτημα θα πρέπει να περιλαμβάνει τον header content-type: multipart/form-data και δέχεται ένα CSV αρχείο και εως 4 query parameters. Η παράμετρος groupId (υποχρεωτική) που ορίζει το group στο οποίο θα εισαχθούν οι συνδρομητές, και τρεις προαιρετικές παραμέτρους (csvField1,csvField2,csvField3) που χρησιμοποιούνται για να αντιστοιχήσουν τις στήλες του CSV αρχείου στα κατάλληλα πεδία του συνδρομητή. Αν δεν οριστούν αυτές οι παράμετροι, η προεπιλεγμένη συμπεριφορά είναι να θεωρηθεί η πρώτη στήλη ως το email του συνδρομητή, και οι άλλες στήλες να αγνοηθούν.

Πίνακας 3: Endpoints /subscriber

4.1.2.1.3 Endpoints /group

Τα παρακάτω endpoints χρησιμοποιούνται για λειτουργίες που αφορούν τον group πόρο. Επιτρέπουν στον client να δημιουργήσει και να επεξεργαστεί τα groups του. Αυτά τα endpoints χρησιμοποιούνται μόνο από αυθεντικοποιημένους χρήστες. Η υλοποίηση τους γίνεται από την κλάση GroupController.

POST /group	Δημιουργεί μία νέα ομάδα για τον αυθεντικοποιημένο χρήστη. Δέχεται το όνομα της ομάδας (groupName) ως query parameter.
GET /group/all	Επιστρέφει όλες τις ομάδες που έχει δημιουργήσει ο χρήστης.
GET /group/{groupId}	Επιστρέφει τα δεδομένα μιας συγκεκριμένης ομάδας βάσει του groupId, μόνο εφόσον αυτή ανήκει στον χρήστη.

Πίνακας 4: Endpoints /group

4.1.2.1.4 Endpoints /campaign

Τα παρακάτω endpoints χρησιμοποιούνται για λειτουργίες που αφορούν τον campaign πόρο. Επιτρέπουν στον client να δημιουργήσει και να επεξεργαστεί τα campaigns του. Επίσης χρησιμοποιούνται και για την διαχείριση της κατάστασης του campaign. Αυτά τα endpoints χρησιμοποιούνται μόνο από αυθεντικοποιημένους χρήστες. Η υλοποίηση τους γίνεται από την κλάση CampaignController.

GET /campaign/{id}	Επιστρέφει τις λεπτομέρειες μιας συγκεκριμένης καμπάνιας με βάση το campaignId.
GET /campaign	Επιστρέφει όλες τις καμπάνιες του χρήστη σε μορφή λίστα, οι οποίες περιλαμβάνουν συνοπτικά στοιχεία.
POST /campaign/{id}/subscriber	Προσθέτει έναν συγκεκριμένο συνδρομητή στην καμπάνια. Δέχεται ως παράμετρο το id της καμπάνιας.
DELETE /campaign/{id}/subscriber/{subscriberId}	Αφαιρεί έναν συνδρομητή από την καμπάνια. Δέχεται ώς παράμετρο το id της καμπάνιας και το id του συνδρομητή.
POST /campaign/{id}/group	Προσθέτει ένα group συνδρομητών στην καμπάνια. Δέχεται ώς παράμετρο το id της καμπάνιας.
DELETE /campaign/{id}/group/{groupId}	Αφαιρεί ολόκληρο group από την καμπάνια. Δέχεται ώς παράμετρο το id της καμπάνιας και το id του group.
PUT /campaign/{id}/title	Θέτει ή ενημερώνει τον τίτλο της καμπάνιας. Δέχεται ώς παράμετρο το id της καμπάνιας.

PUT /campaign/{id}/sender	Ορίζει ή τροποποιεί τον αποστολέα (email sender) της καμπάνιας. Δέχεται ως παράμετρο το id της καμπάνιας.
PUT /campaign/{id}/email/subject	Ορίζει το subject του email της καμπάνιας. Δέχεται ώς παράμετρο το id της καμπάνιας.
PUT /campaign/{id}/email/content	Ορίζει το περιεχόμενο του email της καμπάνιας. Δέχεται ώς παράμετρο το id της καμπάνιας.
GET /campaign/{id}/email/content/spamscore	Επιστρέφει το spam-score του περιεχομένου της καμπάνιας. Δέχεται ώς παράμετρο το id της καμπάνιας.
GET /campaign/email	Χρησιμοποιείται για την βελτιστοποίηση του περιεχομένου ενός email. Δέχεται ώς είσοδο το περιεχόμενο του email, το όνομα του αποστολέα, και το θέμα, και επιστρέφει μια βελτιστοποιημένη μορφή του email.
POST /campaign/{id}/schedule	Αλλάζει την κατάσταση της καμπάνια σε SCHEDULED και ορίζει την ώρα έναρξης. Δέχεται ως παράμετρο το id της καμπάνιας.
PUT /campaign/{id}/execution/pause	Θέτει την καμπάνια σε κατάσταση παύσης (PAUSED), διακόπτοντας προσωρινά την εκτέλεση. Δέχεται ως παράμετρο το id της καμπάνιας.
PUT /campaign/{id}/execution/continue	Συνεχίζει την εκτέλεση μιας PAUSED καμπάνιας. Δέχεται ως παράμετρο το id της καμπάνιας.
PUT /campaign/{id}/execution/terminate	Τερματίζει οριστικά την καμπάνια, διακόπτοντας την αποστολή σε περίπτωση που είναι ενεργή. Δέχεται ως παράμετρο το id της καμπάνιας.

Πίνακας 5: Endpoints /campaign

4.1.2.1.5 Endpoints /resources

Τα παρακάτω endpoints χρησιμοποιούνται για λειτουργίες που αφορούν τον resources πόρο. Με τον όρο resources εννοούμε οτιδήποτε ανεβάζει ο χρήστης στο σύστημα μας, ώστε να το χρησιμοποιήσει στις καμπάνιες του. Αυτό μπορεί να είναι ένα PDF αρχείο, η μια εικόνα. Με τα παρακάτω endpoints ο χρήστης μπορεί να ανεβάζει, να διαγράφει η να βλέπει τα resources του. Αυτά τα endpoints χρησιμοποιούνται μόνο από αυθεντικοποιημένους χρήστες. Η υλοποίηση τους γίνεται από την κλάση ObjectStorageController.

POST /resources/upload	Επιτρέπει την αποστολή (upload) ενός ή περισσότερων αρχείων μέσω multipart/form-data.

GET /resources/files	Επιστρέφει λίστα όλων των αρχείων που έχει ανεβάσει ο αυθεντικοποιημένος χρήστης. Το κάθε αρχείο περιγράφεται με ObjectStorageFileDto, που περιλαμβάνει όνομα αρχείου και προσωρινό σύνδεσμο πρόσβασης (presigned URL).
DELETE /resources/files/{filename}	Διαγράφει ένα συγκεκριμένο αρχείο με βάση την filename παράμετρο, για τον χρήστη που έχει ανεβάσει το αρχείο.

Πίνακας 6: Endpoints /resources

4.1.2.1.6 Endpoint /track

Το παρακάτω endpoint χρησιμοποιείται για λειτουργία που αφορά την παρακολούθηση ανοίγματος των email. Αυτό το endpoint δεν απαιτεί ανθεντικοποιημένο χρήστη και η υλοποίηση του γίνεται από την κλάση TrackingController.

GET /track/{trackingLinkId}	Αυτό το endpoint καλείται από κάθε email που ανοίγεται από τον παραλήπτη. Όταν ο παραλήπτης ανοίγει το email, ο browser ή ο email-client του καλεί αυτό το endpoint με παράμετρο ένα μοναδικό trackingLinkId, και το back-end χρησιμοποιεί αυτό το μοναδικό id ώστε να διαπιστώσει ποιός παραλήπτης άνοιξε το email, και να ενημερώσει τα στατιστικά παρακολούθησης ανοίγματος. Το endpoint επιστρεφει μια εικόνα ενός pixel. Αυτό το endpoint είναι το μοναδικό που δεν ακολουθεί πλήρως το REST πρότυπο, και ο λόγος είναι ότι η μέθοδος HTTP είναι GET ενώ ο σκοπός του είναι να ενημερώσει ενα resource. Η χρήση όμως της μεθόδου GET είναι αναγκαία, καθώς φαινομενικά αυτό το endpoint πρέπει να χρησιμοποιείται για να σερβίρει μια εικόνα, και ο browser απαιτεί να χρησιμοποιεί GET μέθοδος.

Πίνακας 7: Endpoints /track

4.1.2.1.7 Endpoint /campaign-statistics

Το παρακάτω endpoint χρησιμοποιείται από χρήστες για να πάρουν δεδομένα στατιστικών σχετικά με καμπάνιες του. Αυτό το endpoint απαιτεί ανθεντικοποιημένο χρήστη και η υλοποίηση του γίνεται από την κλάση CampaignStatisticsController.

GET /campaign-statistics/{campaignId}	Επιστρέφει αναλυτικά στατιστικά σχετικά με την καμπάνια που αντιστοιχεί στην campaignID παράμετρο. Τα στατιστικά περιλαμβάνουν στοιχεία όπως το σύνολο των email της καμπάνιας, το πόσα παραδόθηκαν, πόσα ανοίχτηκαν και άλλα. άνοιξε το email, και να ενημερώσει τα στατιστικά παρακολούθησης ανοίγματος.

Πίνακας 8: Endpoints /campaign-statistics

4.1.2.1.8 Endpoint /openrate

Το παρακάτω endpoint χρησιμοποιείται από χρήστες για να πάρουν δεδομένα στατιστικών σχετικά με το άνοιγμα των email. Αυτό το endpoint απαιτεί ανθεντικοποιημένο χρήστη και η υλοποίηση του γίνεται από την κλάση OpenRateController.

GET /openrate/{campaignId}/subscribers	Επιστρέφει τους συνδρομητές που άνοιξαν τα email μιας συγκεκριμένης καμπάνιας, με βάση την παράμετρο campaignId. Η απάντηση είναι μια λίστα από subscriberId, η οποία αντιπροσωπεύει τους χρήστες που καταγράφηκαν να έχουν ανοίξει το email.

Πίνακας 9: Endpoints /openrate

4.1.2.2 Αυθεντικοποίηση χρηστών και ασφάλεια

Η ασφάλεια του API του συστήματος έχει υλοποιηθεί με την χρήση του Spring security framework, το οποίο λειτουργεί ως ενδιάμεσο στρώμα ασφαλείας μέσα στην εφαρμογή, παρεμβαίνοντας σε κάθε εισερχόμενο HTTP request. Τα εισερχόμενα HTTP request περνάνε μέσα από μια αλυσίδα φίλτρων τα οποία εφαρμόζουν την ασφάλεια του συστήματος μας.

Η μέθοδος αυθεντικοποίησης που επιλέξαμε να χρησιμοποιήσουμε είναι το JWT (Json Web Token). Το JWT προσφέρει το πλεονέκτημα ότι είναι ένας stateless τρόπος αυθεντικοποίησης ο οποίος δεν απαιτεί την αποθήκευση της κατάστασης του session στο backend, καθιστώντας το σύστημα πιο κλιμακούμενο και αποδοτικό, μειώνοντας σημαντικά την χρήσης της βάσης δεδομένων.

Όλη η λογική της αυθεντικοποίησης βρίσκεται μέσα στο πακέτο *configuration.security*. Η υλοποίηση έχει χωριστεί σε τέσσερις βασικές κλάσεις:

- ***JwtUtil****: Υπεύθυνη για τη δημιουργία, υπογραφή και επικύρωση των JWT. Καθορίζει τα claims (όπως το userId και τους ρόλους), την ημερομηνία λήξης και υπογράφει το token με HMAC SHA-256 χρησιμοποιώντας το μυστικό κλειδί της εφαρμογής.*
- ***JwtAuthenticationManager****: Αναλαμβάνει την επικύρωση του JWT κατά το αίτημα. Ελέγχει την εγκυρότητα του token, ανακτά το userId και φορτώνει τον χρήστη από τη βάση. Δημιουργεί το Authentication αντικείμενο με τους αντίστοιχους ρόλους.*
- ***JwtSecurityContextRepository****: Υλοποιεί την ανάκτηση του token από κάθε εισερχόμενο HTTP αίτημα, είτε από cookie είτε από το Authorization header. Προωθεί το token στο AuthenticationManager για επικύρωση και αποθηκεύει το authentication context.*
- ***RestSecurityConfig****: Ρυθμίζει το Spring Security framework, ορίζοντας τα φίλτρα ασφαλείας, την πολιτική CORS, τους κανόνες πρόσβασης στα endpoints, καθώς και το μηχανισμό authentication και security context repository.*

4.1.2.3 Ροή αυθεντικοποίησης

Η ροή της αυθεντικοποίησης ξεκινά με το login ενός χρήστη χρησιμοποιώντας το endpoint */user/login* στον UserController. Ο χρήστης αποστέλλει στο σώμα του αιτήματος (HTTP body) ένα UserLoginDto, το οποίο περιλαμβάνει το email, τον κωδικό πρόσβασης και μια σημαία που υποδεικνύει αν επιθυμεί παρατεταμένη σύνδεση (persistent login).

Το backend, μέσω της κλάσης UserService, επαληθεύει τα στοιχεία του χρήστη και, σε περίπτωση επιτυχούς ελέγχου, καλεί την κλάση JwtUtil για να δημιουργήσει ένα JSON Web Token (JWT). Στο JWT ορίζονται ως claims το μοναδικό αναγνωριστικό του χρήστη (userId) και ο ρόλος του (STANDARD_USER). Η διάρκεια ζωής του token καθορίζεται δυναμικά: έξι ώρες για προσωρινή σύνδεση και επτά ημέρες για παρατεταμένη. Το token υπογράφεται ψηφιακά με τον αλγόριθμο HAMAC SHA-256, χρησιμοποιώντας το μυστικό κλειδί (secret) που ορίζεται στις ρυθμίσεις της εφαρμογής. Τέλος, το JWT αποστέλλεται στον χρήστη ως ασφαλές HttpOnly cookie μέσω του header Set-Cookie, το οποίο κατασκευάζεται στον UserController και περιέχει το όνομα cookie auth_token.

Στα αιτήματα που ακολουθούν ο client αποστέλλει αυτόματα το JWT μέσα στο HttpOnly cookie με όνομα auth_token. Κατά την παραλαβή κάθε αιτήματος, η κλάση JwtSecurityContextRepository αναζητά το token αρχικά στο cookie auth_token και εναλλακτικά στο HTTP header Authorization (ως Bearer). Αν βρεθεί token, προωθείται στο JwtAuthenticationManager για επικύρωση.

Ο JwtAuthenticationManager ελέγχει την εγκυρότητα του token, επαληθεύοντας υπογραφή και ημερομηνία λήξης, και στη συνέχεια ανακτά το userId από το token. Τέλος, δημιουργεί ένα Authentication αντικείμενο με τους ρόλους του χρήστη. Το αντικείμενο αυτό αποθηκεύεται στο SecurityContect εξασφαλίζοντας ότι το σύστημα αναγνωρίζει τον χρήστη καθ' όλη τη διάρκεια του αιτήματος.

4.1.2.4 UserID custom annotation

Όταν ένα HTTP αίτημα περάσει επιτυχώς τα φίλτρα του Spring Security και ολοκληρωθεί η αυθεντικοποίηση, το αίτημα προωθείται στο επίπεδο των Controllers. Για να διευκολυνθεί η πρόσβαση στο αναγνωριστικό του αυθεντικοποιημένου χρήστη (userId) μέσα στα endpoints, έχει υλοποιηθεί το custom annotation **@UserID**.

Αυτό το annotation, σε συνδυασμό με την κλάση UserIdArgumentResolver, αναλαμβάνει την αυτόματη εξαγωγή του userId από το SecurityContext. Συγκεκριμένα, ο resolver ανιχνεύει τις παραμέτρους με το @UserID annotation και κατά την εκτέλεση του αιτήματος ανακτά το Authentication αντικείμενο από το ReactiveSecurityContextHolder. Από εκεί παίρνει το principal, που αντιστοιχεί στο userId που είχε αποθηκευτεί κατά την αυθεντικοποίηση μέσω JWT.

Αυτό το custom annotation προσφέρει το πλεονέκτημα ότι όλη η λογική της εξαγωγής του userID γίνεται σε ένα μοναδικό σημείο και έτσι αποφεύγεται η επανάληψη κώδικα. Επίσης βοηθάει στον διαχωρισμό των ευθυνών.

```
@PostMapping
fun initializeCampaign(@UserID userId: Int): Mono<CampaignDto> {
    return campaignService.createCampaign(userId) }
```

Πίνακας 10: Χρήση @UserID annotation

Στο παραπάνω παράδειγμα, η μέθοδος initializeCampaign στον CampaignController δέχεται ως παράμετρο το userId με τη χρήση του custom annotation @UserID. Αυτό επιτρέπει την αυτόματη εξαγωγή του αναγνωριστικού του αυθεντικοποιημένου χρήστη από το SecurityContext χωρίς να χρειάζεται επιπλέον κώδικας για ανάγνωση ή ανάλυση του JWT.

4.1.2.5 Διαχείρηση σφαλμάτων

Η διαχείριση των σφαλμάτων σε επίπεδο API (Controller) έχει υλοποιηθεί με την χρήση Controller Advice. Το Controller Advice είναι ένα χαρακτηριστικό του Spring Framework που επιτρέπει τον ορισμό καθολικής, επαναχρησιμοποιούμενης λογικής χειρισμού σφαλμάτων για όλα τα controllers μιας εφαρμογής. Στην πράξη, πρόκειται για μια συμπληρωματική κλάση που περιλαμβάνει συμβουλευτικές (advice) μεθόδους οι οποίες εφαρμόζονται αυτόματα σε όλους τους controllers χωρίς να χρειάζεται να επαναληφθούν.

Η κλάση η οποία υλοποιεί τη διαχείριση των σφαλμάτων είναι η CustomExceptionHandler, η οποία επισημαίνεται με το annotation @RestControllerAdvice. Μέσα σε αυτήν έχουν οριστεί μέθοδοι με το annotation @ExceptionHandler, οι οποίες "παγιδεύουν" συγκεκριμένες κατηγορίες εξαιρέσεων (Exceptions) και επιστρέφουν δομημένες HTTP απαντήσεις με κατάλληλο status code και μήνυμα σφάλματος.

Κάθε μέθοδος που είναι επισημασμένη με το annotation @ExceptionHandler αφορά τη διαχείριση ενός συγκεκριμένου τύπου εξαίρεσης (exception). Στην εφαρμογή μας, μέσω της κλάσης CustomExceptionHandler, γίνεται η καθολική διαχείριση των παρακάτω εξαιρέσεων:

- ***ObjectStorageException****: Επιστρέφει status code 503 Service Unavailable όταν παρουσιαστεί αποτυχία επικοινωνίας με το object storage.*
- ***NotFoundException****: Επιστρέφει status code 404 Not Found όταν ζητηθεί πόρος (π.χ. χρήστης, group, συνδρομητής) που δεν υπάρχει.*
- ***ConflictException****: Επιστρέφει status code 409 Conflict όταν γίνει απόπειρα για λειτουργία που έρχεται σε σύγκρουση με την τρέχουσα κατάσταση (π.χ. εγγραφή υπάρχοντος email).*
- ***IllegalArgumentException****: Επιστρέφει 400 Bad Request για λανθασμένες παραμέτρους ή δεδομένα που αποστέλλονται στον server.*
- ***Exception*** *(γενική): Επιστρέφει 500 Internal Server Error για κάθε απρόβλεπτη εξαίρεση, με προαιρετική εμφάνιση stacktrace (χρήσιμο μόνο για περιβάλλον ανάπτυξης).*

Με αυτόν τον τρόπο επιτυγχάνεται συνεπής και καθαρή διαχείριση λαθών σε όλα τα endpoints, χωρίς επαναλαμβανόμενο κώδικα στους controllers.

4.1.3 Αλγόριθμος βελτιστοποίησης Email

Ο αλγόριθμος βελτιστοποίησης του περιεχομένου των email είναι ένα από τα βασικά χαρακτηριστικά του Core module. Η βελτιστοποίηση βασίζεται σε δύο εξωτερικές υπηρεσίες που χρησιμοποιούνται από το backend: την υπηρεσία αξιολόγησης spam (SpamAssassin) και ένα γλωσσικό μοντέλο το οποίο χρησιμοποιείται για την παραγωγή του βελτιστοποιημένου και αναδιατυπωμένου περιεχομένου.

4.1.3.1 Χρήση API βελτιστοποίησης αλγορίθμου

Ο αλγόριθμος βελτιστοποίησης προσφέρεται μέσω του endpoint */campaign/email/optimized*. Αυτό το endpoint παίρνει ως είσοδο ένα EmailOptimizationDto, το οποίο περιλαμβάνει τρία πεδία: το όνομα του αποστολέα (from), το θέμα του email (subject) και το περιεχόμενο του email (content) σε μορφή Base64. Το αίτημα αποστέλλεται με την HTTP POST μέθοδο.

Στην συνέχεια, η απάντηση είναι ένα OptimizedEmailDto, το οποίο περιλαμβάνει το βελτιστοποιημένο περιεχόμενο σε Base64, το spam score πριν τη βελτιστοποίηση (spamScoreBefore) και το νέο spam score μετά τη βελτίωση (spamScoreAfter). Αυτό επιτρέπει στον χρήστη να συγκρίνει τις δύο τιμές και να διαπιστώσει τη μείωση της πιθανότητας το email να χαρακτηριστεί ως spam.

4.1.3.2 Υπηρεσία SpamAssasin

Για να μπορεί το περιεχόμενο ενός email να βελτιστοποιηθεί, πρέπει πρώτα να εκτιμηθεί η πιθανότητα να χαρακτηριστεί ως ανεπιθύμητο (spam). Για τον σκοπό αυτό, η πλατφόρμα χρησιμοποιεί την υπηρεσία SpamAssassin, ένα λογισμικό ανοιχτού κώδικα που βασίζεται σε κανόνες και αναλύει το περιεχόμενο των email, αποδίδοντας ένα spam score καθώς και ένα αναλυτικό spam report.

Η επικοινωνία με την εξωτερική υπηρεσία του SpamAssasin υλοποιείται από την κλάση SpamAssassinService, η οποία βρίσκεται στο package *service*. Η αρμοδιότητα αυτής της κλάσης είναι η επικοινωνία με την υπηρεσία του SpamAssasin μέσω HTTP request και την αποθήκευση των αποτελεσμάτων στην βάση δεδομένων.

Για να έχουμε όσο πιο ακριβή αποτελέσματα γίνεται από το SpamAssasin, η υπηρεσία δημιουργεί ένα πλήρες MIME αντικείμενο μέσω της βιβλιοθήκης jakarta.mail, το οποίο περιλαμβάνει τόσο την κειμενική (plain text) όσο και την HTML μορφή του email. Το MIME μήνυμα μετατρέπεται σε raw string μορφή και αποστέλλεται στο SpamAssassin API για ανάλυση. Με αυτόν τον τρόπο, το SpamAssassin λαμβάνει ένα πλήρως μορφοποιημένο email, γεγονός που αυξάνει την αξιοπιστία του spam score που επιστρέφει.

Η υπηρεσία SpamAssasin χρησιμοποιείται τόσο από τον αλγόριθμο βελτιστοποίησης, όσο και από την αλγόριθμο δημιουργίας καμπάνιας. Κάθε φορά που ένας χρήστης οριστικοποιεί μια καμπάνια, το περιεχόμενο του email βαθμολογείται από το SpamAssasin και η βαθμολογία αποθηκεύεται στην βάση για μελλοντική αναφορά.

4.1.3.3 Υπηρεσία LLM και OpenAPI specification

Την επικοινωνία και την διαχείριση του LLM την υλοποιεί η κλάση LlmService. Αξιοποιεί το Spring AI framework[35] και συγκεκριμένα την κλάση ChatClient, η οποία επιτρέπει την επικοινωνία με το LLM μέσω OpenAPI Spec. Η χρήση του ChatClient και του OpenAPI Spec[36] επιτρέπει την εύκολη αντικατάσταση του LLM διατηρώντας την λογική της κλάσης ανεξάρτητη από την υλοποίηση του LLM μοντέλου.

Επίσης, αυτή η κλάση είναι υπεύθυνη για την διαχείριση των συνεδριών συνομιλίας με το LLM (Chat session) καθώς και της παραμέτρου temperature του LLM.

4.1.3.4 Υλοποίηση του αλγόριθμου βελτιστοποίησης

Το πρώτο βήμα του αλγορίθμου βελτιστοποίησης είναι η αξιολόγηση του περιεχομένου του email που πρόκειται να βελτιστοποιηθεί. Με την αξιολόγηση του email παράγεται ένα report το οποίο περιέχει ένα spam-score και ένα report που αναλύει το τί προκαλεί τη αυξηση του spam-score.

Στην συνέχεια ακολουθεί το δεύτερο στάδιο του αλγορίθμου. Στο δεύτερο στάδιο δημιουργείται ένα νέο chat-session με το LLM και δημιουργείται το κατάλληλο prompt. Το prompt αποτελείται από τρία κύρια συστατικά:

- *Το system prompt: μια σταθερή οδηγία προς το μοντέλο, η οποία ορίζει τον γενικό σκοπό (βελτιστοποίηση περιεχομένου με στόχο τη μείωση spam χαρακτηριστικών).*

- *Το spam report: το αποτέλεσμα της αξιολόγησης από την υπηρεσία SpamAssassin, το οποίο επισημαίνει τους λόγους για τους οποίους το email θεωρείται spam.*
- *Το αρχικό HTML περιεχόμενο του email: το οποίο πρέπει να επαναδιατυπωθεί με τρόπο ώστε να μειωθεί ο spam score, διατηρώντας όμως το νόημα και τη δομή του.*

Το τρίτο στάδιο του αλγορίθμου είναι ο υπολογισμός του κατάλληλου temperature. Η μεταβλητή temperature υπολογίζεται **δυναμικά** για κάθε ξεχωριστή βελτιστοποίηση. Η συνάρτηση υπολογισμού temperature βασίζεται στο spam-score, και η λογική είναι ότι σε περιπτώσεις με υψηλό spam-score, ορίζουμε υψηλό temperature, ώστε να δώσουμε την ελευθερία στο μοντέλο να προβεί σε πιο δημιουργικές και εκτεταμένες αλλαγές στο περιεχόμενο. Αντιθέτως, όταν το spam-score είναι ήδη χαμηλό, χρησιμοποιείται ένα μικρότερο temperature, ώστε το μοντέλο να διατηρήσει σε μεγαλύτερο βαθμό το αρχικό περιεχόμενο, περιορίζοντας τις μεταβολές.

```
private fun calculateOptimalTemperature(spamScore: Double): Double {
  val multiplier = 0.5
  val adjustedScore = (ln(spamScore + 1) / ln(6.0))
  return (0.2 + adjustedScore * (multiplier)).coerceIn(0.2, 0.9)
}
```

Πίνακας 11: Συνάρτηση υπολογισμού temperature

Η συνάρτηση calculateOptimalTemperature δημιουργήθηκε με βάση τη λογική του trial and error, καθώς δεν υπάρχει κάποιος προκαθορισμένος μαθηματικός τύπος που να προσδιορίζει με ακρίβεια τη βέλτιστη τιμή temperature για ένα LLM σε σχέση με ένα spam score. Αρχικά, παρατηρήθηκαν διαφορετικά spam scores και η απόδοσή τους σε σχέση με τις μεταβολές στο περιεχόμενο που πρότεινε το μοντέλο. Στη συνέχεια, εφαρμόστηκε ένας λογαριθμικός μετασχηματισμός (με βάση το ln(spamScore + 1)), ώστε να μετριαστεί η επίδραση πολύ υψηλών τιμών spam score και να επιτευχθεί ομαλή αύξηση του temperature.

Η επιλογή της βάσης του λογαρίθμου (ln(6.0)) και του multiplier 0.5 προέκυψε εμπειρικά, με στόχο να κλιμακώνονται οι τιμές του temperature σε ένα εύρος μεταξύ 0.2 και 0.9, το οποίο κρίθηκε κατάλληλο μετά από πειραματισμούς. Τέλος, η χρήση της συνάρτησης coerceIn(0.2, 0.9) διασφαλίζει ότι η τελική τιμή παραμένει εντός λογικών ορίων για την αποδοτική λειτουργία του LLM.

Το τέταρτο και τελευταίο στάδιο του αλγορίθμου είναι η αποστολή του prompt στο LLM και η λήψη της βελτιστοποιημένης εκδοχής του email. Αφού ληφθεί η απάντηση, γίνεται καθαρισμός του κειμένου από ειδικά tags, αν υπάρχουν (όπως τα <think> tags), και εξαγωγή μόνο του χρήσιμου περιεχομένου. Το βελτιστοποιημένο email στη συνέχεια επαναξιολογείται με την υπηρεσία SpamAssassin, ώστε να καταγραφεί και να συγκριθεί το νέο spam score με το αρχικό. Αυτό το στάδιο επιτρέπει τον αντικειμενικό έλεγχο της αποτελεσματικότητας της βελτιστοποίησης.

4.1.3.5 System prompts και κανόνες

Κατά τη διαδικασία της βελτιστοποίησης email, ένα από τα πιο σημαντικά στοιχεία είναι η σύνθεση του system prompt που παρέχεται στο LLM. Το system prompt λειτουργεί ως οδηγία προς το μοντέλο, καθορίζοντας με σαφήνεια τον στόχο της επεξεργασίας. Στην περίπτωση της συγκεκριμένης

υλοποίησης, ο στόχος είναι η τροποποίηση του περιεχομένου ενός HTML email με τέτοιο τρόπο ώστε να μειώνεται το spam score, χωρίς όμως να χαθεί η ουσία του αρχικού περιεχομένου του email.

Το περιεχόμενο του system prompt καθώς και οι κανόνες που το συνοδεύουν (π.χ. να μην αλλάζει το νόημα, να διατηρούνται οι σύνδεσμοι, να αποφεύγεται υπερβολική μορφοποίηση) καθορίστηκαν εμπειρικά, μετά από δοκιμές και αξιολόγηση πραγματικών αποτελεσμάτων. Μέσα από μια διαδικασία trial and error, δοκιμάστηκαν διάφορες εκδοχές του prompt, με διαφορετική φρασεολογία, σειρά στοιχείων και περιορισμούς, μέχρι να καταλήξουμε στην πιο αποδοτική και διατύπωση που παράγει χαμηλότερο spam score χωρίς να αλλάζει υπερβολικά πολύ το αρχικό email.

```
You are an expert email optimizer.

Your task:

Input: Raw HTML of an email. SpamAssassin report output for this email.

Output: Modified HTML, strictly with HTML tags only.
```

Πίνακας 12: System prompt βελτιστοποίησης email

```
Rules:

Do not change the HTML structure or any links (href, src, etc.).

Do fix typos and grammatical errors in the text content.

Do modify wording to lower the spam score, focusing especially on SpamAssassin's reported issues.

Do improve user experience (smoother, clearer text, less aggressive language).

Do modify spammy or trigger words (e.g., “FREE”, “GUARANTEED”, “CLICK HERE”, “ACT NOW”, excessive caps or exclamations).

Do not add or remove links, images, or major structural elements.

Only modify text content where necessary to address SpamAssassin flags and common spam triggers.

Final output: Only clean, valid HTML. No explanations, no extra comments.
```

Πίνακας 13: Promted rules βελτιστοποίησης email

4.1.4 Minio object storage

Η εφαρμογή, δίνει την δυνατότητα στους χρήστες να μπορούν να εμπλουτίζουν τα email που δημιουργούν με εικόνες, η ακόμα και να επισυνάπτουν στα email αρχεία, όπως για παράδειγμα PDF αρχεία.

Για την διαχείριση αυτών τον αρχείων και εικόνων κάθε χρήστη, το Core module βασίζεται στην εξωτερική υπηρεσία MinIO, το οποίο είναι μια open-source πλατφόρμα αποθήκευσης αντικειμένων σχεδιασμένη για υψηλή απόδοση, επεκτασιμότητα και αξιοπιστία.

Η διαχείριση της επικοινωνίας με το MinIO γίνεται μέσω της κλάσης ObjectStorageService που είναι υπεύθυνη για την αποθήκευση, ανάκτηση και διαγραφή αρχείων χρηστών. Όταν ένας χρήστης κάνει upload αρχείου για πρώτη φορά στην πλατφόρμα, η κλάση ελέγχει αν υπάρχει ήδη ένα bucket για τον συγκεκριμένο χρήστη. Αν δεν υπάρχει, δημιουργεί αυτόματα το bucket με όνομα που προκύπτει από το hash του userID και ενός μυστικού κλειδιού.

Στη συνέχεια, το αρχείο αποθηκεύεται μέσα στο bucket του χρήστη με ένα obfuscated όνομα αρχείου, το οποίο δημιουργείται μέσω hashing του αρχικού ονόματος και του μυστικού κλειδιού, ώστε να διασφαλίζεται η ιδιωτικότητα και η αποφυγή συγκρούσεων ονομάτων. Παράλληλα, το όνομα του αρχείου καθαρίζεται (sanitize) ώστε να περιέχει μόνο επιτρεπτούς χαρακτήρες.

Η κλάση ObjectStorageService διαχειρίζεται επίσης την ανάκτηση των αρχείων, επιστρέφοντας στον χρήστη τα URL για πρόσβαση στα αποθηκευμένα αρχεία. Αυτά τα URL τα προσθέτει αυτόματα το front-end στο περιεχόμενο του email. Έτσι, όταν ένας παραλήπτης ανοίγει ένα email, όλα τα αιτήματα που αφορούν εικόνες και αρχεία γίνονται στο MinIO, και δεν επιβαρύνεται η υπηρεσία του backend μας.

```
http(s)://<minio-host>/<bucket-name>/<object-name>
```

Πίνακας 14: MinIO URL

Τα URL όλων των αρχείων και εικόνων όλων των χρηστών είναι αναγκαστικά δημόσια και προσβάσιμα από τον καθένα. Για να διασφαλίσουμε την ασφάλεια και την ιδιωτικότητα των δεδομένων, τα ονόματα των buckets και των αρχείων υποβάλλονται σε hashing, συνδυάζοντας το userID ή το αρχικό όνομα αρχείου με ένα μυστικό κλειδί. Αυτό δημιουργεί μοναδικά και μη προβλέψιμα ονόματα, αποτρέποντας την τυχαία ή κακόβουλη πρόβλεψη ή κατασκευή URL. Με αυτή την τεχνική, η πρόσβαση στα αρχεία παραμένει δημόσια, αλλά εξαιρετικά δύσκολη η πρόσβαση μέσω τυχαίας εύρεσης URL.

4.2 Executor module

Το executor module είναι το κομμάτι της εφαρμογής που είναι υπεύθυνο για την εκτέλεση των email campaign που προγραμματίζουν οι χρήστες της εφαρμογής. Σε αυτό το module υλοποιούνται οι προγραμματισμένες διεργασίες, οι οποίες εκτελούνται ανά τακτά χρονικά διαστήματα, και έχουν ως αρμοδιότητα να κάνουν την αρχικοποίηση της εκτελούμενης καμπάνιας, να δημιουργούν το περιεχόμενο των email που θα αποσταλούν, να προσωποποιούν το κάθε email αναλόγως τον παραλήπτη, και τέλος να βάζουν τα έτοιμα email στην ούρα που περιέχει τα email προς αποστολή.

4.2.1 Δομή του Executor module

Η λογική του executor module είναι χωρισμένη σε τρία διαφορετικά στρώματα (layers) και το κάθε στρώμα έχει ξεχωριστές αρμοδιότητες.

Το πάνω στρώμα είναι το στρώμα των schedulers. Εκεί υπάρχει όλη η λογική που αφορά τις προγραμματισμένες εργασίες, και τις παραμέτρους εκτέλεσης τους. Το service στρώμα είναι το

επόμενο, το οποίο περιέχει όλη την επιχειρησιακή λογική των διεργασιών που εκτελούνται από το executor module. Το τελευταίο στρώμα είναι το repository, το οποίο είναι υπεύθυνο για την επικοινωνία με την βάση δεδομένων, και λειτουργεί ως abstraction πάνω από την βάση δεδομένων.

Όσον αφορά την μεταφορά πληροφοριών, το executor module ακολουθεί ακριβώς την ίδια λογική με το core module. Γίνεται χρήση dto τα οποία παρέχουν ένα σαφή ορισμό της δομής των πληροφοριών που ανταλλάσσονται μεταξύ των στρωμάτων. Με τον ίδιο ακριβώς τρόπο ορίζεται και η μεταφορά πληροφοριών μεταξύ των repository και της βάσεων δεδομένων, με την χρήση entities, τα οποία αναπαριστούν τους πίνακες της βάσης.

Τα κομμάτια και στρώματα του module χωρίζονται με σαφήνεια στον κώδικα με την χρήση των scheduler, service, repository, dto και entity πακέτων. Ο διαχωρισμός αυτός προσφέρει έναν ξεκάθαρο τρόπο διαχωρισμού αρμοδιοτήτων και κάνει τον κώδικα πιο κατανοητό, και πιο εύκολα επεκτάσιμο.

4.2.2 Διεργασία αρχικοποίησης καμπάνιας

Η διεργασία που αρχικοποιεί την καμπάνια ενός χρήστη υλοποιείται από την μέθοδο *initializeCampaignExecution* και είναι προγραμματισμένη να εκτελείται ανά εξήντα δευτερόλεπτα.

Το πρώτο βήμα του αλγορίθμου είναι να επιλέξει πιά καμπάνια θα αρχικοποιήσει. Αυτό γίνεται με ένα ερώτημα στην βάση όπου ζητάει μία καμπάνια που είναι σε κατάσταση SCHEDULED και έχει ώρα έναρξης πριν από την τρέχουσα ώρα. Αν παραπάνω από μία καμπάνιες πληρούν τις παραπάνω προϋποθέσεις, τότε η διεργασία επιλέγει την καμπάνια με την πιο προγενέστερη ώρα έναρξης. Στην συνέχεια, ο αλγόριθμος επιχειρεί την αλλαγή της κατάστασης της καμπάνιας σε RUNNING. Αν η κατάσταση της καμπάνιας αλλάξει επιτυχώς στην βάση, τότε ο αλγόριθμος θεωρεί την καμπάνια ως επιλεγμένη και προχωράει στο επόμενο βήμα. Αν η αλλαγή της κατάστασης σε RUNNING δεν γίνει επιτυχώς, τότε αυτό σημαίνει ότι κάποιο άλλο instance του executor module έχει αναλάβει την εκτέλεση της συγκεκριμένης καμπάνιας, και ο αλγόριθμος τερματίζει εκεί.

Το δεύτερο βήμα του αλγορίθμου αφορά την επιλογή όλων των μοναδικών παραληπτών που θα λάβουν email από την επιλεγμένη καμπάνια. Με το κατάλληλο ερώτημα στην βάση ο αλγόριθμος παίρνει μια λίστα που περιέχει το σύνολο των Subscribers οι οποίοι θα λάβουν email. Επειδή ένας συγκεκριμένος Subscriber μπορεί να υπάρχει σε δύο ή και παραπάνω Subscriber groups τα οποία έχουν επιλεχθεί για την καμπάνια, χρησιμοποιείται η λέξη κλειδί DISTINCT στο ερώτημα στην βάση, ώστε ένας subscriber να μην υπάρχει πάνω από μία φορά στην λίστα των παραληπτών.

Εφόσον ο αλγόριθμος έχει την λίστα με όλους τους παραλήπτες της καμπάνιας, προχωράει στο τρίτο βήμα, το οποίο αφορά την αρχικοποίηση όλων τον email που πρόκειται να σταλούν. Για κάθε Subscriber, ο αλγόριθμος εισάγει στον πίνακα *email_outgoing_event* μια σειρά ή οποία έχει το id του Subscriber, το id της καμπάνιας και την σημαία *is_queues* ως *false*. Η εισαγωγή των email σε αυτόν τον πίνακα είναι μέρος της αρχικοποίησης της καμπάνιας, και σηματοδοτεί ότι αυτό το email έχει αρχικοποιηθεί και εκκρεμεί η δημιουργία και η αποστολή του.

Καθ’ όλη τη διάρκεια του αλγορίθμου, οι λειτουργίες εκτελούνται εντός του ίδιου transaction, ώστε να εξασφαλιστεί η ατομικότητα της διεργασίας. Αυτό σημαίνει ότι είτε όλες οι αλλαγές θα ολοκληρωθούν επιτυχώς και θα αποθηκευτούν στη βάση, είτε, σε περίπτωση αποτυχίας, θα γίνει rollback και δεν θα εφαρμοστεί καμία τροποποίηση. Με αυτόν τον τρόπο, η αρχικοποίηση της καμπάνιας γίνεται με συνέπεια, αποτρέποντας περιπτώσεις ατελούς ή διπλής εκτέλεσης σε κατανεμημένα περιβάλλοντα.

```
function initializeCampaignForExecution(){

     campaign = find scheduled campaign with start_time <= now, ordered
by start_time ASC

     if no campaign found: return

    start transaction:
        rowsUpdated = try to set campaign status to RUNNING

        if rowsUpdated == 0:
            raise error "Campaign already RUNNING"

        subscribers = get distinct subscribers for this campaign
        for each subscriber:
            create EmailOutgoingEventEntity (queued = false)
            save it

    commit transaction

}
```

Πίνακας 15: Ψευδοκώδικας initializeCampaignExecution

4.2.3 Διεργασία δημιουργίας και αποστολής email

Η διεργασία αυτή υλοποιείται από την μέθοδο *constructAndQueueMail* και είναι προγραμματισμένη να εκτελείται ανά 3 λεπτά. Έχει μέγιστο χρόνο εκτέλεσης τα 3 λεπτά. Μετά το πέρας αυτής της προθεσμίας η εκτέλεση σταματά. Κατά το διάστημα των τριών λεπτών της εκτέλεσης, ο αλγόριθμος επιλέγει σειριακά email προς αποστολή, χτίζει το προσωποποιημένο περιεχόμενο τους και τα δημοσιοποιεί στην ουρά που περιέχει τα email προς αποστολή.

Το πρώτο βήμα του αλγορίθμου κάνει ερώτημα στην βάση δεδομένων και επιλέγει μια σειρά από τον πίνακα email_outgoing_event η οποία έχει την σημαία *is_queued* ως *false*, έχει την σημαία *is_locked* ως *false* και έχει ως *campaign_id* ένα campaign που έχει κατάσταση ως RUNNING. Οι παραπάνω προϋποθέσεις σιγουρεύουν ότι η σειρά που θα επιλεχθεί, αφορά ένα email που δεν έχει σταλεί ακόμα (is_queued=false), ότι το email δεν επεξεργάζεται από άλλο νήμα η instance του executor module (is_locked=false) και ότι η καμπάνια στην οποία ανήκει το email είναι δεν έχει ακυρωθεί/σταματήσει από τον χρήστη (RUNNING). Εφόσον λοιπόν επιλέξει ο αλγόριθμος μία σειρά από την βάση δεδομένων, στην συνέχεια επιχειρεί να θέσει στην επιλεγμένη σειρά, την σημαία is_locked ως true, στην βάση δεδομένων. Αν το κάνει επιτυχώς, τότε η σειρά έχει επιλεχθεί.

Στο δεύτερο βήμα του αλγορίθμου, γίνονται τα κατάλληλα ερωτήματα στην βάση ώστε να λάβει ο αλγόριθμος τις πληροφορίες που χρειάζεται σχετικά με την καμπάνια και τον Subscriber. Πιο

συγκεκριμένα, χρησιμοποιώντας το campaign_id και το subscriber_id που υπάρχουν στην επιλεγμένη σειρά (βήμα 1), ο αλγόριθμος ζητάει από την βάση το περιεχόμενο του email της καμπάνιας, καθώς και της πληροφορίες που αφορούν τον Subscriber, όπως το email, το όνομα και επώνυμο του.

Στο τρίτο βήμα, ο αλγόριθμος κάνει τις απαραίτητες τροποποιήσεις στο περιεχόμενο του email, ώστε να το προσωποποιήσει αναλόγως τον παραλήπτη. Πιο συγκεκριμένα, αντικαθιστά μέσα στο περιεχόμενο του email, τις λέξεις κλειδιά *{firstname}* και *{lastname}* με το όνομα και το επώνυμο του Subscriber. Επίσης, δημιουργεί ένα μοναδικό URL, το οποίο θα χρησιμοποιηθεί για να διαπιστωθεί το αν θα ανοίξει ο παραλήπτης το email. Το URL αυτό περιέχει ένα τυχαίο token, το οποίο εισάγεται στην βάση δεδομένων στον πίνακα *email_tracking*, μαζί με το *id* του Subscriber και το id της καμπάνιας. Τέλος αυτό το URL εισάγεται στο τέλος του tag του email, ως εικόνα.

Το τέταρτο βήμα, εφόσον το περιεχόμενο του email είναι έτοιμο, είναι να γίνει η επιλογή της κατάλληλης ουράς, που θα δημοσιοποιηθεί το email. Κάθε ενεργός mail server έχει την δική του ξεχωριστή ουρά στον message broker. Συνεπώς, η επιλογή της κατάλληλης ουράς είναι στην ουσία η επιλογή του κατάλληλου mail server. Ο αλγόριθμος κάνει την επιλογή, βασιζόμενος στο spam score της καμπάνιας, αλλά και το ιστορικό του χρήστη. Ο αλγόριθμος επιλογής mail server αναλύεται στην παρακάτω υποενότητα.

Αφού το email έχει προσωποποιηθεί, εμπλουτιστεί με το tracking pixel και αποθηκευτεί στη βάση, το τελευταίο βήμα της διεργασίας είναι η δημιουργία και δημοσίευση ενός PublishedMailDto αντικειμένου στην επιλεγμένη ουρά αποστολής.

Το αντικείμενο PublishedMailDto περιλαμβάνει τα εξής πεδία:

- ***uniqueId****: Ένα μοναδικό UUID που ταυτοποιεί την αποστολή του συγκεκριμένου email.*
- ***title****: Το θέμα (subject) του email.*
- ***sender****: Η email διεύθυνση αποστολέα.*
- ***receiverEmail****: Η email διεύθυνση του παραλήπτη.*
- ***mailContentBase64****: Το πλήρες HTML περιεχόμενο του email σε Base64 μορφή.*

Το αντικείμενο μετατρέπεται σε JSON string και δημοσιεύεται στην ουρά. Αυτό το μήνυμα θα ληφθεί αργότερα από το Publisher Module, το οποίο θα το αποκωδικοποιήσει και θα εκτελέσει την αποστολή του email μέσω του κατάλληλου mail server.

4.2.3.1 Αλγόριθμος επιλογής Mail-server

Η αλγόριθμος υλοποιείται από την κλάση MailServerSelectionService, και λαμβάνει ως είσοδο το userId ενός χρήστη και, αξιοποιώντας τις τελευταίες μετρήσεις spam από τις καμπάνιες του, προσπαθεί να βρει τον καταλληλότερο mail server για να στείλει τα επόμενα emails. Ο αλγόριθμος συνδυάζει ιστορικά δεδομένα spam με όρια/χαρακτηριστικά που έχουν οριστεί για κάθε server.

- Ανάκτηση των 5 πιο πρόσφατων καμπανιών του χρήστη. Από αυτές τις 5 καμπάνιες παίρνει το spam score της κάθε μίας και αν ο χρήστης έχει λιγότερο από 5 καμπάνιες, τότε η πεντάδα ολοκληρώνεται με την προεπιλεγμένη τιμή 2.5 ως spam score.
- Από τα ανακτημένα 5 spam score, υπολογίζεται η διάμεση τιμή. Με βάση αυτή την τιμή θα γίνει επιλογή του mail-server.

- Ανακτώνται από την βάση δεδομένων, όλοι οι mail-server που δεν είναι blacklisted, και ταυτόχρονα έχουν μεγαλύτερο spam-threshold από την υπολογισμένη μέση τιμή του προηγούμενου βήματος.
- Οι ανακτημένοι mail server κατατάσσονται με δύο κριτήρια:
- **Spam threshold** (αύξουσα σειρά): Προτιμώνται οι servers με χαμηλότερο threshold γιατί είναι πιο "αξιόπιστοι".
- **Open rate (φθίνουσα σειρά):** Αν δύο servers έχουν ίδιο threshold, επιλέγεται αυτός με το υψηλότερο ποσοστό ανοίγματος email, που δείχνει καλύτερη απόδοση στην πράξη.
- Ο πρώτος mail-server που είναι στην λίστα, είναι αυτός που επιλέγεται για να την αποστολή της καμπάνιας.

Υλοποιημένος είναι επίσης και ένας fallback μηχανισμός, Σε περίπτωση που κανένας mail server δεν πληροί τα κριτήρια για επιλογή, τότε επιλέγεται ο mail-server με το υψηλότερο spam threshold που δεν είναι blacklisted.

Τέλος, το αποτέλεσμα του αλγόριθμου αποθηκεύεται σε μια cache, ώστε να χρειαστεί να εκτελεστεί μόνο μία φορα για κάθε καμπάνια, κατά την δημιουργία του πρώτου email. Κατά την δημιουργία των επόμενων email της καμπάνιας, γίνεται αναζήτηση στην cache, και αν υπάρχει ήδη επιλεγμένος mail server, δεν ξαναεκτελείται ο αλγόριθμος.

4.2.4 Διεργασία τερματισμού καμπάνιας

Η διεργασία αυτή υλοποιείται από την μέθοδο *updateCampaignStatusToEnded* και είναι προγραμματισμένη να εκτελείται ανά 1 λεπτό. Σκοπός αυτής της διεργασίας είναι να αλλάζει την κατάσταση των καμπανιών που έχουν ολοκληρωθεί σε ENDED.

Ο αλγόριθμος αυτής της διεργασίας, ξεκινά κάνοντας ένα ερώτημα στην βάση δεδομένων επιλέγοντας την καμπάνια που είναι σε κατάσταση RUNNING, και ο χρόνος έναρξης της είναι ο πιο προγενέστερος (Εκτελείται την περισσότερη ώρα).

Στην συνέχεια, γίνεται ένα ακόμη ερώτημα στην βάση δεδομένων, ώστε να διαπιστωθεί το πόσα email εκκρεμούν ακόμα για αποστολή για την συγκεκριμένη επιλεγμένη καμπάνια. Το πόσα email εκκρεμούν ακόμα διαπιστώνεται από τον πίνακα email_outgoing_event και την στήλη is_queued. Αν δεν υπάρχει καμία σειρά στην βάση δεδομένων με campaign_id το id της επιλεγμένης καμπάνιας, και is_queued ως false, τότε αυτό σημαίνει ότι όλα τα email έχουν σταλεί, και η καμπάνια μπορεί να αλλάξει σε ENDED.

Σε αντίθετη περίπτωση, αν υπάρχει έστω και μία σειρά στην βάση με is_queued ως false, τοτε ο αλγόριθμος δεν αλλάζει την κατάσταση της καμπάνιας και τερματίζει.

4.2.5 Διεργασία αποστολής email κατά την εγγραφή χρήστη

Η διεργασία αυτή υλοποιείται από την μέθοδο *sendAccountConfirmationMails* και είναι προγραμματισμένη να εκτελείται ανά 15 δευτερόλεπτα. Σκοπός αυτής της διεργασίας είναι να στέλνει τα email τα οποία αφορούν την εγγραφή χρηστών στην πλατφόρμα και την επιβεβαίωση των email που όρισαν στα στοιχεία εγγραφής τους.

Αρχικά, κατά την εγγραφή ενός χρήστη, το core module αποθηκεύει τον χρήστη στον πίνακα user της βάσης δεδομένων, και θέτει το πεδίο status ως *PENDING_ACTIVATION*. Αυτό το πεδίο σηματοδοτεί ότι αυτός ο χρήστης δεν έχει επιβεβαιώσει ακόμα τον λογαριασμό του. Στην συνέχεια, εισάγει στον

πίνακα *user_account_confirmation* μια εγγραφή, η οποία περιέχει ένα μοναδικό token, το id του χρήστη, το πεδίο *email_status* ως *NOT_SENT* και ένα πεδίο *email_send_timestamp* που αρχικοποιείται ως κενό.

Ο αλγόριθμος λοιπόν της διεργασίας ξεκινά κάνοντας ένα ερώτημα στην βάση δεδομένων, και παίρνοντας όλες τι σειρές από τον πίνακα *user_account_confirmation*, που έχουν το πεδίο status ως *NOT_SENT*. Μετά, για κάθε μία από τις επιλεγμένες σειρές, θέτει το status ως *SENDING*, και αν τεθεί επιτυχώς η αλλαγή του status, τότε ο αλγόριθμος προχωρά στο επόμενο βήμα.

Συνεχίζοντας, ο αλγόριθμος, κάνει ερώτημα στην βάση δεδομένων και παίρνει τις πληροφορίες του χρήστη στο οποίο πρέπει να στείλει το email επιβεβαίωσης. Μετά, χτίζει το περιεχόμενο του email, και προσθέτει το μοναδικό URL, που περιέχει το μοναδικό token που υπάρχει μέσα στον πίνακα *user_account_confirmation*.

Τέλος, δημιουργεί ένα αντικείμενο τύπου *PublishedMailDto* το οποίο περιέχει ως παραλήπτη το email του χρήστη, και το περιεχόμενο του email με το URL επιβεβαίωσης. Αυτό το αντικείμενο σειριοποιείται σε μορφή String και δημοσιοποιείται σε μια τυχαία ούρα, από τις ουρές αποστολής.

4.2.6 Διεργασία ενημέρωσης κατάστασης παράδοσης email

Η διεργασία αυτή υλοποιείται από την μέθοδο *updateEmailDeliveryStatus* και είναι προγραμματισμένη να εκτελείται ανά 60 δευτερόλεπτα. Σκοπός αυτής της διεργασίας είναι να ενημερώσει τον πίνακα της βάσης δεδομένων *email_outgoing_event* για την κατάσταση παράδοσης του email. Η ενημέρωση αυτή δίνει την δυνατότητα στον χρήστη να βλέπει σε πραγματικό χρόνο τα email που έχουν παραδοθεί στους παραλήπτες, καθώς και πόσα email απέτυχαν να παραδοθούν.

Πριν εξηγήσουμε όμως τον αλγόριθμο αυτής της διεργασίας, πρέπει να εξηγήσουμε όλη την διαδικασία που ακολουθείται, ώστε να λάβει το backend τις πληροφορίες παράδοσης για κάθε email.

4.2.6.1 Εξαγωγή πληροφοριών παράδοσης από mail server

Το τελικό στάδιο αποστολής κάθε email είναι ο mail-server, ο οποίος ο οποίος είναι υπεύθυνος για την πραγματική αποστολή του email στον παραλήπτη μέσω του πρωτοκόλλου SMTP.

Κατά την αποστολή, ο mail server (π.χ. Postfix, Sendmail):

- ***Λαμβάνει το email*** *από το backend (Publisher Module).*
- ***Προσπαθεί να παραδώσει το email*** *στον απομακρυσμένο mail server του παραλήπτη.*
- ***Καταγράφει το αποτέλεσμα*** *(επιτυχία ή αποτυχία) στο τοπικό αρχείο καταγραφής.*

Για την παρακολούθηση της κατάστασης παράδοσης κάθε email, απαιτείται η συλλογή των σχετικών πληροφοριών από τον mail server. Η διαδικασία αυτή υλοποιείται μέσω ενός Bash script, το οποίο εκτελείται περιοδικά ανά τρία λεπτά (με την χρήση cron) απευθείας στον mail server.

Η βασική λειτουργία του script είναι να αναλύει το αρχείο καταγραφής (/var/log/mail.log) και να εξάγει για κάθε γραμμή τα εξής δεδομένα:

- ***message_id****: Το μοναδικό αναγνωριστικό του email.*
- ***mailbox****: Η διεύθυνση του παραλήπτη.*
- ***status****: Η κατάσταση παράδοσης (π.χ. sent, bounced, deferred).*

- ***server_response****: Η πλήρης απάντηση που επέστρεψε ο mail server.*

Τα παραπάνω δεδομένα εισάγονται απευθείας στον πίνακα *email_delivery_event* της βάσης δεδομένων του backend. Ο πίνακας αυτός έχει ένα ακόμη πεδίο που λειτουργεί ως σημαία, το πεδίο *is_processed*. Στην ουσία ο πίνακας αυτός λειτουργεί σαν κλώνος των logs του mail server, με την διαφορά ότι τα δεδομένα είναι οργανωμένα και μπορούμε να τα χρησιμοποιήσουμε.

4.2.6.2 Αλγόριθμος ενημέρωσης κατάστασης παράδοσης email

Έχοντας λοιπόν ως δεδομένο ότι στον πίνακα *email_delivery_event* υπάρχουν τα στοιχεία παράδοσης που έχουν εξαχθεί από τον mail server, τώρα το μόνο που μένει είναι η ενημέρωση του πίνακα email_outgoing_event.

Όλη η διαδικασία ενημέρωσης της κατάστασης παράδοσης βασίζεται στο μοναδικό UUID που δημιουργείται στο αντικείμενο PublishedMailDto, κατά την δημιουργία και αποστολή των email. Με βάση αυτό το αναγνωριστικό γίνεται η αντιστοίχιση μεταξύ των logs του mail server, και των εγγραφών στην βάση του backend.

Το πρώτο βήμα του αλγορίθμου είναι να πάρει όλες τις εγγραφές από τον πίνακα *email_delivery_event* που έχουν την σημαία *is_processed* ως *fals*e. Ουσιαστικά, επιλέγει όλες τις εγγραφές του πίνακα που δεν έχουν επεξεργαστεί.

Στην συνέχεια, για κάθε εγγραφή που επέλεξε, βρίσκει την αντίστοιχη σειρά στον πίνακα *email_outgoing_event*, με βάση το μοναδικό *message_id*. Μετά ενημερώνει την εγγραφή του *email_outgoing_event* με το κατάλληλη τιμή στα πεδία *is_sent, mail_server_response* και *sent_timestamp*.

Τέλος, ενημερώνει το πεδίο *is_processed* στον πίνακα *email_delivery_event*, για να σηματοδοτήσει ότι η συγκεκριμένη εγγραφή έχει επεξεργαστεί.

4.3 Publisher module

Το publisher module αναλαμβάνει να φέρει εις πέρας το τελευταίο στάδιο της διαδικασίας αποστολής email. Η αρμοδιότητα του είναι να λαμβάνει τα έτοιμα email από την ουρές του RabbitMQ, και να τα προωθεί στους κατάλληλους Mail-servers.

4.3.1 Δομή του Publisher module

Το module αποτελείται από δύο στρώματα. Το πρώτο είναι το Service layer, το οποίο υλοποιεί όλη την λογική που αφορά την επικοινωνία με τον message broker, και την λογική που ασχολείται με την κατανάλωση των μηνυμάτων που προέρχονται από τις ουρές, καθώς και την επικοινωνία με τους ίδιους τους mail servers. Το δεύτερο στρώμα είναι το στρώμα repository, που είναι αποκλειστικά υπεύθυνο για την επικοινωνία με την βάση δεδομένων, και παρέχει ένα abstraction, ώστε να μπορεί να χρησιμοποιεί το service layer την βάση δεδομένων.

Χρησιμοποιούνται τα packages *service* και *repository* τα οποία περιέχουν την λογική για κάθε στρώμα.

4.3.2 Αλληλεπίδραση με το RabbitMQ

Η επικοινωνία με τον message broker (RabbitMQ) υλοποιείται μέσω της βιβλιοθήκης *spring-boot-starter-amqp*, η οποία παρέχει ολοκληρωμένη υποστήριξη για το πρωτόκολλο AMQP (Advanced Message Queuing Protocol). Το publisher module χρησιμοποιεί τα components *RabbitTemplate, RabbitAdmin* και *SimpleMessageListenerContainer* για την αποστολή, δημιουργία και λήψη μηνυμάτων.

4.3.2.1 Δημιουργία ουρών

Το publisher module είναι υπεύθυνο για την δημιουργία των κατάλληλων ούρων στον message broker. Κάθε ουρά που δημιουργείται αντιστοιχεί σε έναν ενεργό mail-server, και το executor module χρησιμοποιεί αυτές τις ουρές για να στείλει τα μηνύματα στον κατάλληλο mail-server.

Πιο συγκεκριμένα, στη βάση δεδομένων υπάρχει ο πίνακας *mail_server*, ο οποίος περιέχει τις ρυθμίσεις όλων των ενεργών mail servers του συστήματος, όπως το *hostname*, το *port*, και τα διαπιστευτήρια σύνδεσης. Κατά την εκκίνηση του publisher module, η μέθοδος *setupQueuesAndListeners* εκτελείται και κάνει τα εξής βήματα για να αρχικοποιήσει τις ουρές:

- *Εκτελείται ένα ερώτημα στην βάση δεδομένων, και λαμβάνονται όλες οι εγγραφές του πίνακα mail_server. Κάθε εγγραφή αντιστοιχεί σε έναν ξεχωριστό mail server.*
- *Για κάθε mail server, δημιουργείται μία νέα ουρά στο RabbitMQ με όνομα ίδιο με το url του mail server (εφόσον δεν υπάρχει ήδη)*
- *Καταχωρείται σε έναν in-memory ConcurrentHashMap το όνομα της ουράς ώστε να είναι διαθέσιμη κατά την επεξεργασία των μηνυμάτων.*
- *Δημιουργείται και ενεργοποιείται ένας listener (SimpleMessageListenerContainer) για την παρακολούθηση της αντίστοιχης ουράς. Ο listener αυτός αναλαμβάνει να καταναλώνει τα μηνύματα που φτάνουν στην ουρά και να τα προωθεί για επεξεργασία.*

Αυτός ο τρόπος δυναμικής δημιουργίας και διαχείρισης των ουρών προσφέρει ευελιξία, καθώς δεν απαιτείται στατική διαμόρφωση των ουρών. Η πλατφόρμα μπορεί να προσαρμόζεται αυτόματα σε αλλαγές στον αριθμό ή τις ρυθμίσεις των mail servers. Επίσης, δίνει την δυνατότητα της γρήγορης προσθήκης η αφαίρεσης ενός mail server/ουράς, χωρίς να απαιτούνται χειροκίνητες ρυθμίσεις στον broker

4.3.2.2 Queue listeners

Για κάθε ουρά που δημιουργείται, ταυτόχρονα δημιουργείται και ένας listener για την συγκεκριμένη ουρά. Ένας listener, σε χαμηλό επίπεδο, λειτουργεί ως συνεχής καταναλωτής μηνυμάτων που παραμένει συνδεδεμένος με μια συγκεκριμένη ουρά στον message broker. Δημιουργεί μια συνεχή σύνδεση με τον broker και εγγράφεται στην ουρά, ώστε να λαμβάνει ειδοποιήσεις για την άφιξη νέων μηνυμάτων.

Με το που φτάνει ένα μήνυμα σε κάποια από τις ουρές, ο message broker παραδίδει αυτό το μήνυμα στον listener. Ο Listener με την σειρά του, επιβεβαιώνει το μήνυμα, και στην συνέχεια καλεί τον προγραμματισμένο χειριστή (callback handler), την μέθοδο *processMessage* δίνοντας ως παράμετρο το μήνυμα. Αυτές είναι οι δύο αρμοδιότητες του listener.

Η σύνδεση του listener είναι ασύγχρονη και event-driven, διατηρείται ενεργή όσο η εφαρμογή τρέχει και ο broker στέλνει νέα μηνύματα χωρίς την ανάγκη επανειλημμένης ερώτησης (polling). Αυτό βελτιώνει την απόδοση και μειώνει την καθυστέρηση στη λήψη μηνυμάτων.

4.3.3 Διαδικασία επεξεργασίας μηνυμάτων

Η διαδικασία αυτή είναι ο πυρήνας του publisher module, και περιέχει όλη την λογική που αφορά την επεξεργασία των μηνυμάτων που λαμβάνονται από την ουρά, και την αποστολή τους στον mail server.

Η ροή ξεκινά από τον queue listener, ο οποίος αφού λάβει ένα μήνυμα από την ουρά, εκκινεί την διαδικασία επεξεργασίας, καλώντας την μέθοδο processMessage. Η μέθοδος αυτή λαμβάνει ως παραμέτρους το όνομα της ουράς από την οποία προέρχεται το μήνυμα, και το περιεχόμενο του μηνύματος.

Ο αλγόριθμος στην συνέχεια κάνει τα εξής βήματα:

- *Αρχικά, επιβεβαιώνει ότι το όνομα της ουράς αφορά μια υπάρχουσα ουρά, χρησιμοποιώντας το ConcurrentHashMap το οποίο περιέχει τα ονόματα όλων των ουρών.*
- *Στην συνέχεια, ο αλγόριθμος, μετατρέπει το JSON string του μηνύματος σε πίνακα κλειδιών-τιμών (Map<String, String>), που περιέχει τα στοιχεία του email, όπως:*
- *Μοναδικό αναγνωριστικό μηνύματος (uniqueId)*
- *Διεύθυνση αποστολέα (sender)*
- *Διεύθυνση παραλήπτη (receiverEmail)*
- *Θέμα (title)*
- *Περιεχόμενο του μηνύματος κωδικοποιημένο σε Base64 (mailContentBase64)*
- *Έπειτα, χρησιμοποιώντας τα δεδομένα που εξάγονται από το μήνυμα, ο αλγόριθμος δημιουργεί ένα αντικείμενο τύπου MimeMessage μέσω του JavaMailSender. Το MimeMessage αναπαριστά το ηλεκτρονικό μήνυμα με όλες τις απαραίτητες κεφαλίδες και το περιεχόμενο. Συγκεκριμένα, ορίζονται πεδία όπως ο αποστολέας, ο παραλήπτης, το θέμα, καθώς και το σώμα του email, το οποίο μπορεί να περιέχει HTML.*
- *Μετά, προστίθενται στο μήνυμα κάποιες επιπλέον προαιρετικές κεφαλίδες :*
- ***X-Message-id**: Ένα μοναδικό αναγνωριστικό για το συγκεκριμένο μήνυμα. Χρησιμοποιείται για ιχνηλασιμότητα, ώστε να μπορεί να εντοπιστεί το μήνυμα σε logs ή συστήματα παρακολούθησης. Αυτό το αναγνωριστικό χρησιμοποιείται ώστε να αναγνωριστεί η κατάσταση αποστολής κάθε email στα logs του mail server*
- ***X-Mailer**: Δηλώνει το λογισμικό ή το σύστημα που χρησιμοποιήθηκε για την αποστολή του email.*
- ***X-Original-Sender**: Περιέχει τη διεύθυνση του αρχικού αποστολέα του μηνύματος.*
- ***X-Report-Abuse**: Παρέχει πληροφορίες ή οδηγίες για το πού μπορεί κάποιος να αναφέρει κακόβουλη χρήση ή ανεπιθύμητα μηνύματα (spam)*
- ***X-Feedback-ID**: Παρόμοιο με το X-Message-id, χρησιμοποιείται για αναφορά και παρακολούθηση του μηνύματος, συχνά σε συστήματα ανατροφοδότησης (feedback loops) ώστε να συνδέονται τα συμβάντα με συγκεκριμένα emails.*
- *Τέλος, το μήνυμα αποστέλλεται στον κατάλληλο mail server, με την χρήση του πρωτοκόλλου SMTP*

4.4 Αρχιτεκτονική συστήματος

Το σύστημα και η αρχιτεκτονική του είναι σχεδιασμένη με γνώμονα την απόδοση, την εύκολη κλιμακωσιμότητα, και φυσικά την ευκολία συντήρησης και επέκτασης του συστήματος. Έχει γίνει

ξεκάθαρος διαχωρισμός ευθυνών, και κάθε κομμάτι του συστήματος έχει τις δικές του συγκεκριμένες αρμοδιότητες.

Η βάση δεδομένων είναι κοινή για όλα τα κομμάτια του συστήματος, αλλά κάθε κομμάτι έχει περιορισμένη πρόσβαση και μπορεί να χρησιμοποιήσει τους πίνακες της βάση που αφορούν μόνο αυτό. Η κοινή βάση δεδομένων μας επιτρέπει να κρατήσουμε ένα χαμηλό επίπεδο πολυπλοκότητας, και να αποφύγουμε την επανάληψη πληροφοριών, και ταυτόχρονα, οι περιορισμοί που έχουμε επιβάλλει στην χρήση της βάσης από τα κομμάτια του συστήματος, κάνει το σύστημα πολύ πιο εύκολα συντηρήσιμο, καθώς αλλαγές σε ένα κομμάτι του συστήματος, ακόμα και αν αφορούν την βάση δεδομένων, δεν επηρεάζουν τα άλλα συστήματα σε μεγάλο βαθμό, ή και καθόλου.

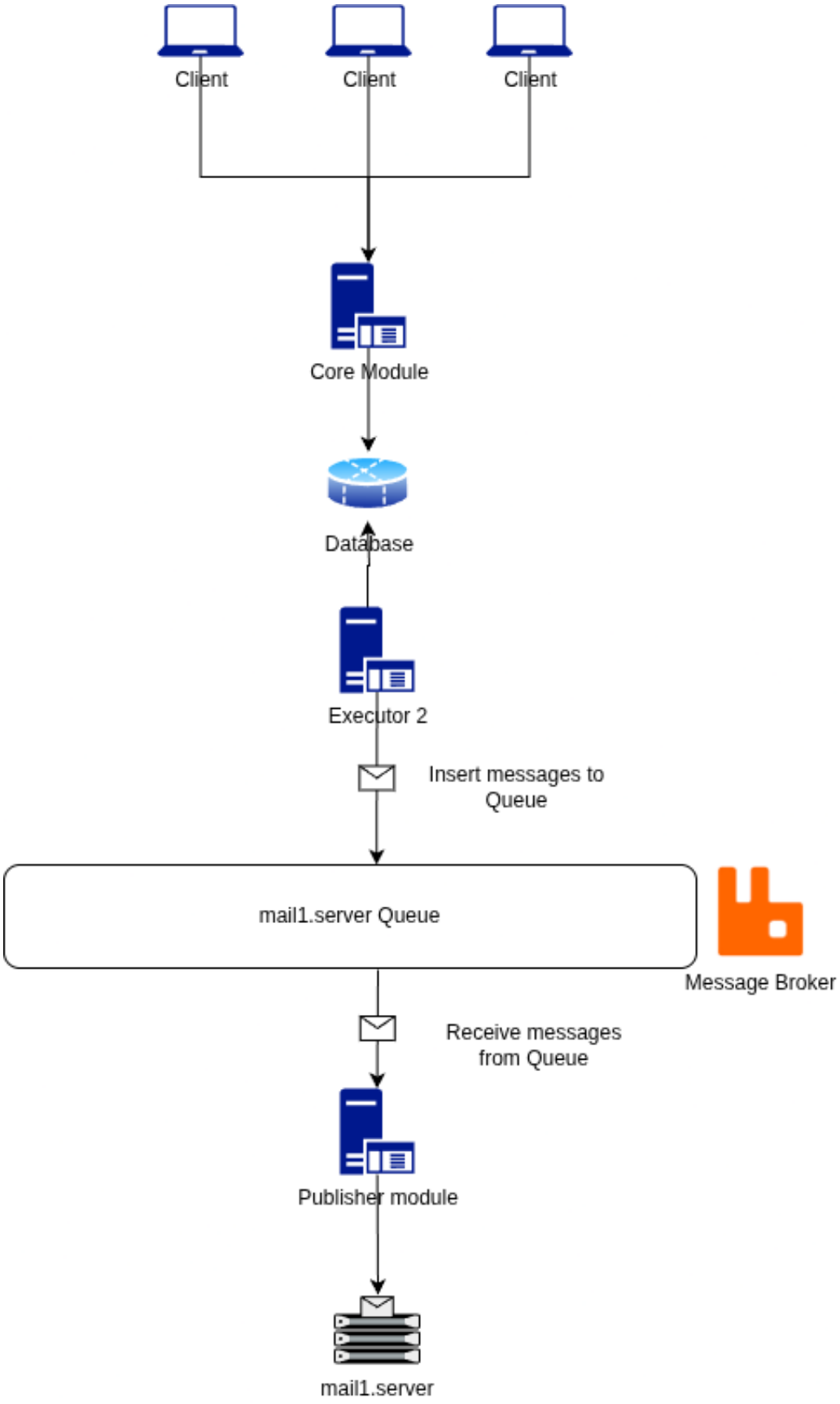
4.4.1 Αρχιτεκτονική υψηλού επιπέδου

Το σύστημα αποτελείται από τρία modules:

- ***Core Module****: Είναι το κομμάτι του συστήματος που υλοποιεί το API, το οποίο χρησιμοποιούν οι πελάτες, μέσω του frontend, για να αλληλοεπιδράσουν με το σύστημα. Αποθηκεύει στην βάση δεδομένων όλα όσα αφορούν τους χρήστες, όπως καμπάνιες που έχουν δημιουργήσει. Το core module και η επικοινωνία με τους πελάτες γίνεται με stateless τρόπο, πράγμα που επιτρέπει την εύκολη κλιμάκωση του.*
- ***Executor module****: Είναι το κομμάτι του συστήματος που έχει ως αρμοδιότητα την εκτέλεση των καμπανιών των χρηστών. Χρησιμοποιεί τους κατάλληλους πίνακες στην βάση δεδομένων ώστε να ανιχνεύσει τις προγραμματισμένες καμπάνιες και στην συνέχεια προχωρά στην εκτέλεση τους. Δεν έχει απευθείας επικοινωνία με το core module, επικοινωνεί μόνο έναν message broker, στο οποίο δημοσιοποιεί τα email που είναι έτοιμα προς αποστολή.*
- ***Publisher module****: Ευθύνη του publisher module είναι να λαμβάνει μηνύματα από τον message broker, να κάνει την κατάλληλη μετατροπή σε μορφή email και να τα προωθεί στους κατάλληλους mail server. Έχει περιορισμένη χρήση της βάσης δεδομένων, και δεν επικοινωνεί απευθείας με τα άλλα modules.*

Βασικό κομμάτι του συστήματος είναι ο **message broker**. Ο message broker είναι ο διαμεσολαβητής, ο οποίος αναλαμβάνει την επικοινωνία μεταξύ των executor και publisher modules. Πιο συγκεκριμένα, ο broker υλοποιεί μία ή και παραπάνω ουρές. Σε αυτες τις ουρές, τον ρόλο του εκδότη (Publisher) τον έχει το executor module. Δημοσιοποιεί μηνύματα τα οποία αφορούν έτοιμα email που είναι για αποστολή. Το publisher module είναι ο καταναλωτής αυτών τον μηνυμάτων. Θα μπορούσαμε να πούμε λοιπόν ότι η επικοινωνία μεταξύ των modules είναι μονόδρομη, καθώς ο executor στέλνει μηνύματα, και ο publisher τα λαμβάνει.

Τέλος, από το σύστημα δεν θα μπορούσε να λείπει ένας ή και παραπάνω mail server. Ο mail server είναι το τελευταίο στάδιο στην αρχιτεκτονική, και μοναδική του αρμοδιότητα είναι να λαμβάνει email από το publisher module και να τα στέλνει στον τελικό παραλήπτη.



Εικόνα 3: Πλήρης αρχιτεκτονική συστήματος

‘Ενας άλλος τρόπος να περιγράψουμε την αρχιτεκτονική του συστήματος είναι να το χωρίσουμε σε δύο μεγάλα υποσυστήματα.

- *Το API του συστήματος που αποτελείται αποκλειστικά από το core module.*
- *Το σύστημα εκτέλεσης των καμπανιών χρήστη, που αποτελείται από τα executor και publisher modules, από τον message broker και από τον mail-server.*

‘Όπως συμπεραίνουμε από την παραπάνω εικόνα, ο συνδετικός κρίκος των δύο υποσυστημάτων είναι η βάση δεδομένων.

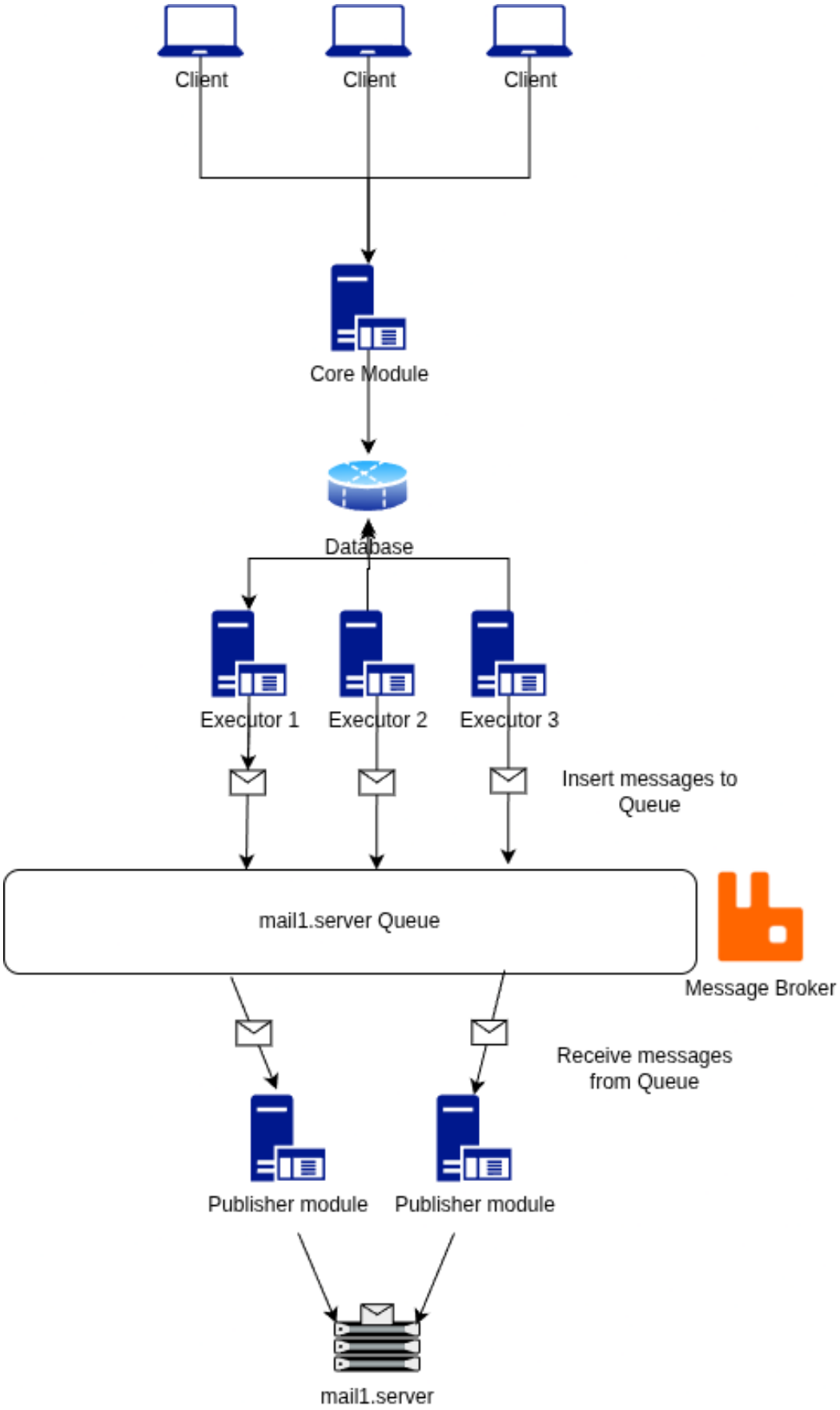
4.4.2 Κλιμακωσιμότητα του συστήματος

Η αρχιτεκτονική των microservices, μας δίνει την δυνατότητα να κλιμακώνουμε κάθε κομμάτι του συστήματος με διαφορετικούς ρυθμούς. Παρακάτω θα εξετάσουμε μερικά παραδείγματα κλιμάκωσης, όπου το κάθε παράδειγμα εξηγεί έναν διαφορετικό τρόπο κλιμάκωσης, ανάλογα με τις απαιτήσεις και τον φόρτο κάθε υποσυστήματος.

4.4.2.1 Κλιμάκωση ρυθμού αποστολής email

Ας υποθέσουμε, ότι το σύστημα μας έχει λίγους χρήστες, οι οποίοι όμως δημιουργούν καθημερινά καμπάνιες με εκατομμύρια email. Σε αυτή την περίπτωση, το API της εφαρμογής μας δεν θα δέχεται πολύ φόρτο, καθώς μιλάμε μόνο για 5 χρήστες. Αντιθέτως, το υποσύστημα που είναι υπεύθυνο για την δημιουργία και αποστολή email δεν θα μπορεί να ανταπεξέλθει αν δεν υπάρχει η κατάλληλη κλιμάκωση.

Η σωστή προσέγγιση λοιπόν είναι να κλιμακώσουμε το executor module ώστε να υπάρχουν τρία διαφορετικά instances, τα οποία θα εκτελούν παράλληλα καμπάνιες, και ταυτόχρονα να κλιμακώσουμε το publisher module σε δύο διαφορετικά instances, τα οποία επίσης θα τρέχουν παράλληλα. Ο λόγος που δεν κλιμακώνουμε με τον ίδιο ρυθμό τα modules, είναι γιατί το executor module δέχεται πολύ περισσότερο φόρτο για κάθε email που αποστέλλεται, σε σχέση με το publisher module. Επίσης ο mail server μπορεί να ανταπεξέλθει, για αυτό δεν υπάρχει κλιμάκωση.

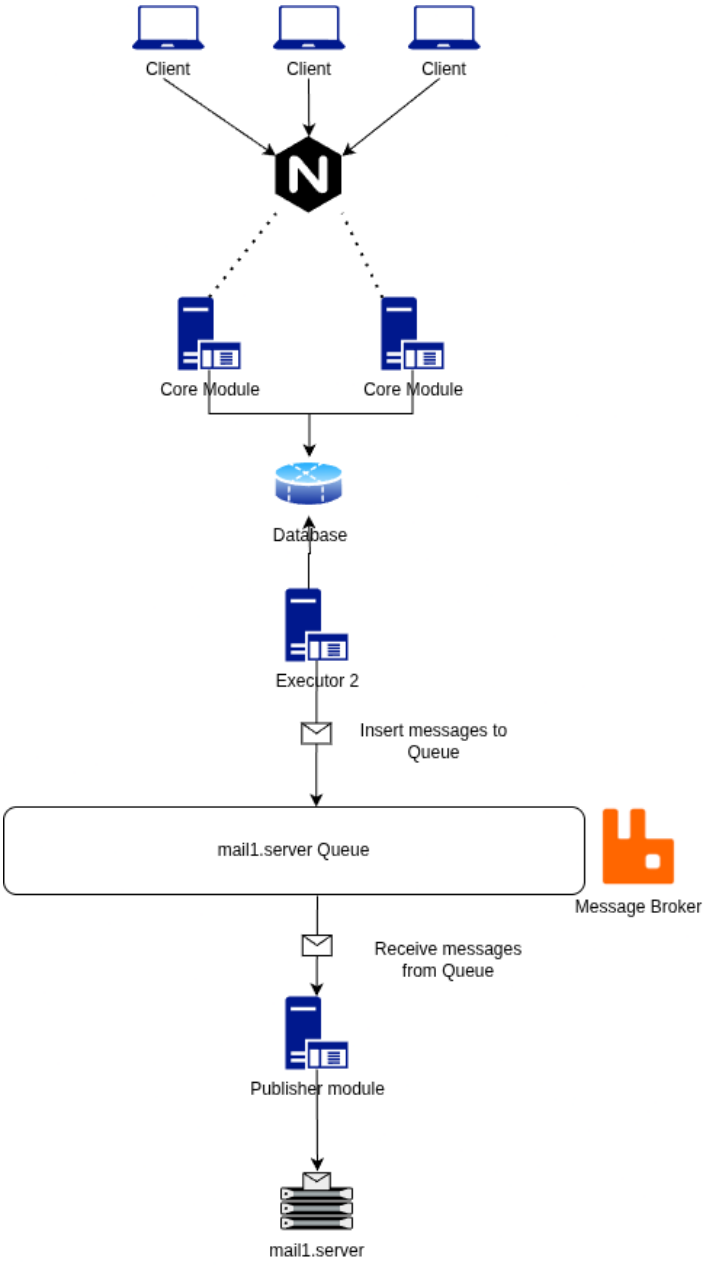


Εικόνα 4: Παράδειγμα κλιμάκωσης 1

4.4.2.2 Κλιμάκωση API

Ας υποθέσουμε για αυτό το παράδειγμα, ότι το σύστημα μας έχει αρκετούς χρήστες, ο οποίοι χρησιμοποιούν το σύστημα μας για να δημιουργούν και να αποστέλλουν καμπάνιες των λίγων email. Σε αυτή την περίπτωση, το API της εφαρμογής θα δέχεται αρκετό φόρτο, το υποσύστημα όμως που ασχολείται με την δημιουργία και αποστολή email θα μπορεί να ανταπεξέλθει, διότι το συνολικό πλήθος των email δεν θα είναι μεγάλο.

Σε αυτή την περίπτωση θα πρέπει να κλιμακώσουμε το core module, ώστε να βεβαιωθούμε ότι μπορεί να ανταπεξέλθει στα πολλαπλά αιτήματα χρηστών, χωρίς να επηρεαστεί η εμπειρία τους, λόγω καθυστερήσεων. Η κλιμάκωση του core module μπορεί να γίνει είτε με την χρήση ενός load-balancer, είτε με την χρήση round-robin DNS. Εμείς επιλέγουμε να το κάνουμε την χρήση ενός load-balancer.

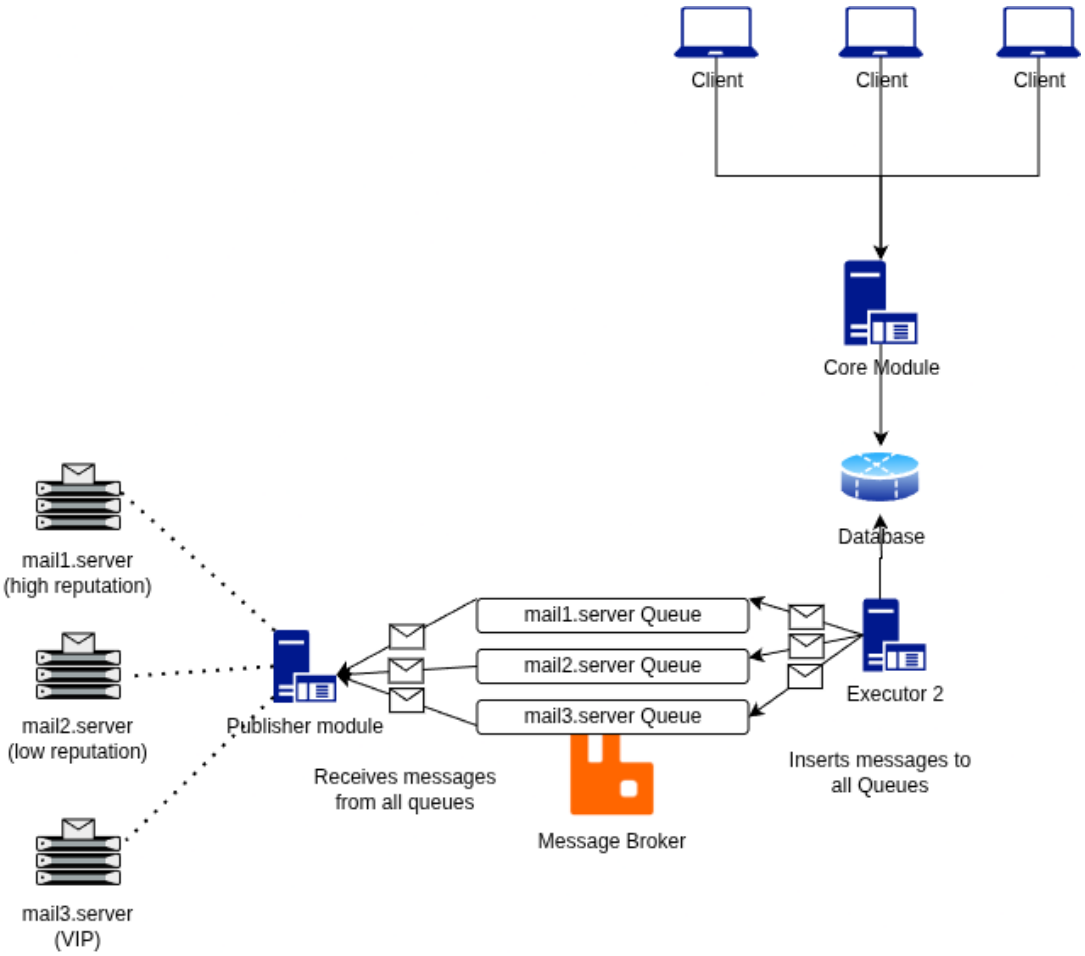


Εικόνα 5: Παράδειγμα κλιμάκωσης 2

4.4.3 Email servers

Η αρχιτεκτονική του συστήματος δίνει την δυνατότητα να χρησιμοποιηθούν πολλαπλοί mail servers, όπου όλοι θα στέλνουν ένα μέρος των email καμπανιών. Υπάρχουν δύο λόγοι για τους οποίους θα θέλαμε να χρησιμοποιήσουμε παραπάνω από έναν mail server:

- *Αρχικά, με το να υπάρχουν πολλοί mail server, αυξάνεται ο ρυθμός αποστολής email του συστήματος. Αν διαπιστώσουμε λοιπόν ότι ο mail server γίνεται ο περιοριστικός παράγοντας στον ρυθμό αποστολής email, τότε πολύ εύκολα μπορούμε να προσθέσουμε και άλλους.*
- *Ο δεύτερος λόγος έχει να κάνει με τον μηχανισμό reputation που χρησιμοποιούν οι μεγάλες πλατφόρμες email. Κάθε mail server ,ή πιο συγκεκριμένα κάθε IP αποστολής , αποκτά σταδιακά ένα σκορ αξιοπιστίας με βάση τη συμπεριφορά του. Με τη χρήση πολλαπλών mail servers, μπορούμε να διαχειριστούμε καλύτερα το reputation ανά server και να απομονώσουμε προβληματικές IP, χωρίς να επηρεαστεί η συνολική ικανότητα αποστολής του συστήματος. Για παράδειγμα, καμπάνιες οι οποίες έχουν χαρακτηριστεί ως "υψηλού spam score" μπορούν να αποστέλλονται με την χρήση ενός mail server με χαμηλό reputation, ενώ ταυτόχρονα, καμπάνιες από έμπιστους χρήστες μπορούν να αποστέλλονται από mail server με υψηλή αξιοπιστία.*



Εικόνα 6: Παράδειγμα πολλαπλών mail server

4.4.4 Βάση δεδομένων

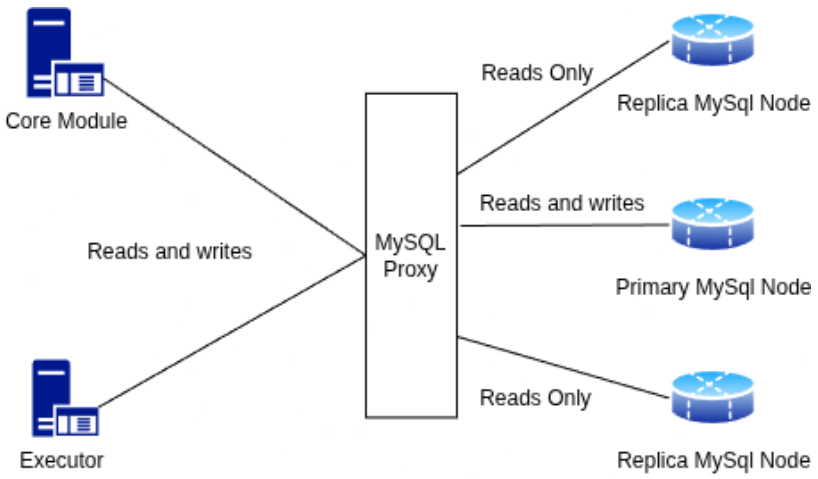
Η βάση δεδομένων είναι ο πυρήνας του συστήματος, και είναι ο συνδετικός κρίκος μεταξύ των δύο υποσυστημάτων, του API και του υποσυστήματος εκτέλεσης καμπανιών. Επιλέξαμε να χρησιμοποιήσουμε ως βάση δεδομένων την MySQL, γιατί μεταξύ άλλων προσφέρει λύσεις που μας επιτρέπουν να την κλιμακώσουμε (μερικώς) ώστε να υποστηρίξει μεγαλύτερο φόρτο χρηστών.

Το επίπεδο απομόνωσης που επιλέξαμε για τη βάση δεδομένων είναι το *Read Committed*, το οποίο εξασφαλίζει ότι κάθε SELECT επιστρέφει μόνο δεδομένα που έχουν ήδη γίνει commit από άλλες συναλλαγές. Με αυτόν τον τρόπο αποφεύγονται dirty reads, ενώ παράλληλα διατηρείται υψηλή απόδοση και μειώνεται ο κίνδυνος εμφάνισης deadlocks σε περιβάλλον με υψηλή ταυτόχρονη πρόσβαση, όπως όταν υπάρχουν πολλαπλά executor modules για παράδειγμα.

Η επιλογή του συγκεκριμένου isolation level έγινε γιατί προσφέρει μια ισορροπία μεταξύ συνέπειας και επιδόσεων, γεγονός που είναι κρίσιμο για την περίπτωση μας, όπου τα transactional guarantees είναι σημαντικά (π.χ. δεν πρέπει να σταλεί διπλό email, ούτε να αρχικοποιηθεί μια καμπάνια δύο φορές), αλλά απαιτείται και υψηλός ρυθμός εισαγωγής/ανάκτησης δεδομένων.

Όσον αφορά την κλιμακωσιμότητα, οι επιλογές που υπάρχουν για την MySQL είναι τρεις:

- ***Κάθετη κλιμάκωση****: Περιλαμβάνει την αναβάθμιση του υλικού στον ίδιο server, όπως αύξηση της RAM, ταχύτερος δίσκος, περισσότερα CPU cores. Είναι η πιο απλή και άμεση λύση, αλλά σε αυτή την περίπτωση, η βάση δεδομένων παραμένει ως το μοναδικό σημείο αποτυχίας.*
- ***Οριζόντια κλιμάκωση ανάγνωσης****: Με την χρήση του μηχανισμού master–replica που προσφέρει η MySQL, μπορούμε να εξυπηρετούμε αναγνώσεις από δευτερεύοντες κόμβους (replicas), διατηρώντας τις εγγραφές μόνο στον primary. Αυτό θα μειώσει τον φόρτο στον primary κόμβο και θα ανεβάσει την δυνατότητα της βάσης να διαχειριστεί παραπάνω read/writes. Επίσης αυτή η λύση δεν επηρεάζει τον κώδικα του συστήματος, γιατί μπορεί να χρησιμοποιηθεί για παράδειγμα ProxySQL το οποίο αναλαμβάνει αποκλειστικά την κατανομή των read/writes.*
- ***Οριζόντια κλιμάκωση εγγραφών****: Τα δεδομένα κατανέμονται σε πολλαπλές βάσεις , με την κάθε μία να φιλοξενεί μέρος του συνόλου των δεδομένων. Το partitioning μπορεί να γίνει βάσει client ID, campaign ID ή άλλου σταθερού κριτηρίου. Η λογική του sharding ενσωματώνεται είτε στον κώδικα της εφαρμογής είτε μέσω middleware όπως το Vitess, που προσφέρει query routing και consistency πάνω από πολλαπλά MySQL instances.*



Εικόνα 7: Παράδειγμα οριζόντιας κλιμάκωση ανάγνωσης

Κεφάλαιο 5ο: Υλοποίηση front-end

Στο κεφάλαιο αυτό θα αναλυθεί η δομή και κάποια βασικά τμήματα της frontend εφαρμογής.

Για την δημιουργία του frontend χρησιμοποιήθηκε η Vue.js έκδοση 3.4.18 με την χρήση του composition api. Χρησιμοποιήθηκε επίσης η γλώσσα Typescript και το framework Quasar. Επίσης, χρησιμοποιήθηκαν οι εξής βιβλιοθήκες:

- *Vue router*
- ***Pinia** για το state management*
- ***Axios** για την επικοινωνία μέσω http για της κλήσεις στο API του backend*
- *Το **Vue I18n** για τα λεκτικά και μεταφράσεις τις εφαρμογής*
- ***Vue Email Editor[37]**, έναν open-source email template επεξεργαστή.*

5.1 Οργάνωση κώδικα

Η δομή των φακέλων που ακολουθήθηκε για την οργάνωση του κώδικα ήταν η εξής:

```
src/
├── assets/          # Static assets (images, fonts)
├── boot/            # Boot files (executed before app initialization)
├── components/      # Reusable Vue components
├── css/             # Global CSS/SCSS styles
├── i18n/            # Internationalization resources
├── layouts/         # Layout components
├── pages/           # Page components
├── router/          # Vue Router configuration
├── stores/          # Pinia stores for state management
└── App.vue          # Root Vue component
```

Εικόνα 8: Δομή φακέλων front-end

Αρχικά, στον φάκελο *assets* έχουν τοποθετηθεί όλα τα αρχεία όπως οι εικόνες που χρειάζονται για την εφαρμογή όπως το λογότυπο της. Στην συνέχεια έχουμε τον φάκελο *boot* που περιλαμβάνει εντός του αρχεία που έχουν να κάνουν κυρίως με την παραμετροποίηση βιβλιοθηκών όπως για παράδειγμα το axios και το i18n. Στον φάκελο **component** τοποθετούνται όλα τα επαναχρησιμοποιούμενα στοιχεία της εφαρμογής όπως η επάνω μπάρα, το footer καθώς και το burger menu. Για την αποθήκευση όλων των αρχείων που έχουν να κάνουν με το στιλιστικό κομμάτι τις εφαρμογής έχουν τοποθετηθεί εντός του φακέλου *css* καθώς και στον φάκελο *i18n* τα αρχεία που έχουν να κάνουν με τα λεκτικά και τις μεταφράσεις τις εφαρμογής.

Ο υπόλοιπος κώδικας οργανώθηκε χρησιμοποιώντας τους φακέλους:

- ***Layouts** που εντός τους υπάρχουν τα 2 βασικά layout της εφαρμογής, το βασικό που έχει πρόσβαση ο χρήστης μόνο αφού έχει κάνει την διαδικασία του login καθώς και το authentication layout το οποίο «κλειδώνει» έναν χρήστη που δεν έχει κάνει σύνδεση στον λογαριασμό του στις σελίδες όπως σύνδεσης, εγγραφής και επαναφοράς κωδικού. Με αυτό τον τρόπο γίνεται πολύ εύκολη η διαχώριση αρχικών ενεργειών και πρόσβασης στις ενεργείς που μπορεί να κάνει ο χρήστης.*
- ***Pages**, εδώ συναντιούνται οι σελίδες που τρέχουν εντός των layouts. Χρησιμοποιώντας τις σελίδες μπορεί να γίνει διαχώριση της λογικής μεταξύ των περεταίρω ενεργειών που επιτρέπονται από κάθε layout. Για παράδειγμα εντός του MainLayout υπάρχουν οι σελίδες AnalyticsPage που*

παρουσιάζουν τα στατιστικά για τις καμπάνιες του εκάστοτε συνδεδεμένου χρήστη, ενώ υπάρχουν και οι σελίδες όπως LoginPage που χρησιμοποιούνται από το AuthLayout όταν ένας χρήστης δεν είναι συνδεδεμένος.

- *Στον φάκελο **router** υπάρχουν εντός του οι routers που αντιστοιχούν σελίδες σε layouts καθώς και επιτρέπουν ή απαγορεύουν την εμφάνιση τους ανάλογα με το εάν ο χρήστης είναι συνδεδεμένος ή όχι.*
- *Για την παραμετροποίηση του station manager υπάρχει ο φάκελος του **stores** που εκεί βρίσκουμε τα αρχεία παραμετροποίησης της βιβλιοθήκης pinia.*
- *Και τέλος, το ποιο σημαντικό κομμάτι τις εφαρμογής αποτελεί το αρχείο **App.vue** το οποίο είναι υπεύθυνο για να ξεκινήσει η εφαρμογή καθώς περιέχει εντός του όλα τα συνδετικά τμήματα ώστε τα παραπάνω κομμάτια να μπορούν να συνδεθούν και να δουλέψουν μαζί.*

Στα επόμενα κεφάλαια θα αναλυθούν ξεχωριστά τα παραπάνω κομμάτια τις εφαρμογής.

5.2 Layout System

Στο κομμάτι των layout εμφανίζονται δυο βασικά layout που γύρω τους έχει δομηθεί η εφαρμογή. Αυτά είναι το Authentication Layout καθώς και το Main Layout.

5.2.1 Authentication Layout

Είναι υπεύθυνο για τις διαδικασίες που έχουν να κάνουν με σύνδεση και εγγραφή του χρήστη. Με την χρήση αυτό του layout «κλειδώνεται» ο χρήστης μόνο στις παραπάνω σελίδες και δεν του επιτρέπει να μπορεί να έχει πρόσβαση στις επόμενες λειτουργίες τις εφαρμογής εάν δεν κάνει σύνδεση σε περιπτώσεις που έχει λογαριασμό ή εγγραφή. Δεν περιέχει κανένα navigation element, υποστηρίζει το dark mode ή light mode καθώς και επικεντρώνεται, κάνοντας όσο πιο απλό και κατανοητό γίνεται, το flow τις συνδέσεις και αποσύνδεσης.

5.2.2 Main Layout

Το main layout Main Layout.vue χρησιμεύει ως η κύρια διάταξη για τους αυθεντικοποιημένος χρήστες. Εντός του περιέχει όλες τις σελίδες που έχουν να κάνουν με την βασική λειτουργικότητα της εφαρμογής. Παρέχει έναν responsive header με το λογότυπο της εφαρμογής, ένα αριστερό κεντρικό μενού που παρέχει links για όλα τα τμήματα της εφαρμογής όπως: Email Editor, Statistics κλπ. Παρέχει επίσης την δυνατότητα για dark mode ή light mode αλλάζοντας την βασική εμφάνιση της εφαρμογής ανάλογα με τις προτιμήσεις του χρήστη. Τέλος παρέχει μια καθολική δυνατότητα αποσύνδεσης η οποία είναι πρόσβαση από οποιαδήποτε σελίδα υπάρχει εντός του main layout.

5.2.3 Page Components

Κάθε σελίδα είναι υπεύθυνή για ένα συγκεκριμένο λειτουργικό κομμάτι της εφαρμογής. Ιεραρχικά οργανώνονται κάτω από κάποιο layout που είναι υπεύθυνο να παρέχει σε κάθε σελίδα κάποιες πληροφορίες που παραμένουν ιδίες σε όλη την εφαρμογή (όπως για παράδειγμα τα στοιχεία του χρήστη). Παρακάτω αναλύονται η κάθε σελίδα ξεχωριστά.

5.2.4 Index Page

Στην Index σελίδα υπάρχει η βασική σελίδα της εφαρμογής που εμφανίζει την λίστα με τις καμπάνιες που έχουν δημιουργηθεί. Αποτελεί την πρώτη σελίδα που θα δει ο χρήστης όταν κάνει σύνδεση μέσα στην εφαρμογή και το κινητήριο τμήμα της.

5.2.5 Editor Page

Στην σελίδα Editor περιλαμβάνεται όλη η διαδικασία για την δημιουργία μια καμπάνιας. Σε αυτό το κομμάτι συναντάμε σε βήματα την διαδικασία από την δημιουργία της καμπάνιας με πεδία όπως τίτλο και ποιος την στέλνει, την δημιουργία του html τμήματος του email, την βελτιστοποίηση του περιεχομένου, την επιλογή των ομάδων που θα λάβουν την καμπανιά και τέλος το να τρέξει η καμπανιά άμεσα ή σε κάποια συγκεκριμένη στιγμή στο μέλλον. Αποτελεί το βασικό τμήμα της εφαρμογής με αρκετά μεγάλο λειτουργικό βάρος.

5.2.6 Contacts Page

Παρέχει την δυνατότητα στον χρήστη να μπορεί να επεξεργαστεί και να δει τις επαφές που έχει ανεβασμένες στην εφαρμογή. Ειδικότητά μπορεί να ανεβάσει τις επαφές του χρησιμοποιώντας ένα csv αρχείο προσαρμόζοντας τους τίτλους στις στήλες σε αυτό. Παράλληλα δίνει την δυνατότητα να δημιουργήσει ομάδες επαφών με συγκεκριμένο όνομα ώστε να μπορεί να τις επιλέξει αργότερα στην αποστολή της καμπάνιας email. Τέλος μπορεί να προβάλει, να ψάξει και να επεξεργαστεί οποιοδήποτε στοιχείο κάποιας εφαρμογής και αυτό να αποθηκευτεί πίσω στο backend.

5.2.7 Campaign Page

Η σελίδα campaign page αποτελεί την διεπαφής της εφαρμογής λειτουργώντας ως ένα κέντρο παρακολούθησης για τις εκστρατείες email. Αποτελεί ένα πίνακα ελέγχου που παρουσιάζει μετρήσεις απόδοσης εκστρατειών σε πραγματικό χρόνο. Οι χρήστες μπορούν να εξετάσουν στατιστικά στοιχεία συμπεριλαμβανομένων των ποσοστών παράδοσης, των ποσοστό ανοίγματος και των μοτίβων αλληλεπίδρασης των παραληπτών. Η σελίδα επίσης δίνει την δυνατότητα στον χρήστη να διακόψει προσωρινά την εκτέλεση της καμπάνιας, να την συνεχίσει ξανά καθώς και να την σταματήσει εντελώς.

5.2.8 Analytics Page

Για την προβολή στατιστικών αποτελεσμάτων για τις καμπάνιες υπάρχει η σελίδα analytics. Στην συγκεκριμένη σελίδα υπάρχει οργανωμένο όλο το κομμάτι της εφαρμογής που παρουσιάζει στατιστικά που αφορούν κυρίως την απόδοση της κάθε καμπάνιας, καθώς και στο σύνολο όλων τους. Κάποια από τα στατιστικά που παρέχονται αποτελούν τα εξής:

- *Συνολικός αριθμός εξαετιών*
- *Συνολικό ποσοστό ανοίγματος (open rate) όλων των καμπανιών*
- *Όγκος συνολικών μηνυμάτων που έχουν αποσταλεί.*

Η παρουσίαση τους γίνεται με την χρήση δια δραστικών γραφημάτων που απεικονίζουν την κατανομή καταστάσεων των εκστρατειών και τα ποσοστά παράδοσης, ανοίγματος και αναπήδησης των email. Παράλληλα παρέχεται ένας λεπτομερής πίνακας απόδοσης επιτρέποντας την ταξινόμηση και αναζήτηση, παρέχοντας στους χρηστές τη δυνατότητα να αλγούν σε βάθος τα δεδομένα και να λαμβάνουν αποφάσεις για την βελτιστοποίηση μελλοντικών εκστρατειών.

5.2.9 Profile Page

Η σελίδα ProfilePage αποτελεί ένα συστατικό της διεπαφής χρήστη στην εφαρμογή Mail Sender, προσφέροντας ένα ολοκληρωμένο περιβάλλον διαχείρισης προσωπικών στοιχείων και ρυθμίσεων

λογαριασμού. Η σελίδα είναι δομημένη με μια διακριτή ιεραρχία πληροφοριών, παρουσιάζοντας αρχικά τα βασικά στοιχεία του χρήστη όπως διεύθυνση ηλεκτρονικού ταχυδρομείου και όνομα χρήστη και στη συνέχεια προσφέροντας λειτουργικότητα αλλαγής κωδικού πρόσβασης με ενσωματωμένους μηχανισμούς επαλήθευσης. Με αυτόν τον τρόπο μπορεί ο χρήστης να αποκτήσει πρόσβαση και να διαχειριστεί τα στοιχεία του με ευκολία.

5.2.10 Login Page

Η σελίδα login αποτελεί την πρώτη σελίδα που βλέπει ο χρήστης όταν εισέρχεται στην εφαρμογή. Χρησιμοποιεί δυο πεδία email και password ώστε να δώσει την δυνατότητα στον χρήστη να παρέχει τα διαπιστευτήρια του για να μπορεί να έχει πρόσβαση στην εφαρμογή. Το πεδίο email υποστηρίζει έλεγχο για το εάν η τιμή που έχει εισάγει ο χρήστης είναι επιτρεπτή, δηλαδή εάν είναι της μορφής *mailaddress@domain.code*. Η σελίδα είναι υπεύθυνη για την παραλαβή των στοιχείων του χρήστη και να κάνει την κλήση στο authentication api του backend ώστε να μπορεί να γίνει έλεγχος ύπαρξης και ορθότητας των στοιχείων του χρήστη.

5.2.11 Register Page

Σε περίπτωση που ο χρήστης δεν έχει λογαριασμό υπεύθυνη για την δημιουργία του είναι η σελίδα Register. Στην σελίδα αυτή ο χρήστης έχει την δυνατότητα να εγγραφή στην εφαρμογή χρησιμοποιώντας την διεύθυνση email του. Παρέχει μια φόρμα με πεδία:

- ***Username****: Το όνομα του χρήστη, μοναδικό χαρακτηριστικό.*
- ***Email****: Η διεύθυνση email του χρήστη, μοναδικό χαρακτηριστικό, χρησιμοποιείται για την επαλήθευση της διεύθυνσης email του χρήστη καθώς και ως default sender address για τις καμπάνιες email που θα δημιουργήσει.*
- ***Password****: Ο κωδικός που θα χρησιμοποιεί ο χρήστης για να εισέλθει στον λογαριασμό του. Το συγκεκριμένο πεδίο αποκρύπτεται κατά την πληκτρολόγηση του για την ασφάλεια του.*
- ***Confirmation Password****: Ο χρήστης πρέπει να πληκτρολόγηση ξανά τον κωδικό που έβαλε στο πρώτο πεδίο password. Το πεδίο αυτό αποκρύπτεται και πρέπει να ταιριάζει με το πεδίο password. Σε περίπτωση που οι κωδικοί δεν είναι οι ίδιοι τότε εμφανίζει μήνυμα λάθους και δεν επιτρέπει την εγγραφή του χρήστη στην εφαρμογής.*

Όταν συμπληρωθεί ολόκληρη η φόρμα επιτρέπεται στον χρήστη να πατήσει το κουμπί Register. Με αυτό τον τρόπο αποστέλλεται μέσω HTTP call το αίτημα από την σελίδα στο backend REST API για την εγγραφή του χρήστη μέσω του Axios.

5.3 Styling

Για την στιλιστική επιμέλεια της εφαρμογής χρησιμοποιήθηκε η επέκταση του κλασικού CSS SCSS ή αλλιώς Syntactically Awesome Style Sheets. Η SCSS επιτρέπει για την πιο εύκολη δημιουργία κανόνων για το πως θα εμφανίζεται η εφαρμογή στιλιστικά. Επιτρέπει την χρήση μεταβλητών, κανόνων, μειγμάτων καθώς και συναρτήσεων τα οποία μεταφράζονται σε CSS που θα μπορεί αργότερα να καταλάβει ο browser. Παράλληλα βοηθάει στο να παραμείνουν οργανωμένα χωρίς το χάος που κυριαρχεί συνήθως στα απλά CSS. Οργώνονται σε δυο αρχεία, το ένα αποτελεί το *quasar.variables.sass* που εντός του συναντάτε οι βασικές χρωματικές επιλογές επιμέρους κομματιών της εφαρμογής. Για παράδειγμα κρατάει το βασικό χρώμα της εφαρμογής, το δευτερεύων χρώμα της, τα χρώματα σε θετικές ενημερώσεις, τα χρώματα σε αρνητικές ενημερώσεις, τα χρώματα σε πληροφορίες καθώς και σε προειδοποίησης. Με αυτό τον τρόπο μπορεί να επαναχρησιμοποιηθούν ξανά και σε άλλες εφαρμογές που χρειάζεται να υπάρχει κοινό στιλιστικό κομμάτι. Τέλος τα επιμέρους CSS που έχουν να κάνουν μόνο με την συγκεκριμένη εφαρμογή βρίσκονται στο αρχείο *app.sass* οπού εντός του μπορεί κάποιος

να βρει των κώδικα που αφορά το πως η εφαρμογή αλληλοεπιδρά και εμφανίζεται στον χρήστη. Από τα βασικότερα κομμάτια τις αποτελούν οι μετατροπές στο weskit, το dark mode καθώς και το light mode.

Τέλος όταν κάποια sass αφορούν συγκεκριμένα τμήματα σελίδων και δεν μπορούν να γενικευτούν σε άλλες σελίδες τότε αποτελούν κομμάτι είτε του component είτε της σελίδας. Εντός αυτών των αρχείων υπάρχουν κομμάτια επισημασμένα με το *<style>* τα οποία περιέχουν την ειδική μορφοποίηση που εφαρμόζεται μόνο στο ανάλογο component ή σελίδα. Με αυτό τον τρόπο έχουμε πλήρης έλεγχο του πως εμφανίζεται αλλά και αλληλοεπιδρά η εφαρμογή με τον χρήστη με έναν πιο εύκολο και οργανωμένο τρόπο.

5.3.1 Dark mode

Η σκοτεινή λειτουργία υλοποιείται χρησιμοποιώντας την ενσωματωμένη υποστήριξη σκοτεινής λειτουργίας του Quasar. Με αυτό τον τρόπο μπορούμε ευκολά και γρηγορά να υποστηρίξουμε την σκοτεινή λειτουργία καθώς και να παρέχουμε την δυνατότητα αλλαγής σε φωτεινή λειτουργία ανάλογα με τις ανάγκες του χρήστη.

5.4 Api Integration

Για την επικοινωνία της εφαρμογής frontend με το backend έχει χρησιμοποιηθεί η βιβλιοθήκη Axios[38].

5.4.1 Παραμετροποίηση

Η παραμετροποίηση του Axios γίνεται στο αρχείο *axios.ts*. Εντός αυτού δημιουργήσετε η βασική οντότητα του οπού περιχέει το βασικό τμήμα του URL το οποίο αποτελεί το URL του backend REST API. Έχοντας ανοιχτή την παραμετροποίηση *withCredentials: true* επιτρέπει μαζί με κάθε request που γίνεται να προσθέτει και το authentication header για την αυθεντικοποίηση του χρήστη με το backend REST API. Με αυτό τον τρόπο η εφαρμογή μπορεί να έχει πρόσβαση στο instance του Axios globally ώστε ένα μπορούν όλα τα components να πραγματοποιήσουν κλήσεις προς το backend.

5.4.2 Αυθεντικοποίηση

Η εφαρμογή χρησιμοποιώντας το Axios αρχικά στέλνει ένα POST αίτημα στο backend. Εάν αυτό είναι επιτυχές, το JWT token αποθηκεύεται σε ένα HTTP-only cookie για ασφάλεια. Λόγω της παραμετροποίησης που αναφέρθηκε πάνω, αυτό συμπεριλαμβάνεται αυτόματα στα περισσότερα HTTP requests. Σε περίπτωση που υπάρχει κάποιο πρόβλημα κατά την αυθεντικοποίηση του χρήστη με το backend (λάβει τον κωδικό HTTP 401) ο interceptor του Axios αναλαμβάνει την ανακατεύθυνση του χρήστη στην σελίδα login ώστε να επανασυνδεθεί. Ταυτόχρονα γίνεται έλεγχος του expiration date claim του JWT token ώστε εάν έχει αυτό λήξη και υπάρχει refresh token, τότε φροντίζει για την ανανέωση του token. Με αυτό τον τρόπο μειώνεται το χρονικό διάστημα που πρέπει ο χρήστης να επαναλάβει χειροκίνητα την διαδικασία σύνδεσης του καθώς φροντίζεται αυτόματα από την εφαρμογή.

5.4.3 Διαχείριση καμπάνιας

Ο Editor page χρησιμοποιεί εκτενώς τις λειτουργίες του axios. Διαδικασίες όπως η δημιουργία εκστρατειών, φόρτωση υπαρχουσών εκστρατειών, αποθήκευση λεπτομερειών εκστρατειών καθώς και του προγραμματισμού εκστρατειών αποτελούν κάποιες από αυτές. Για την δημιουργία αυτών τον HTTP request χρησιμοποιούνται μέθοδοι HTTP όπως το GET, POST αλλά και το PUT.

5.4.4 Βελτιστοποίηση Email

Η σελίδα βελτιστοποίησης email αποτελείται από μια σελίδα χωρισμένη σε τέσσερα βήματα οπού το ένα ακολουθεί το επόμενο.

Με την χρήση κάθετου pagination υπάρχει το πρώτο βήμα κατά το οποίο ο χρήστης θέτει τα βασικά στοιχεία της καμπάνιας στα διαθέσιμα πεδία. Αυτά τα πεδία αποτελούν το όνομα της καμπάνιας, το email του αποστολέα καθώς και το θέμα (subject) του email. Παράλληλα ασύγχρονα φορτώνει και ο email επεξεργαστής, ο οποίος καλείται ως βιβλιοθήκη. Κατά την διάρκεια φόρτωσης του email επεξεργαστή εμφανίζεται στον χρήστη μια σελίδα φορτώματος. Αφού φορτώσει εμφανίζεται ένας drag and drop editor ο οποίος επιτρέπει στον χρήστη να τοποθέτηση στήλες, κουμπιά, διαχωριστικά, κεφαλίδες, κείμενο, εικόνες, καρουζέλ εικόνων, κουμπιά για μέσα μαζικής δικτυώσεις καθώς και μενού. Τέλος δίνεται η δυνατότητα στον χρήστη να προσθέσει και δικό του συγκεκριμένο κώδικα html σε περίπτωση που αυτό τον εξυπηρετεί. Στην υποσέλιδα αυτή μπορεί να σώσει μια καμπανιά ή/και να περάσει στο επόμενο βήμα το οποίο είναι η βελτιστοποίηση του mail content.

Στην επόμενη υποσέλιδα συναντάτε το βελτιστοποιημένο από το LLM κείμενο. Η υποσέλιδα χωρίζεται σε δυο τμήματα αριστερά και δεξιά. Στα αριστερά εμφανίζεται το αρχικό email όπως ο χρήστης έφτιαξε, μαζί με το spam score του από κάτω το οποίο έχει πάρει το frontend με μια κλήση POST από το backend (αποστολή email content και παραλαβή του score). Στην δεξιά σελίδα εμφανίζεται πλέον το βελτιστοποιημένο από το Large Language Model (LLM), σε αυτή την περίπτωση το DeepSeek, και από ακριβώς κάτω του το νέο spam score που λαμβάνει ξανά με το post request που αναφέρθηκε πριν. Το συγκεκριμένο βήμα αποτελεί ένα αρκετά βάρη υπολογιστικά κομμάτι της εφαρμογής και για αυτό χρειάστηκαν τρεις συγκεκριμένες ενεργείς από την μεριά του frontend. Αυξήθηκε για το συγκεκριμένο request το timeout του axios σε πολύ μεγαλύτερο χρονικό πλαίσιο από τα υπόλοιπα request καθώς ανάλογα με τον υπολογιστή που τρέχει το LLM μπορεί να χρειαστεί το frontend να περιμένει από 2 έως και 10 λεπτά. Παράλληλα Προστέθηκε κατάλληλο μήνυμα και spinner κατά την διαδικασία που το frontend περιμένει την απάντηση ώστε ο χρήστης να παραμένει ενημερωμένος για το ότι δεν έχει κολλήσει η εφαρμογή και ότι αναμένει αποτελέσματα. Η τελευταία ενεργεία αποτελεί τον μηχανισμό κατά τον οποίον γίνεται έλεγχος για το εάν το αποτέλεσμα είναι καλύτερο ή όχι. Σε περίπτωση που το αποτέλεσμα έχει το ίδιο ή χειρότερο score στην αρχική έκδοση αναλάμβανε το frontend να ξανά κάνει το ίδιο χρονοβόρο API request. Μεταφέροντας αυτή την λογική στο backend απλοποίησε αρκετά τον κώδικα του frontend ενώ παράλληλα έγινε πολύ μικρότερη η χρήση του bandwidth (έγινε μικρότερο το μέγεθος τον δεδομένων που ανταλλάζουν μεταξύ τους το frontend με το backend). Τέλος, εμφανίζονται δυο κουμπιά τα οποία επιτρέπουν στον χρήστη να επιλέξει το αρχικό του μήνυμα ή το βελτιστοποιημένο από το LLM. Στην δεύτερη περίπτωση το frontend στέλνει στο backend το νέο HTML του email το οποίο θα αντικαταστήσει το παλιό υπάρχων και θα μεταβιβάσει τον χρήστη στο επόμενο βήμα. Στην πρώτη περίπτωση απλά γίνεται μεταβίβαση του χρήστη στο επόμενο βήμα.

Στο τρίτο υποτίμημα της σελίδας αποτελεί το κομμάτι τις επιλογής ομάδων που θα αποσταλεί η καμπανιά. Σε αυτό το σημείο ο χρήστης είναι σε θέση να επιλέξει πολλαπλές ομάδες οι οποίες δεν έχουν διπλότυπες επαφές. Εμφανίζονται τα κουμπιά για αποθήκευση των επίλογων του καθώς και το να συνεχίζει στο επόμενο βήμα. Με την συνέχεια του στο επόμενο βήμα γίνεται το POST request στο backend το οποίο περιέχει τα IDs των group που έχουν επιλεχθεί να λάβουν την καμπανιά.

Τελικό βήμα αποτελεί ο προγραμματισμός της καμπάνιας. Ο χρήστης μπορεί να επιλέξει είτε να σταλθεί άμεσα, είτε με την χρήση ενός ημερολογίου να επιλέξει αρχικά την ημέρα και έπειτα την συγκεκριμένη ώρα που θα ήθελε να ξεκινήσει η αποστολή των email της καμπάνιας του. Με αυτό τον τρόπο το frontend στέλνει με ένα POST request και τα τελευταία δεδομένα που χρειάζεται το backend ώστε να

μπορεί να διαχειριστεί πληροίς την καμπανιά. Με την αποστολή του τελευταίου request ο χρήστης γίνεται απευθείας ανακατεύθυνση στην σελίδα στατιστικών για την καμπανιά που μόλις έχει δημιουργήσει. Αυτό ήταν μια επιλογή ώστε ο χρήστης να μπορεί να επαλήθευση για μια ακόμα φορά όλα τα στοιχεία που έχει εισάγει καθώς και το περιεχόμενο του email του και σε περίπτωση λάθους να μπορεί άμεσα να σταματήσει την εκτέλεση της.

Κεφάλαιο 6ο: Deployment εφαρμογής

Για την εγκατάσταση της εφαρμογής και για το deployment των backend services μπορούν να αξιοποιηθούν τρεις διαφορετικοί μέθοδοι ανάλογα με την υποδομή που είναι διαθέσιμη. Σε αυτό το κεφάλαιο θα αναλυθούν οι τρόποι με τους οποίους μπορεί να τρέξει η εφαρμογή δίνοντας μεγάλη έμφαση στα backend microservices.

- *Εγκατάσταση ως service*
- *Docker[39]*
- *Docker & Kubernetes[40]*

6.1 Εγκατάσταση ως service

6.1.1 Εγκατάσταση των jar

Για την εγκατάσταση ως services οι οδηγίες που ακολουθούν είναι κυρίως γραμμένες για συστήματα Linux και πιο συγκεκριμένα για images βασισμένα στο Debian όπως για παράδειγμα Ubuntu ή Fedora. Σε περίπτωση που κάποιος θέλει να τρέξει τα microservices σε περιβάλλον Windows γίνεται αυστηρή σύσταση για χρήση είτε του τρόπου που χρησιμοποιεί Docker είτε την χρήση του WSL (Windows Subsystem for Linux).

Η εργασία προϋποθέτει την εγκατάσταση της Java (έκδοση 17 ή νεότερη). Η εγκατάσταση της μπορεί να γίνει με την χρήση των εντολών:

- *sudo apt update*
- *sudo apt install openjdk-17-jdk*

Μετά την εγκατάσταση της μπορεί να γίνει επαλήθευση της έκδοση με την εντολή:

- *java –version*

Το επόμενο βήμα είναι να γίνει κλωνοποίηση του project πάνω από το site του GitHub για να υπάρχει πρόσβαση στον κώδικα της εργασίας. Αυτό μπορεί να γίνει με την χρήση της εντολής:

- *git clone https://gitlab.com/email-platform/backend.git*

Στην συνέχεια, μπορεί πλέον να δημιουργηθούν τα αρχεία jar τα οποία εμπεριέχουν και μπορούν να τρέξουν την εφαρμογή με την εντολή:

- *./gradlew build*

Με αυτό τον τρόπο το Gradle χτίζει όλα τα κομμάτια τις εφαρμογής που χρειάζονται για να είναι πλήρως λειτουργική η εφαρμογή. Με την χρήση του γίνονται παράλληλα build το core, ο publisher και ο executor στον φάκελο *builds/libs*. Με την χρήση των jar ο χρήστης είναι σε θέση να:

- *Ξεκινήσει την εφαρμογή με την χρήση της Java*
- *Ρύθμιση systemd service*
- *Χρήση της java σε συνδυασμό με την εντολή nohup*

6.1.2 Εγκατάσταση περιφερειακών προγραμμάτων

Για την σωστή λειτουργία της εφαρμογής αναγκαία είναι η χρήση κάποιον εξωτερικών προγραμμάτων τα οποία είναι υπεύθυνα για διάφορες λειτουργίες τις εφαρμογής. Αναγκαία αποτελούν τα εξής:

- *MySQL, βάση δεδομένων*
- *RabbitMQ, σύστημα message queue*
- *Minio, βάση δεδομένων στα πρότυπα του s3(αποθήκευση εικόνων email και επισυναπτόμενων)*
- *SpamAssassin, σύστημα ανίχνευσης spam*
- *Python για πρόσβαση στο SpamAssassin με χρήση REST API*
- *Nginx σε περίπτωση που το LLM τρέχει μέσω LM Studio*

6.1.2.1 Εγκατάσταση MySQL

Για να γίνει εγκατάσταση η MySQL[7] στον server χρησιμοποιείτε τις εντολές:

- *sudo apt update*
- *sudo apt install MySQL-server*

Το επόμενο βήμα είναι να δημιουργηθεί η βάση δεδομένων που χρειάζεται η εφαρμογή.

- *Σύνδεση στην βάση δεδομένων με την εντολή:* `mysql -u root -p`
- *Στο τερματικό της SQL:*
- *CREATE DATABASE email_platform;*
- *ALTER USER 'root'@'localhost' IDENTIFIED BY 'my_passwd';*
- *FLUSH PRIVILEGES;*

Με αυτό τον τρόπο δημιουργείται η βάση με όνομα *email_platform* και γίνεται αλλαγή του κωδικού του χρήστη σε αυτόν που μπορεί στο αρχείο παραμετροποίησης των αρχείων Jar.

6.1.3 Εγκατάσταση RabbitMQ

Το RabbitMQ[41] είναι ένας διακομιστής μηνυμάτων που επιτρέπει την ασύγχρονη επικοινωνία μεταξύ των διαφόρων μικροϋπηρεσιών της πλατφόρμας και αποτελεί αναγκαίο τμήμα της εφαρμογής.

Η εγκατάσταση της μπορεί να γίνει με την χρήση:

- *sudo apt update*
- *sudo apt install rabbitmq-server*

Επίσης μπορεί να γίνει και εγκατάσταση του προσθετού διαχείρισης που δίνει την δυνατότητα στον χρήστη να δημιουργεί clusters καθώς και να ελέγχει διαφορά στατιστικά όπως την κατάσταση των ουρών, τα μηνύματα που επεξεργάζονται ανά δευτερόλεπτο και αλλά:

- *sudo rabbitmq-plugins enable rabbitmq_management*

Μετά την ενεργοποίηση του προσθέτου ο χρήστης μπορεί να έχει πρόσβαση στο περιβάλλον διαχείρισης μέσω του προγράμματος περιήγησης στη διεύθυνση http://localhost:15672 με προεπιλεγμένα διαπιστευτήρια: *guest/guest*

6.1.4 Εγκατάσταση του MinIO

Το MinIO[42] είναι ένας συμβατός με το Amazon S3[43] διακομιστής αποθήκευσης αντικειμένων που χρησιμοποιείται για την αποθήκευση αρχείων, όπως συνημμένα email και εικόνες. Δεν είναι απαραίτητος αλλά αποτελεί μια αρκετά καλή λύση ως CDN (Content Delivery Network). Στην θέση του μπορεί να μην χρησιμοποιηθεί κάποιος ή να μπει οποιοσδήποτε άλλος S3.

Για την εγκατάσταση του ακολουθούνται τα εξής βήματα:

- *Εγκατάσταση του MinIO:*
  - *wget https://dl.min.io/server/minio/release/linux-amd64/minio*
  - *chmod +x minio*
  - *sudo mv minio /usr/local/bin/*

- *Δημιουργία ενός καταλόγου για τα δεδομένα του MinIO. Αυτός ο κατάλογος θα χρησιμοποιηθεί για την αποθήκευση όλων των αντικειμένων που θα αποθηκευτούν στο MinIO.*
    - *mkdir -p ~/minio/data*
- *Εκκίνηση του MiniIO:*
    - *minio server ~/minio/data --console-address ":9001" --address ":9003"*
- *Πρόσβαση στην κονσόλα του MinIO στη διεύθυνση http://localhost:9001 και σύνδεση με τα προεπιλεγμένα διαπιστευτήρια:*
    - *Όνομα χρήστη: minioadmin*
    - *Κωδικός πρόσβασης: minioadmin*

Μετά τη σύνδεση, συνιστάται να αλλάξετε τα προεπιλεγμένα διαπιστευτήρια για λόγους ασφαλείας, ειδικά σε περιβάλλοντα παραγωγής.

6.1.5 Εγκατάσταση SpamAssassin

Το SpamAssassin είναι ένα εργαλείο ανίχνευσης spam που χρησιμοποιείται για τον έλεγχο των email πριν από την αποστολή τους. Επειδή το SpamAssassin δεν λειτουργεί με απευθείας HTTP χρειάζεται πέρα από την εγκατάσταση του ιδίου του SpamAssassin και η γλώσσα προγραμματισμού python3.

- *Εγκατάσταση του SpamAssassin:*
    - *sudo apt update*
    - *sudo apt install spamassassin spamc*
- *Ενεργοποίηση και εκκίνηση του SpamAssassin:*
    - *sudo systemctl enable spamassassin*
    - *sudo systemctl start spamassassin*
- *Εγκαταστήστε την Python και το Flask για το API του SpamAssassin:*
    - *sudo apt install python3 python3-pip*
    - *pip3 install flask*
- *Δημιουργία εφαρμογής Flask ως API του SpamAssassin:*
    - *mkdir -p ~/spamassassin-api*
    - *cd ~/spamassassin-api*
- *Δημιουργία ενός αρχείου με όνομα app.py με περιεχόμενο του Δημιουργία ενός αρχείου με όνομα app.py με περιεχόμενο του ~/docker/spamassassin/app.py Στο αρχείο αυτό περιέχεται το API που μπορεί να γίνει προσβάσιμο το spamassassin μέσω HTTP. Αυτό το script δημιουργεί ένα απλό API με ένα τελικό σημείο /check που δέχεται περιεχόμενο email σε μορφή JSON, το αποθηκεύει προσωρινά σε ένα αρχείο και το στέλνει στο SpamAssassin για έλεγχο. Στη συνέχεια, αναλύει την έξοδο και επιστρέφει τα αποτελέσματα σε μορφή JSON.*
- *Εκκίνηση του API για το SpamAssassin με την εντολη:* `python3 app.py` *. Αυτή η εντολή ξεκινά τον διακομιστή Flask που φιλοξενεί το API. Ο διακομιστής θα ακούει στη θύρα 5000 και θα δέχεται αιτήσεις από οποιαδήποτε διεύθυνση IP.*

6.1.6 Εγκατάσταση Nginx Proxy για το LM Studio

Κατά την ανάπτυξη της εφαρμογής χρησιμοποιήθηκε το LM Studio με σκοπό να μπορεί να τρέξει μέσα σε αυτό το μοντέλο μηχανικής μάθησης για την βελτιστοποίηση των email. Το LM Studio παρέχει την δυνατότητα παροχής OpenAI API στο οποίο μπορεί να συνδεθεί το τμήμα της εφαρμογής του core με όνομα SpringAI. Με το τελευταίο update του SpringAI δεν γίνεται υποστήριξη για το HTTP/1.1 αλλά μόνο για την ανανεωμένη θεώρηση του HTTP/2 το οποίο το LM Studio δεν υποστηρίζει. Έτσι έπρεπε να δημιουργηθεί μια γέφυρα μεταξύ των 2 υπηρεσιών με την χρήση του nginx.

- *Εγκατάσταση Nginx:*
    - *sudo apt update*
    - *sudo apt install nginx*
- *Παραμετροποίηση του nginx.conf αρχείου:*

```
server {

    listen 8087;

    location / {

      proxy_pass http://127.0.0.1:1234;

      proxy_set_header Host $host;

      proxy_set_header X-Real-IP $remote_addr;

  }
}
```

Πίνακας 16: Nginx configuration

Αυτή η ρύθμιση δημιουργεί έναν διακομιστή που ακούει στη θύρα 8087 και προωθεί όλες τις αιτήσεις στο LM-Studio που τρέχει στη θύρα 1234. Οι κεφαλίδες Host και X-Real-IP διατηρούνται για σωστή λειτουργία.

- *Επανεκκίνηση του Nginx*
    - *sudo systemctl restart nginx*

Με αυτόν τον τρόπο γίνεται εύκολη η πρόσβαση του μοντέλου μηχανικής μάθησης που τρέχει στο LM Studio μέσω του SpringAI OpenAI API. Σε περίπτωση που δεν χρησιμοποιηθεί το LM Studio η χρήση του nginx δεν είναι υποχρεωτική.

6.1.7 Εκτέλεση Microservices

Μετά την εγκατάσταση και ρύθμιση όλων των υποστηρικτικών υπηρεσιών, είναι δυνατή η εκκίνηση των κύριων microservices της πλατφόρμας.

- *Εκτέλεση του **core**: java -jar build/libs/core-0.0.1-SNAPSHOT.jar. Η υπηρεσία Core είναι η κεντρική υπηρεσία της πλατφόρμας που διαχειρίζεται τους χρήστες, τις καμπάνιες και άλλες βασικές λειτουργίες. Ακούει στη θύρα 8080 από προεπιλογή.*
- *Εκτέλεση της υπηρεσίας **publisher**: java -jar build/libs/publisher-0.0.1-SNAPSHOT.jar. Η υπηρεσία Publisher είναι υπεύθυνη για τη δημιουργία και τον προγραμματισμό των καμπανιών email. Ακούει στη θύρα 8081 από προεπιλογή.*
- *Εκτέλεση της υπηρεσίας **executor**: java -jar build/libs/executor-0.0.1-SNAPSHOT.jar. Η υπηρεσία Executor είναι υπεύθυνη για την πραγματική αποστολή των email και την επεξεργασία των αναφορών παράδοσης. Θα ακούει στη θύρα 8082 από προεπιλογή.*

Περεταίρω μπορεί να χρησιμοποιηθεί και η εντολή nohup μαζί με την εντολή java ή να δημιουργηθούν services του *systemd* για να μπορούν να τρέχουν κανονικά στο παρασκήνιο.

6.1.7.1 Ρύθμιση παραμέτρων

Η εφαρμογή χρησιμοποίει το σύστημα ρύθμισης παραμέτρων του Spring boot. Με αυτόν τον τρόπο ο χρήστης μπορεί να προσαρμόσει ανά πάσα στιγμή τις ρύθμισης με τους εξής τρόπους:

- *Επεξεργασία του αρχείου **application.properties** στον κατάλογο src/main/resources πριν από τη δημιουργία.*
- *Παροχή μεταβλητών περιβάλλοντος κατά την εκτέλεση των αρχείων JAR.*

- *Δημιουργία ενός αρχείου **application.properties** στον ίδιο κατάλογο με τα αρχεία JAR.*

```
// Βάση δεδομένων
spring.r2dbc.username=root
spring.r2dbc.password=my_passwd
spring.r2dbc.url=r2dbc:mysql://localhost/email_platform
// MongoDB
spring.data.mongodb.host=localhost
spring.data.mongodb.port=27017
// RabbitMQ
spring.rabbitmq.host=localhost
spring.rabbitmq.port=5672
spring.rabbitmq.username=guest
spring.rabbitmq.password=guest
// MinIO
minio.host=http://localhost:9003
minio.key.access=minioadmin
minio.key.secret=minioadmin
// LLM
spring.ai.openai.api-key=lm-studio
spring.ai.openai.base-url=http://127.0.0.1:8087
```

Πίνακας 17: Βασικές ιδιότητες ρύθμισης στο application.properties

6.2 Docker

Ο δεύτερος δυνατός τρόπος να τρέξει κάποιος την εφαρμογή είναι με την χρήση του Docker. Το Docker αποτελεί ένα πολύ χρήσιμο εργαλείο για να μπορεί ο χρήστης να τρέξει, με ελάχιστη παραμετροποίηση του συστήματος του καθώς και αυτής, την εφαρμογή και να ελέγξει την λειτουργικότητά της. Προτείνεται για περιβάλλοντα δοκίμων, ανάπτυξης ή περιβάλλοντα παραγωγής τα οποία ο προγραμματιστής γνωρίζει ότι δεν θα υπάρχει πολύ μεγάλος φόρτος εργασίας. Το Docker παρέχει ένα συνεπές και απομονωμένο περιβάλλον για όλες τις υπηρεσίες.

6.2.1 Προ απαιτούμενα

Κάτ. ελάχιστο χρειάζεται να γίνει εγκατάσταση του Docker για τη δημιουργία και εκτέλεση των containers καθώς και το Docker Compose ώστε να μπορούν να οριστούν και να διαχειριστούν πολλαπλά containers.

6.2.1.1 Εγκατάσταση

Η εγκατάσταση του Docker γίνεται με τις εντολές[44]:

- *sudo apt update*
- *sudo apt install docker.io*

- *sudo systemctl enable docker*
- *sudo systemctl start docker*

Ενώ για το περιβάλλον τον Windows πρέπει να ενεργοποιηθεί το WSL (Windows Subsystem for Linux)[45] και να κατεβάσει κάποιος το Docker από το επίσημο site.

6.2.2 Ανάπτυξη

Για να μπορεί κάποιος να τρέξει το πρόγραμμα υπάρχουν δυο διαφορετικοί τρόποι. Ο πρώτος είναι να φτιάξει το δικό του *docker-compose.yml* αρχείο ενώ ο δεύτερος να χρησιμοποιηθεί αυτό το οποίο υπάρχει μέσα στο Project. Προτείνεται να χρησιμοποιηθεί αυτό που βρίσκεται μέσα στο project για πολύ πιο εύκολη εμπειρία καθώς είναι ήδη παραμετροποιημένο όπως θα έπρεπε.

Για να ξεκινήσει κάποιος τις υπηρεσίες στο root φάκελο του project αρκεί να τρέξει την παρακάτω εντολή. Με την χρήση αυτής ξεκινάνε όλες οι διεργασίες που έχουν οριστεί μέσα στο αρχείο του docker-compose.yml .

- *docker compose up –d*

Έλεγχος για το εάν τρέχουν οι διεργασίες μπορεί να γίνει με την εντολή:

- *docker compose ps*

Οι εικόνες (images) των microservices μπορούν να φτιαχτούν είτε τοπικά, είτε κάποιος να τις κατεβάσει από το image store του GitLab. Για να μπορεί κάποιος προγραμματιστής να φτιάξει κάποιο εξατομικευμένο image με κάποια δική του αλλαγή μπορεί με την χρήση του Gradle task, σε κάθε ένα από τα microservices:

- *./gradlew bootBuildImage*

Με αυτό τον τρόπο το Gradle είναι ρυθμισμένο να φτιάχνει πρώτα το Jar αρχείο και έπειτα να φτιάχνει ένα image το οποίο τρέχει εντός του ένα μικρό περιβάλλον Linux (συνήθως το alpine Linux) και την εφαρμογή με την χρήση του Jar.

Σε περίπτωση που θέλει να γίνει publish το image στο repository του GitLab μπορεί να χρησιμοποιηθεί το custom Gradle task:

- *./gradlew buildAndPushAllImages*
- *./gradlew buildAndPushImage*

Στην πρώτη περίπτωση το Gradle θα φτιάξει όλα τα images για όλα τα microservices και θα τα ανεβάσει στο repository του GitLab και θα γίνουν διαθέσιμα για κατέβασμα από τα άτομα που έχουν πρόσβαση στο repository (read access). Ενώ, στην πρώτη περίπτωση θα φτιάξει μεμονωμένα ένα image και θα το ανεβάσει στο repository ανάλογα για ποιο microservice έχει τρέξει αυτό το Gradle task. Για το ανέβασμα ενός image στο GitLab πρέπει αν έχει προηγηθεί η σύνδεση του χρήστη με το GitLab με την εντολή:

- *docker login*

6.2.3 Πρόσβαση στις Υπηρεσίες

Μετά την επιτυχή εκκίνηση όλων των containers, ο χρήστης μπορεί να αποκτήσει πρόσβαση στις διάφορες υπηρεσίες μέσω των ακόλουθων διευθύνσεων:

- ***Core API**: http://localhost:8080*
- ***Publisher API**: http://localhost:8081*
- ***Executor API**: http://localhost:8082*
- ***RabbitMQ Management**: http://localhost:15672 (guest/guest)*

- ***MinIO Console**: http://localhost:9001 (minioadmin/minioadmin)*
- ***Fake SMTP Web Interface**: http://localhost:1080*

Αυτές οι διευθύνσεις είναι προσβάσιμες από τον υπολογιστή του χρήστη. Οι θύρες που καθορίζονται στο αρχείο docker-compose.yml αντιστοιχίζονται από τα containers στον υπολογιστή, επιτρέποντας την πρόσβαση στις υπηρεσίες.

6.3 Kubernetes

Στο κεφάλαιο αυτό θα γίνει αναφορά στο πως μπορεί ο χρήστης να κάνει deploy την εφαρμογή με την χρήση του Kubernetes[40]. Το Kubernetes αποτελεί ένα εργαλείο διαχειρίσης και ενορχήστρωσης images παρέχοντας την δυνατότητα να γίνεται αυτόματα κλιμάκωση των microservices ανάλογα με τον φόρτο εργασίας που υπάρχει.

6.3.1 Προ απαιτούμενα

Για να μπορεί να γίνει το deploy θα πρέπει ο χρήστης να έχει εγκατεστημένο κάποιο εργαλείο Kubernetes ή να έχει πρόσβαση σε ανάλογη cloud εγκατάσταση όπως το AKS (Azure Kubernetes Service), GKS (Google Kubernetes Service) ή το EKS (Amazon Elastic Kubernetes Service). Το συγκεκριμένο κεφάλαιο θα αναφερθεί κύριως σε εγκατάσταση σε τοπικό Kubernetes με την χρήση του MiniKube, kubectl και του kustomize.

Το MiniKube αποτελεί ένα εργαλείο το οποίο επιτρέπει στον χρήστη να τρέξει τοπικά ένα instance του Kubernetes. Στοχεύει κυρίως σε development περιβάλλοντα καθώς και σε χρήστες που μαθαίνουν Kubernetes. Για την διαχείριση του MiniKube χρησιμοποιείτε το εργαλείο Kubectl που μέσα από αυτό μπορεί ο χρήστης να συνδεθεί με το MiniKube ή κάποιο άλλο Kubernetes instance και να διαχειριστεί τα containers που τρέχουν σε αυτό.

6.3.1.1 Εγκατάσταση Kubectl

Για την εγκατάσταση του Kubectl σε Linux αρκεί να ακολουθηθούν τα τρία βήματα:

- *curl -LO https://dl.k8s.io/release/$(curl -L-s https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl*
- *chmod +x kubectl*
- *sudo mv kubectl /usr/local/bin/*

6.3.1.2 Εγκατάσταση MiniKube

Για την εγκατάσταση του MiniKube σε Linux αρκεί να ακολουθηθούν τα εξής βήματα:

1. *curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube_latest_amd64.deb*
2. *sudo dpkg -i minikube_latest_amd64.deb*

Έπειτα, με την χρήση της εντολής minikube start μπορεί ο χρήστης να ξεκινήσει το τοπικό Kubernetes cluster του. Σε περίπτωση που πρέπει να γίνει παραμετροποίηση της διαθέσιμης μνήμης και CPU αυτό είναι εφικτό με τις παραμέτρους –memory=value για περιορισμό της μνήμης (πχ –memory==4096 για περιορισμό της στα 4GB), ενώ μπορεί να γίνει ρύθμιση των διαθέσιμων CPU με την χρήση της –cpus=values (πχ –cpu=2 για ρύθμιση χρήσης μόνο 2 CPU).

6.3.1.3 Ανάπτυξη

Ως πρώτο βήμα πρέπει να γίνει κλωνοποίηση του project από το GitLab:

- *git clone https://gitlab.com/email-platform/backend.git*

Έπειτα, στον φάκελο Kubernetes μπορούν να βρεθούν όλα τα απαραίτητα αρχεία για να μπορεί να τρέξουν όλα τα microservices καθώς και οι υπηρεσίες υποστηρίξεις όπως για παράδειγμα βάσεις δεδομένων. Με την χρήση της εντολής:

- *kubectl apply -k .*

Με την παραπάνω εντολή θα εκτελεστούνε οι εντολές που υπάρχουν μέσα στο αρχείο *kustomization.yml*. Αυτό το αρχείο συγχωνεύει όλα τα αρχεία YAML και τα εφαρμόζει στο Kubernetes cluster. Με αυτό τον τρόπο δημιουργούνται όλα τα απαραίτητα deployments, services καθώς και τα PersisntendVolumeClaims τα οποία διατηρούν και κρατούν τα δεδομένα τα οποία πρέπει να σώζονται πάντα (πχ βάση δεδομένων MySQL, καθώς θέλουμε να είναι διαθέσιμη ακόμα και μετρά από κάποιο restart ή παύση λειτουργείας της εφαρμογής).

Για να γίνει έλεγχος της κατάστασης των deployment αυτό μπορεί να γίνει με τις εντολές:

- *kubectl get deployments*
- *kubectl get pods*
- *kubectl get services*

Οι εντολές αυτές εμφανίζουν την κατάσταση διάφορων πόρων Kubernetes. Με αυτές μπορεί κάποιος να επιβεβαιώσει ότι όλα τα pods είναι σε κατάσταση “Running” και όλα τα deployments είναι «Available». Σε περίπτωση που υπάρχει κάποιο πρόβλημα με την χρήση της παρακάτω εντολής μπορεί κάποιος να έχει πρόσβαση στα σφάλματα λάθους του συγκεκριμένου pod καθώς και στα logs αυτού:

- *kubectl describe pod <pod-name>*

Το Kubernetes βασίζεται στα images που παράγονται μέσω του Docker.

6.3.1.4 Πρόσβαση στις υπηρεσίες

Από προεπιλογή, οι υπηρεσίες εκτίθενται εντός του Kubernetes cluster μόνο. Για να μπορεί κάποιος να έχει πρόσβαση σε αυτές απέξω αρκεί να ακολουθηθούν ένας από τους δυο διαθέσιμους τρόπους. Ο πρώτος αφορά κύριος το Minikube και προτείνεται για development περιβάλλοντα κύριος και ο δεύτερος ο οποίο χρησιμοποιεί το ingress. Για τον δεύτερο τρόπο συγκεκριμένα χρειάζεται ένα αρχείο ρυθμίσεων το οποίο είναι διαθέσιμο στον φάκελο Kubernetes.

Για την πρώτη περίπτωση πρέπει να χρησιμοποιηθεί το port-forward που παρέχει το εργαλείο kubectl. Στην περίπτωση αυτού του project γίνεται με τις εντολές:

- *kubectl port-forward service/core 8080:8080*
- *kubectl port-forward service/publisher 8081:8081*
- *kubectl port-forward service/executor 8082:8082*
- *kubectl port-forward service/minio 9003:9003 9001:9001*
- *kubectl port-forward service/rabbitmq 15672:15672*
- *kubectl port-forward service/fake-mail-server 1080:1080*

Με αυτό τον τρόπο γίνονται διαθέσιμες στο τοπικό δυτικό όλες οι υπηρεσίες που τρέχουν εντός του MiniKube.

Για την ρύθμιση μέσω ingress αρκεί απλά να εκτελεστεί ξανά η εντολή apply με όρισμα το αρχείο που περιέχει τις ρυθμίσεις για το ingress:

1. *kubectl apply -f ingress.yaml*

Αυτή η εντολή εφαρμόζει τον πόρο Ingress, ο οποίος ρυθμίζει έναν εξωτερικό ελεγκτή εισόδου για να δρομολογεί την εξωτερική κυκλοφορία στις κατάλληλες υπηρεσίες εντός του cluster.

6.3.1.5 Κλιμάκωση των υπηρεσιών

Ένα από τα μεγάλα θετικά που παρέχει το Kubernetes έναντί άλλον λύσεων είναι το γεγονός ότι μπορεί να κάνει αυτόματη κλιμάκωση των microservices ανάλογα με τον φόρτο εργασίες που υπάρχει εκείνη την στιγμή. Αυτό μπορεί να επιτευχθεί με την χρήση του Horizontal Pod Autoscaler, το οποίο προσαρμόζει τον αριθμό των αντιγράφων με βάση τη χρήση CPU ή κάποιας άλλης μετρικής. Σε περίπτωση που χρειάζεται συγκεκριμένη ρύθμιση για τα ποσά deployments είναι διαθέσιμα αυτό μπορεί να γίνει και χειροκίνητα με την χρήση της εντολής scale. Με αυτό τον τρόπο μπορεί να ρυθμιστεί ο αριθμός κάθε deployment ξεχωριστά ανάλογα με τον όγκο ή τον τύπο εργασίες του κάθε microservice.

1. *kubectl scale deployment core --replicas=1*
2. *kubectl scale deployment publisher --replicas=2*
3. *kubectl scale deployment executor --replicas=3*

6.3.2 Συμπεράσματα

Σε αυτό το κεφάλαιο υπάρχουν αναλυτικές οδηγίες για την εγκατάσταση της εφαρμογής χρησιμοποιώντας τρεις μεθόδους:

- *Εγκατάσταση ως **Υπηρεσία***
- *Εγκατάσταση με **Docker***
- *Εγκατάσταση με **Kubernetes***

Η επιλογή μεθόδου εγκατάστασης έχει να κάνει κυρίως με τις απαιτήσεις και την χρήση που πρόκειται να γίνει στην εφαρμογή. Η κάθε μια αποτελεί λύση για ένα διαφορετικό πρόβλημα ανάλογα την χρήση που θα πρέπει να γίνει.

Η εγκατάσταση ως **υπηρεσία** παρέχει τον μεγαλύτερο έλεγχο της εφαρμογής, απαιτεί περισσότερη χειροκίνητη ρύθμιση, λειτουργικό σύστημα Linux καθώς και πολύ μεγάλη εξοικείωση με αυτό.

Η εγκατάσταση με **Docker** παρέχει καλή απομόνωση της εφαρμογής καθώς και ευκολία στην εγκατάσταση της. Παρέχει ένα πλήρες λειτουργικό περιβάλλον άμεσα χωρίς να χρειαστεί να επηρεαστεί ή να εγκατασταθεί τίποτε άλλο (εκτός του docker) στον host. Το βασικό αρνητικό αυτής της μεθόδου αποτελεί το γεγονός ότι η κλιμάκωση του είναι δύσκολη σε περιπτώσεις που χρειάζεται περισσότερους πόρους από αυτούς που έχει σχεδιαστεί αρχικά να έχει. Προτείνεται κυρίως για περιβάλλοντα δοκίμων καθώς και για μικρά releases.

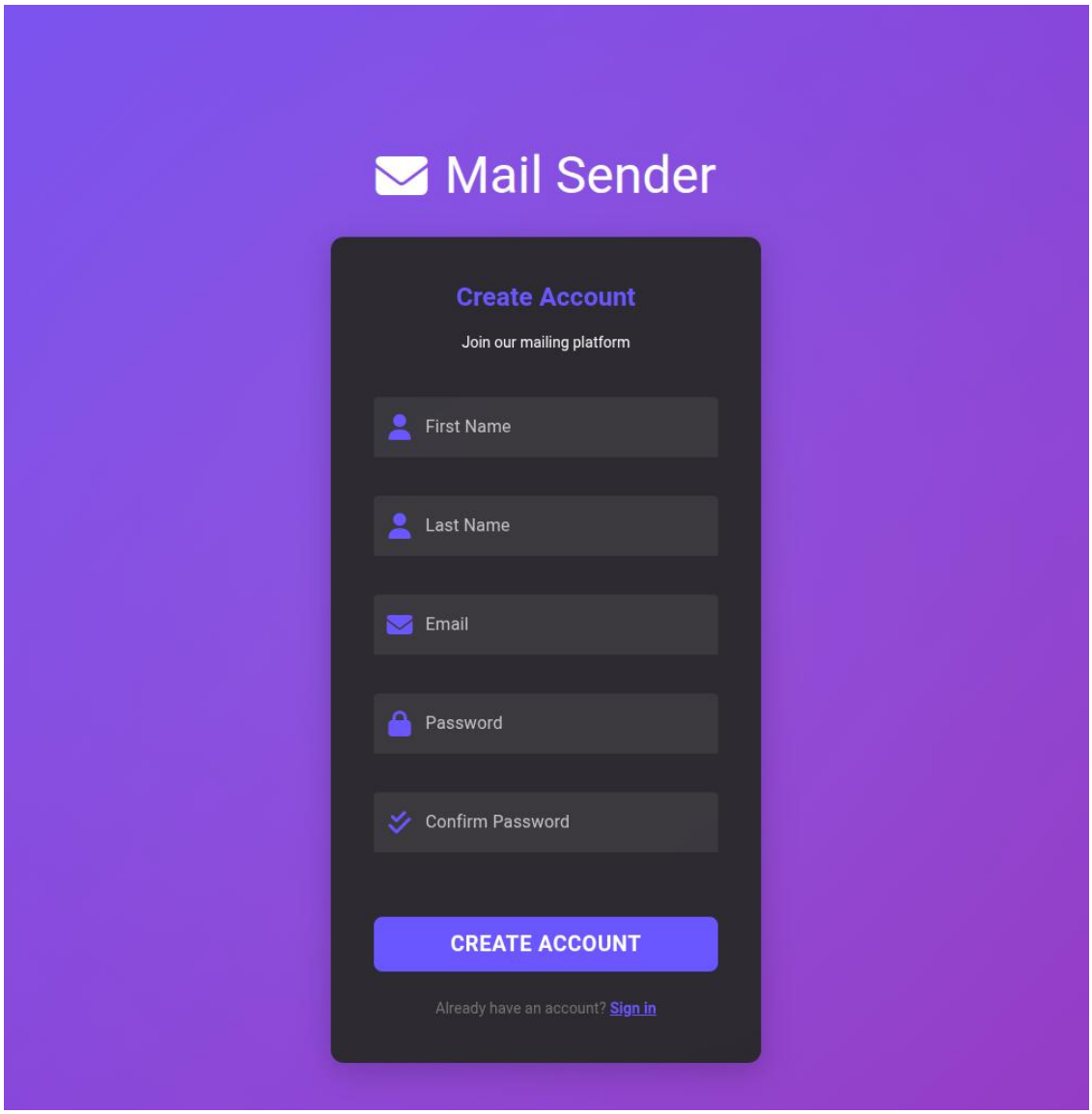
Τέλος, η εγκατάσταση μέσω **Kubernetes** ενσωματώνει τα θετικά στοιχεία του Docker και λύνει το βασικό πρόβλημα του, την κλιμάκωση. Το μεγάλο αρνητικό του είναι το γεγονός ότι είναι δύσκολη η χρήση του και η παραμετροποίηση του σε περίπτωση που κάποιος δεν είναι εξοικειωμένος με αυτό. Προτείνεται για περιβάλλοντα παραγωγής για χρήση στο cloud καθώς μπορεί να «κλείσουν» instances όταν αυτά δεν χρησιμοποιούνται μειώνοντας δραστικά το κόστος.

Κεφάλαιο 7ο: Χρήση εφαρμογής

7.1 Σελίδα εγγραφής χρήστη

Το πρώτο βήμα που πρέπει να κάνει ένας χρήστης προκειμένου να χρησιμοποιήσει την εφαρμογή είναι να δημιουργήσει έναν λογαριασμό χρήστη. Στην σελίδα εγγραφής ο χρήστης καλείται να καταχωρήσει τα εξής στοιχεία:

- *Όνομα*
- *Επώνυμο*
- *Διεύθυνση email*
- *Κωδικός πρόσβασης*
- *Επιβεβαίωση κωδικού πρόσβασης*

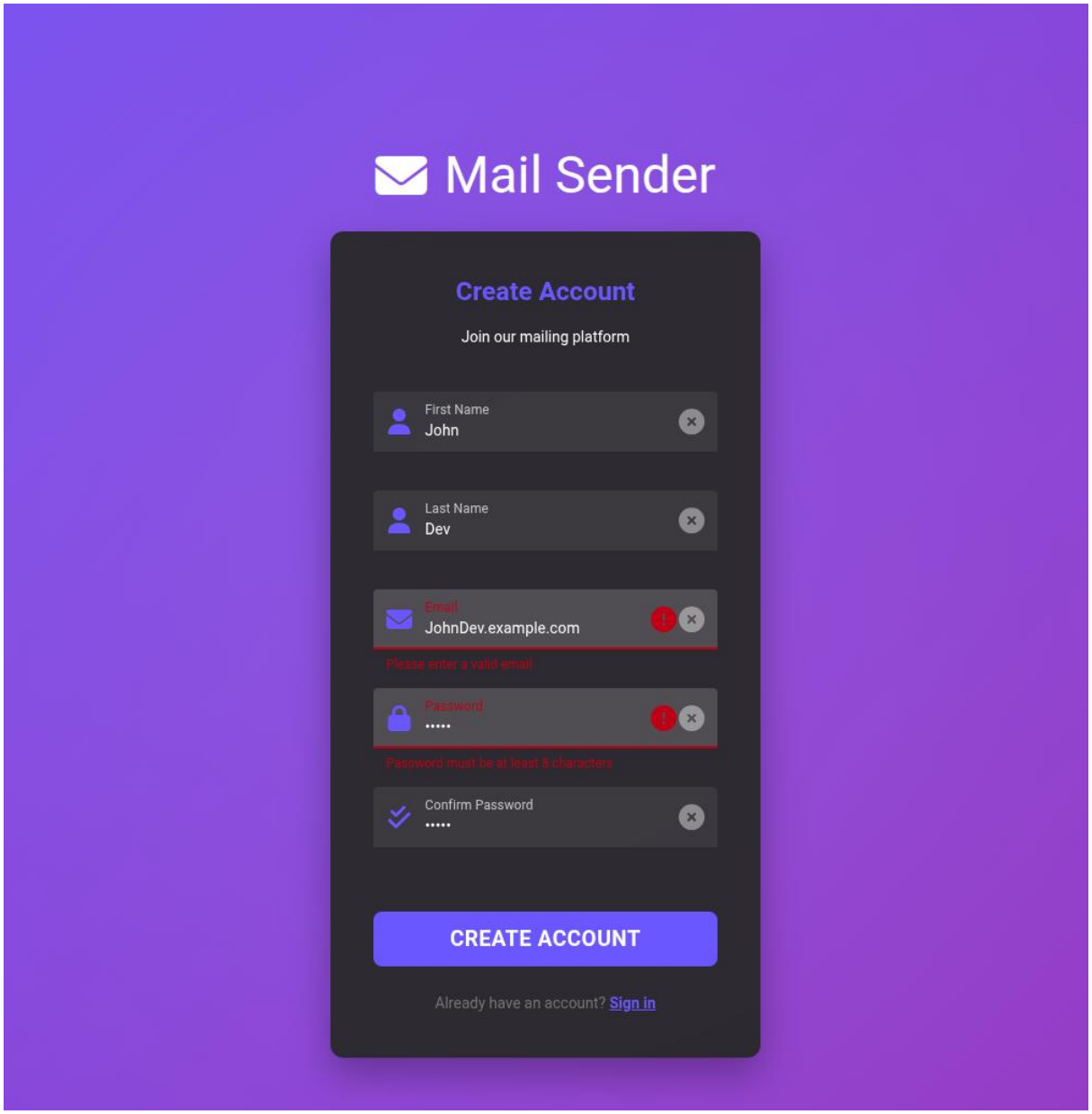


Εικόνα 9: Σελίδα εγγραφής χρήστη

Η φόρμα έχει τους κατάλληλους κανόνες επικύρωσης ώστε να αποτρέψει τον χρήστη να την συμπληρώσει με μη έγκυρα στοιχεία την φόρμα εγγραφής. Οι κανόνες επικύρωσης είναι οι εξής:

- *Η διεύθυνση email πρέπει να έχει μια έγκυρη μορφή διευθυνσης email*
- *Ο κωδικός πρέπει να είναι το ελάχιστο 8 χαρακτήρες*
- *Το πεδίο επιβεβαίωσης κωδικού πρέπει να συμπίπτει με το πεδίο κωδικού*

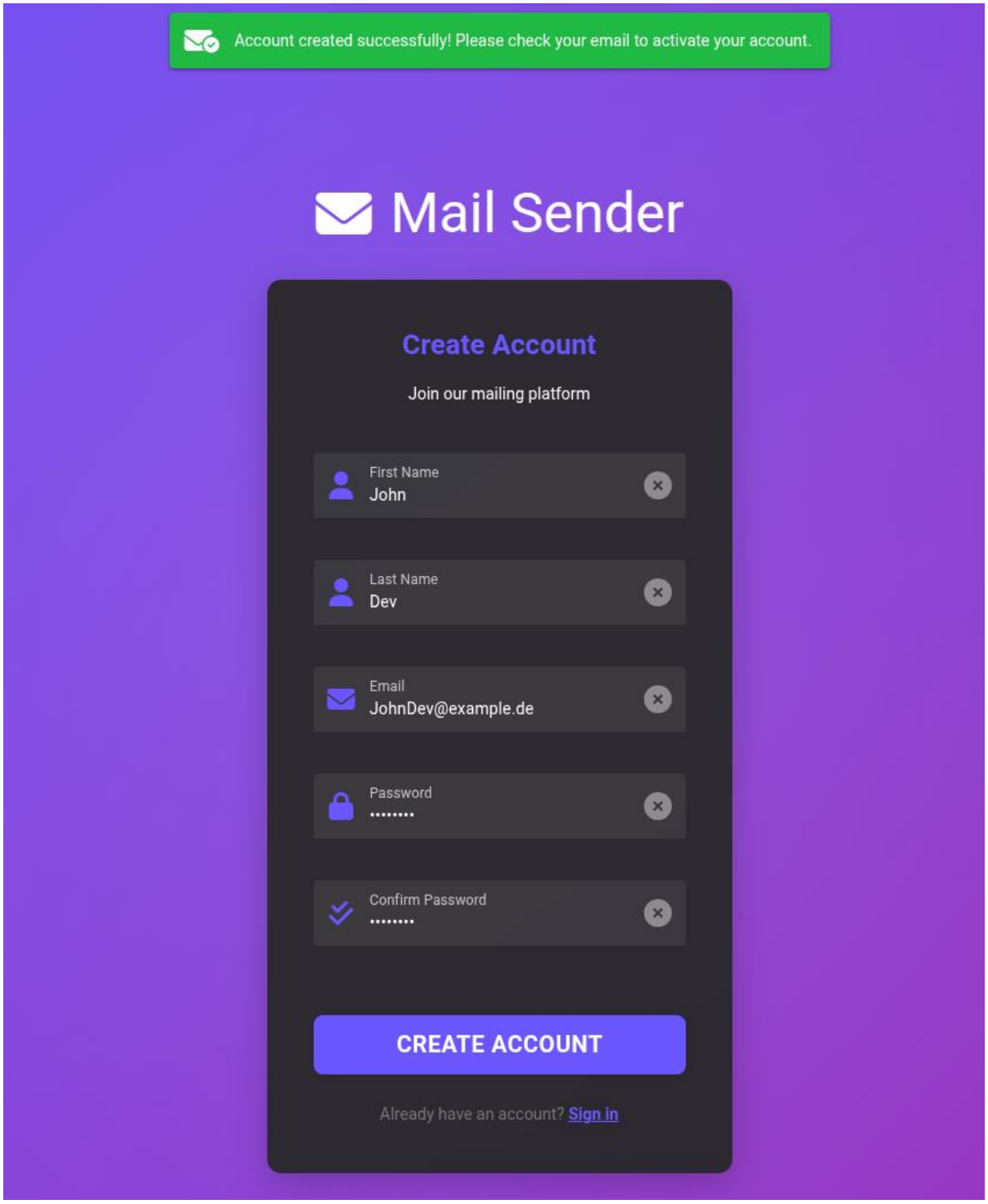
Σε περίπτωση αποτυχίας επικύρωσης ενός πεδίου, ο χρήστης ενημερώνεται με το κατάλληλο μήνυμα, με σαφής οδηγίες για την σωστή συμπλήρωση του πεδίου που δεν επικυρώθηκε.



Εικόνα 10: Επικύρωση στοιχείων στην σελίδα εγγραφής

Αφού ο χρήστης συμπληρώσει σωστά την φόρμα εγγραφής, στην συνέχεια ολοκληρώνει την εγγραφή του πατώντας το κουμπί *'Create account'*.

Αφού δημιουργηθεί επιτυχώς ο λογαριασμός ενημερώνεται ότι πρέπει να ενεργοποιήσει τον λογαριασμό του πατώντας τον σύνδεσμο που του έχει σταλεί στην διεύθυνση email του.

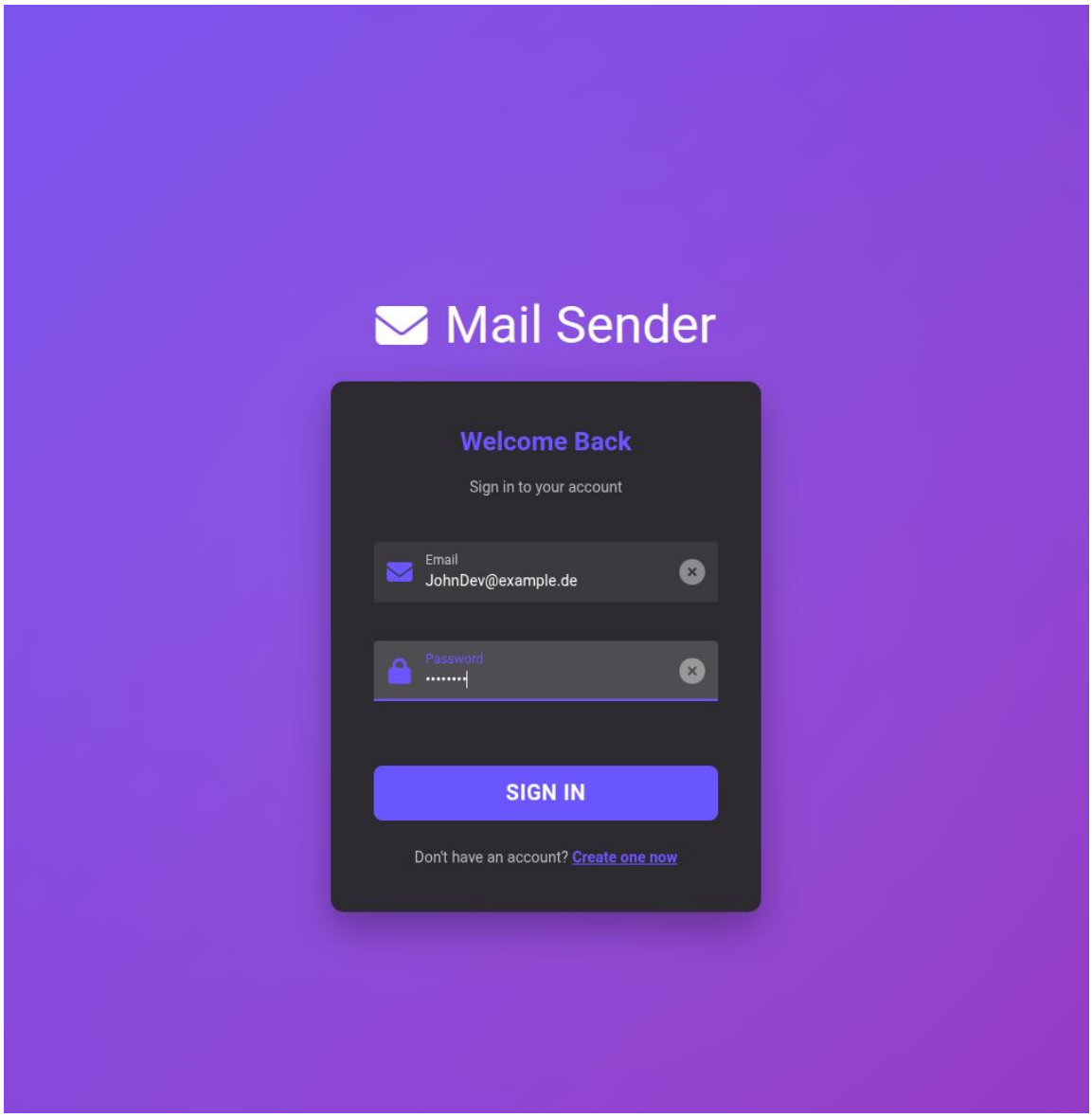


Εικόνα 11: Επιτυχής εγγραφή

7.2 Σελίδα εισόδου χρήστη

Εφόσον ένας χρήστης έχει κάνει εγγραφή και επιβεβαίωση του λογαριασμού του, τότε προκειμένου να ξεκινήσει να χρησιμοποιεί την εφαρμογή, πρέπει να αυθεντικοποιηθεί μέσω της σελίδας εισόδου χρήστη.

Στην σελίδα αυτή καταχωρεί την διεύθυνση email του, με την οποία έκανε εγγραφή, καθώς και τον κωδικό χρήστη.



Εικόνα 12: Σελίδα εισόδου χρήστη

Σε περίπτωση που καταχωρήσει λάθος στοιχεία, τότε εμφανίζεται το κατάλληλο μήνυμα ώστε να ενημερωθεί ο χρήστης και να ξαναπροσπαθήσει. Αν η είσοδος χρήστη είναι επιτυχής, και ο χρήστης αυθεντικοποιηθεί, τότε ο χρήστης ανακατευθύνεται στην αρχική σελίδα της εφαρμογής.

7.3 Αρχική σελίδα (Homepage)

Η αρχική σελίδα είναι η σελίδα η οποία παρέχει όλα τα εργαλεία στον χρήστη για να χρησιμοποιήσει την εφαρμογή.

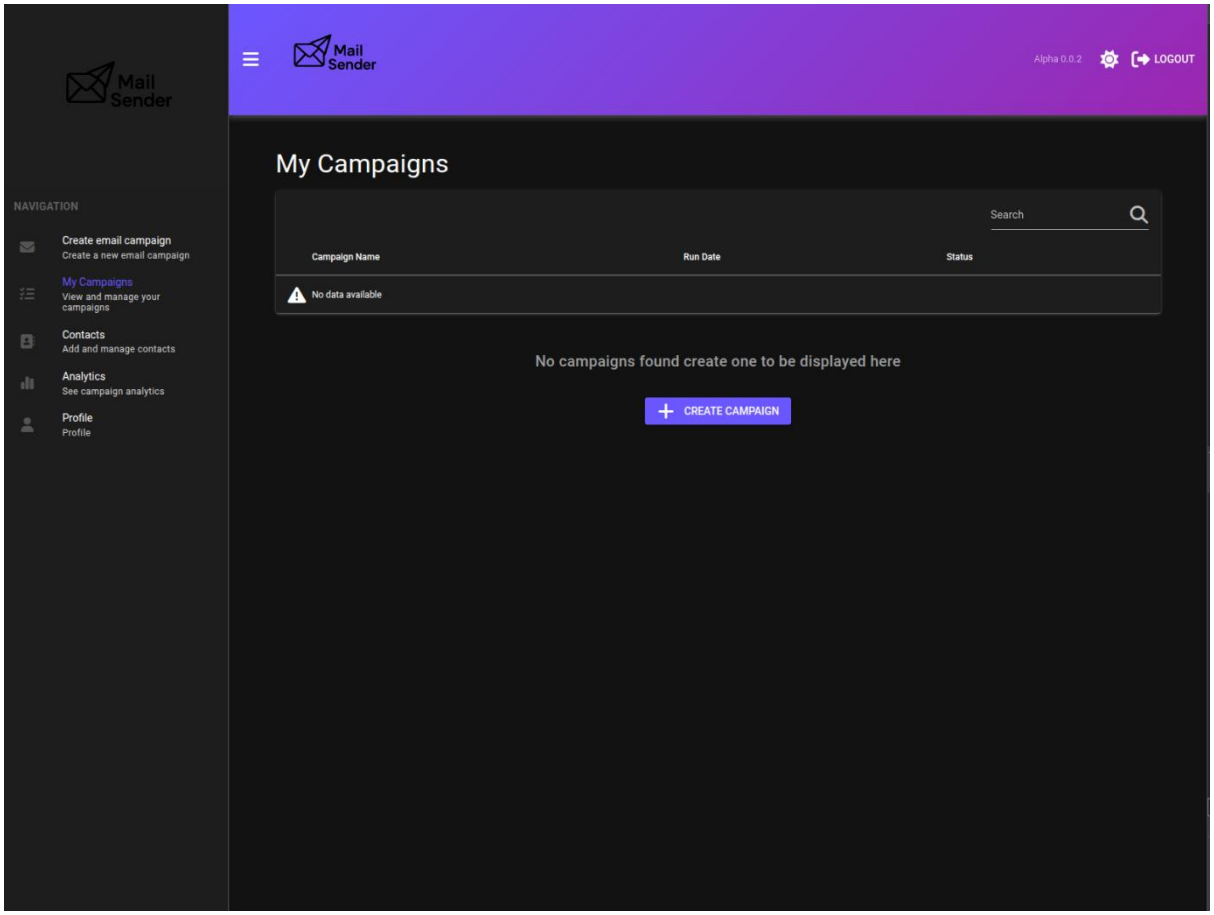
Στα αριστερά, ένα μενού πλοήγησης επιτρέπει στον χρήστη να περιηγηθεί και να εκτελέσει όλες τις λειτουργίες που δίνει η εφαρμογή, καθώς και να δεί στατιστικά των καμπανιών που έχει στείλει, και να διαχειριστεί τον λογαριασμό του.

Πιο συγκεκριμένα το μενού πλοήγησης περιέχει τα εξής:

- *Δημιουργία καμπάνιας*
- *Σύνοψη των καμπανιών χρήστη*
- *Διαχείριση επαφών*
- *Αναλυτικά στοιχεία αποστολής καμπανιών*
- *Διαχείριση λογαριασμού*

Η προεπιλεγμένη σελίδα, που αντικρίζει ο χρήστης αφού κάνει είσοδο λογαριασμού, είναι η σελίδα σύνοψης καμπανιών.

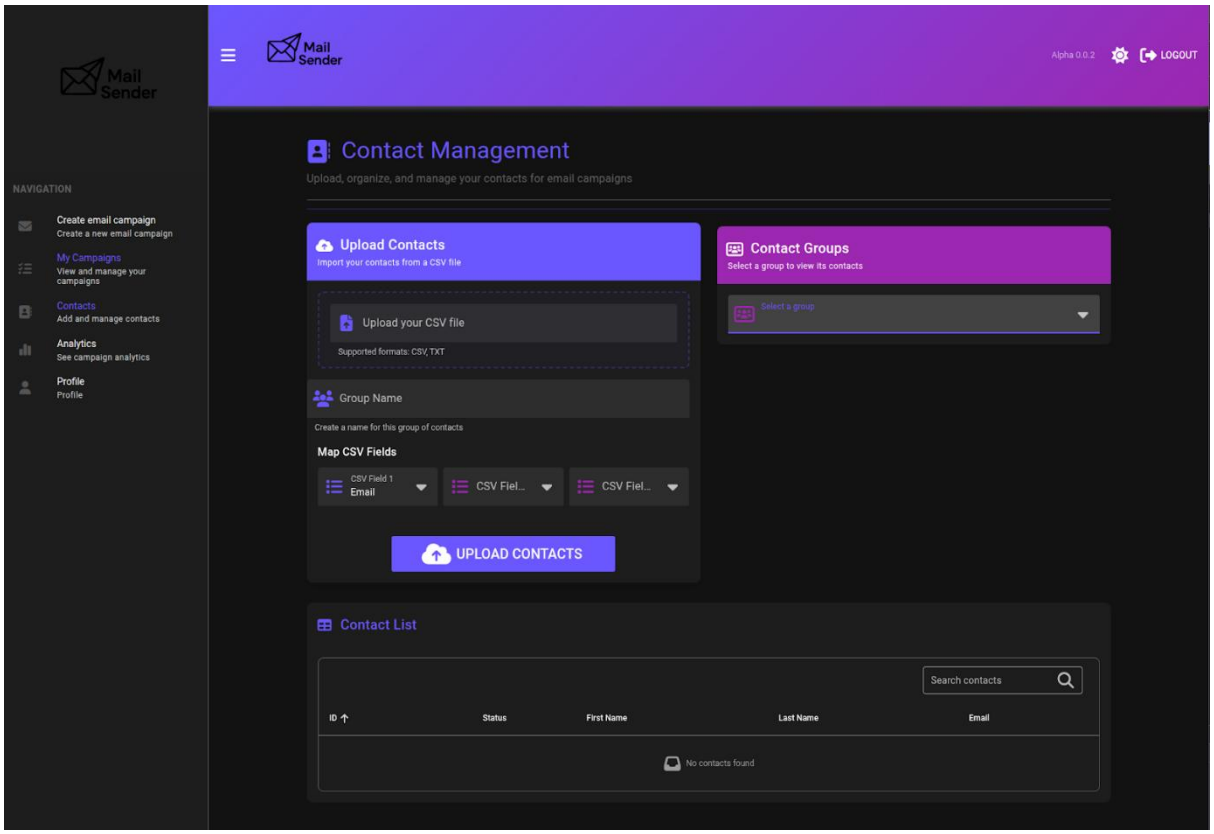
Πάνω δεξιά υπάρχει το κατάλληλο κουμπί που επιτρέπει στον χρήστη να αποσυνδεθεί από τον λογαριασμό του, καθώς και ένα κουμπί που του δίνει την δυνατότητα να επιλέξει ανάμεσα σε φωτεινό και σκοτεινό θέμα.



Εικόνα 13: Αρχική σελίδα εφαρμογής

7.4 Σελίδα διαχείρισης επαφών

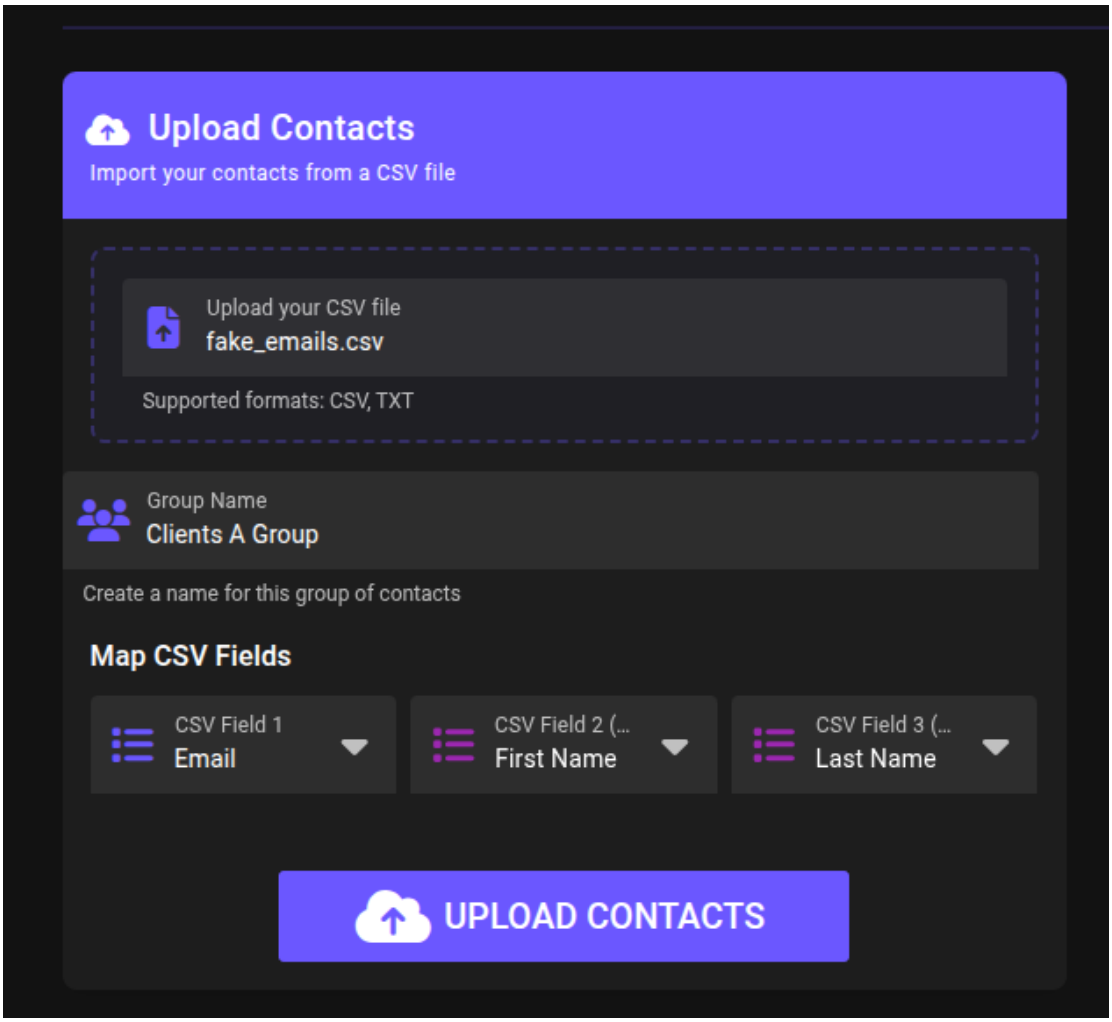
Η σελίδα διαχείρισης επαφών δίνει την δυνατότητα στον χρήστη να διαχειριστεί τις ομάδες επαφών του, καθώς και να εισάγει νέες επαφές και να δημιουργήσει νέες ομάδες. Το πρώτο βήμα που πρέπει να κάνει ένας νέος χρήστης της εφαρμογής είναι να εισάγει ένα σύνολο επαφών, ώστε στην συνέχεια να δημιουργήσει μια καμπάνια που να αφορά αυτές τις επαφές.



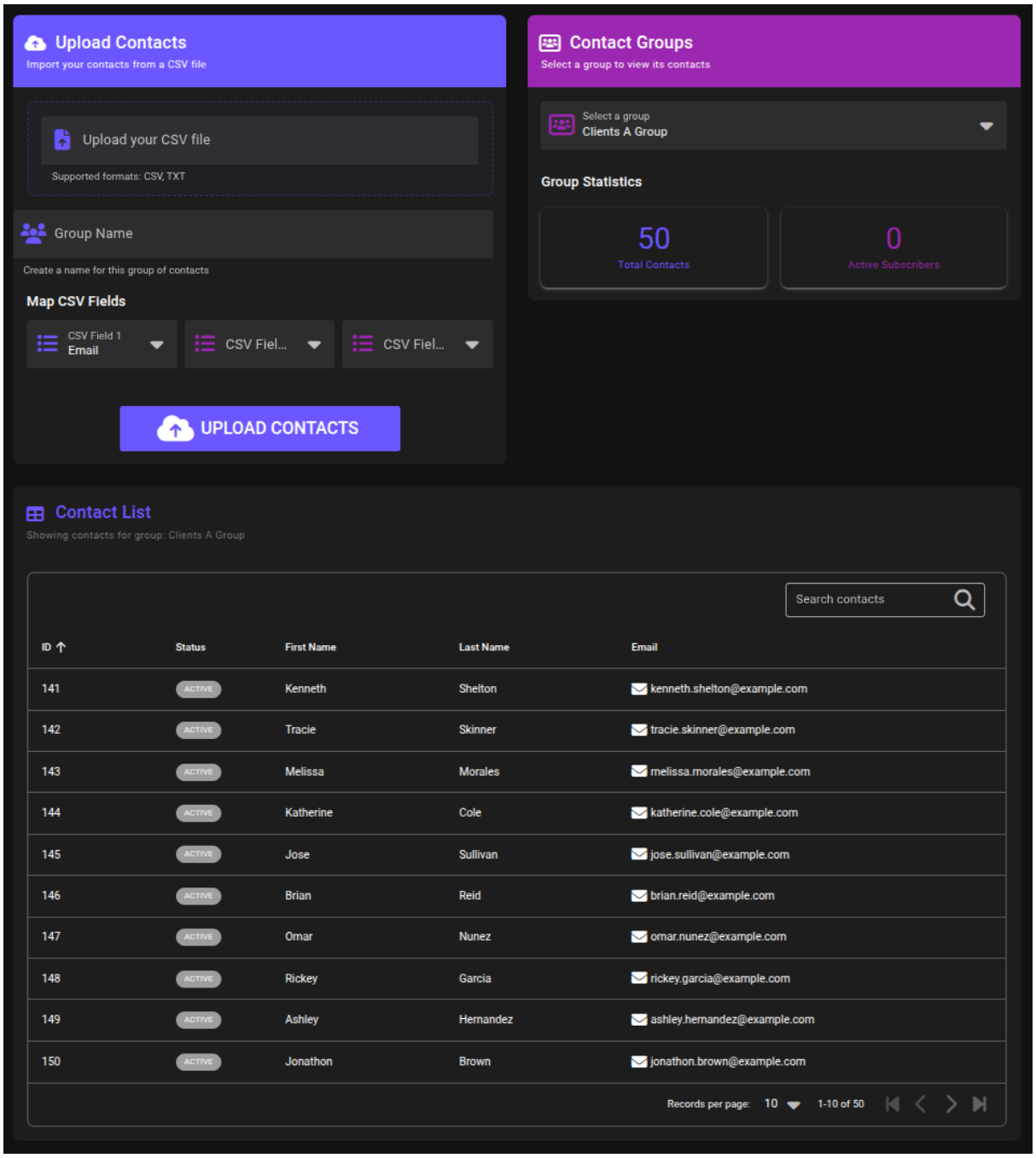
Εικόνα 14: Σελίδα εισαγωγής επαφών

Η εισαγωγή επαφών γίνεται με την διαδικασία drag and drop ενός αρχείου csv. Αυτό το csv αρχείο θα πρέπει να περιέχει σε κάθε γραμμή ενα email, και προαιρετικά ένα όνομα και ένα επώνυμο. Αφού ο χρήστης επιλέξει το csv αρχείο που θέλει, πρέπει στην συνέχεια να χρησιμοποιήσει το μενού *"Map CSV Fields*" ώστε να αντιστοιχίσει τα στοιχεία κάθε επαφής, στην κατάλληλη στήλη του CSV αρχείου. Τέλος, ο χρήστης καλείται να ορίσει ένα όνομα ομάδας (Group Name), μέσα στην οποία θα εισαχθούν οι επαφές.

Μετά την επιτυχή εισαγωγή επαφών ο χρήστης μπορεί να επιλέξει στο αριστερό μενού την ομάδα την οποία μόλις δημιούργησε, και να δει κάθε μεμονωμένη επαφή που εισήχθη σε αυτή την ομάδα.



Εικόνα 15: Εισαγωγή CSV με επαφές



Εικόνα 16: Επιλογή ομάδας

7.5 Σελίδα δημιουργίας καμπάνιας

Η σελίδα αυτή είναι ο πυρήνας της εφαρμογής. Είναι η σελίδα που επιτρέπει στον χρήστη να δημιουργήσει το περιεχόμενο ενός email και να το στείλει μαζικά στις επαφές του.

Η σελίδα δημιουργίας καμπάνιας καθοδηγεί τον χρήστη σε μια διαδικασία τεσσάρων βημάτων, όπου σε κάθε βήμα ο χρήστης ρυθμίζει διαφορετικές παραμέτρους της καμπάνιας.

Τα βήματα είναι τα εξής:

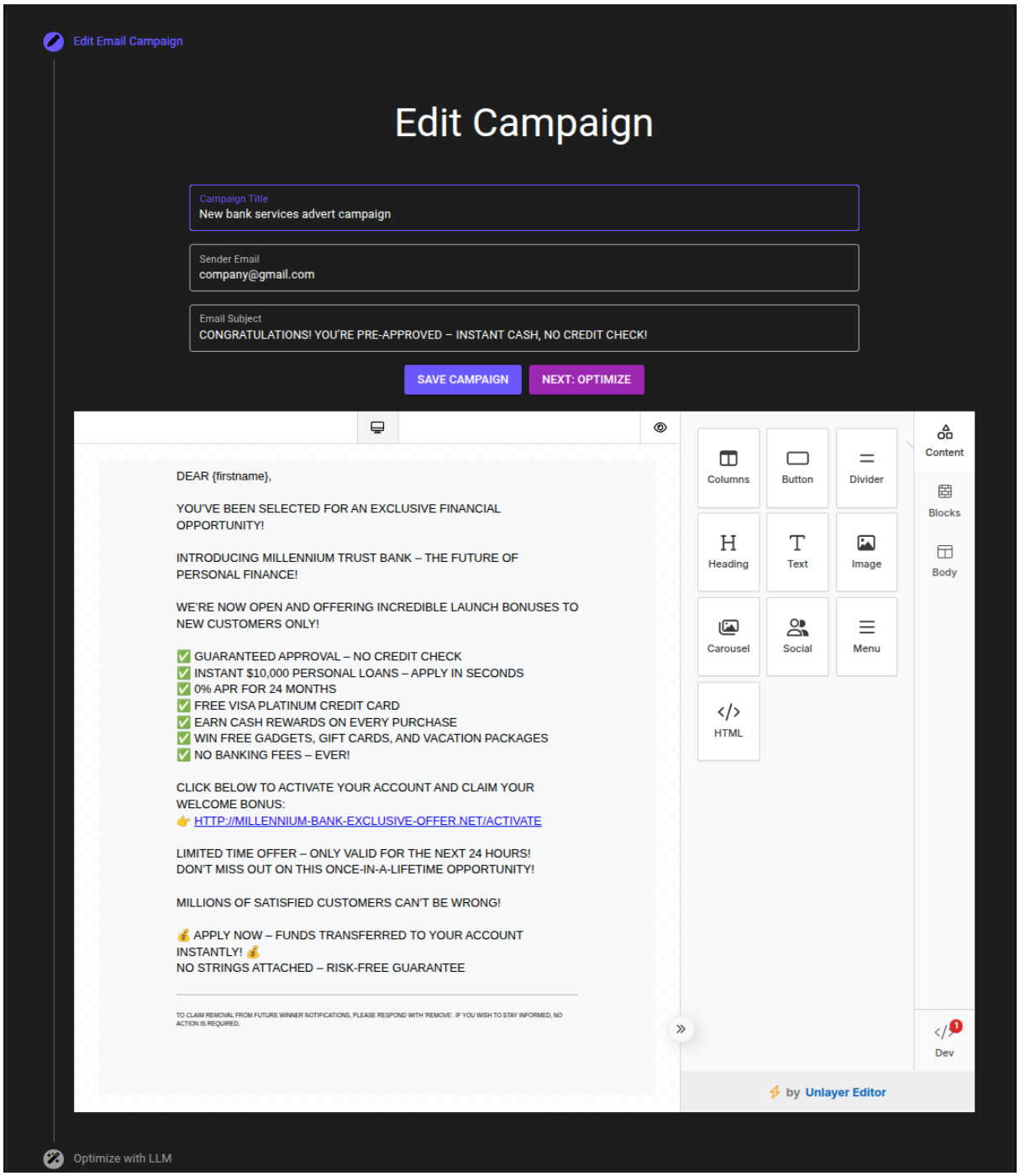
- *Create Email Campaign: Δημιουργία του περιεχομένου της καμπάνιας*
- *Optimize with LLM: Βελτιστοποίηση του περιεχομένου*
- *Select Recipients: Επιλογή των παραληπτών της καμπάνιας*
- *Schedule Campaign: Προγραμματισμός αποστολής καμπάνιας*

7.5.1 Δημιουργία περιεχομένου

Στο πρώτο βήμα ο χρήστης καλείται να θέσει ένα τίτλο για την καμπάνια, την διευθυνση email η οποία θα τεθεί ώς αποστολέας της καμπάνιας, καθώς και το θέμα του email. Στην συνέχεια μπορεί να δημιουργήσει το περιεχόμενο του email με την χρήση ενός visual editor.

Ο visual editor προσφέρει στον χρήστη την δυνατότητα να χτίσει εύκολα το email του, να χρησιμοποιήσει εικόνες κάνοντας τες drag and drop μέσα στο περιεχόμενο του email. Επίσης μπορεί να χρησιμοποιήσει το *${firstname}* και *${lastname}* ώστε να προσωποποιήσει το email σε κάθε ξεχωριστό παραλήπτη. Η εφαρμογή κατα την αποστολή θα αντικαταστήσει αυτά τα placeholders με το όνομα και επώνυμο κάθε παραλήπτη.

Ο χρήστης, οποιαδήποτε στιγμή επιθυμεί, μπορεί να αποθηκεύσει την πρόοδό του πατώντας το κουμπί "*Save campaign*". Η καμπάνια θα αποθηκευτεί ως "Draft campaign" και μπορεί να συνεχίσει την δημιουργία της άλλη στιγμή. Όταν τελειώσει με το πρώτο βήμα, ο χρήστης πρέπει να αποθηκεύσει ξανά την πρόοδο του, και στην συνεχεια να προχωρήσει στο επόμενο βήμα πατώντας το κουμπί "*NEXT: OPTIMIZE*".

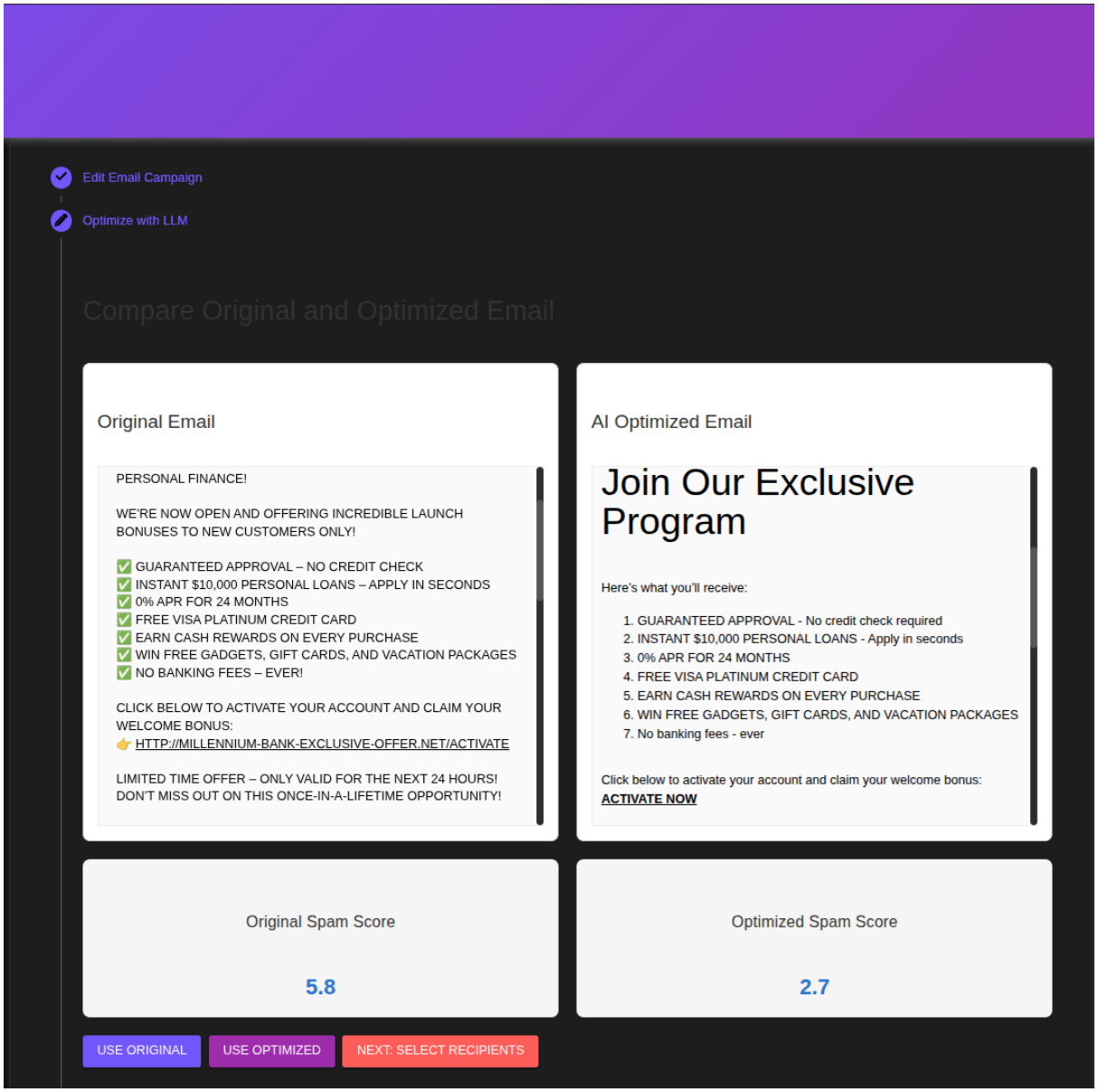


Εικόνα 17: Δημιουργία καμπάνιας (Δημιουργία email)

7.5.2 Βελτιστοποίηση περιεχομένου email

Στο δεύτερο βήμα της διαδικασίας δημιουργίας καμπάνιας, παρουσιάζεται στον χρήστη μια βελτιστοποιημένη έκδοση του email που δημιούργησε. Πιο συγκεκριμένα, στα αριστερά παρουσιάζεται το αρχικό email που δημιούργησε ο χρήστης, και στα δεξιά είναι η βελτιστοποιημένη έκδοση του email. Επίσης, ο χρήστης μπορεί να δει το Spam-score για το αρχικό email που δημιούργησε, καθώς και για το βελτιστοποιημένο email.

Έπειτα δίνεται η επιλογή να χρησιμοποιηθεί για την καμπάνια το αρχικό email ή να επιλεχθεί το βελτιστοποιημένο email. Αφού γίνει η επιλογή, μετά ο χρήστης προχωρά στο τρίτο βήμα, πατώντας το κουμπί *NEXT: SELECT RECIPIENTS*.

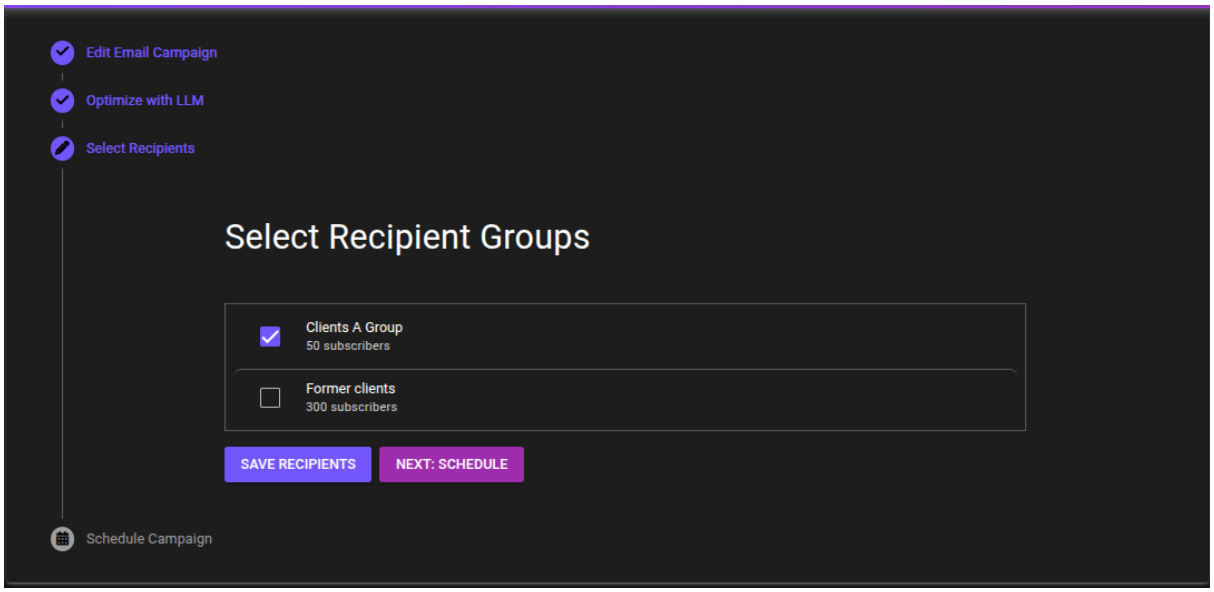


Εικόνα 18: Δημιουργία καμπάνιας (Βελτιστοποίηση email)

7.5.3 Επιλογή παραληπτών καμπάνιας

Στο τρίτο βήμα δημιουργίας καμπάνιας, ο χρήστης καλείται να επιλέξει όλες τις ομάδες συνδρομητών που θα παραλάβουν το email που θα αποσταλεί.

Όλες οι ομάδες συνδρομητών παρουσιάζονται στον χρήστη με μορφή λίστας, και για κάθε ομάδα ο χρήστης μπορεί να δει το όνομα της, και το πλήθος συνδρομητών. Αφού γίνει η επιλογή των ομάδων, στην συνέχεια ο χρήστης προχωρά στο τελευταίο βήμα της διαδικασίας, πατώντας το κουμπί *NEXT: SCHEDULE*.

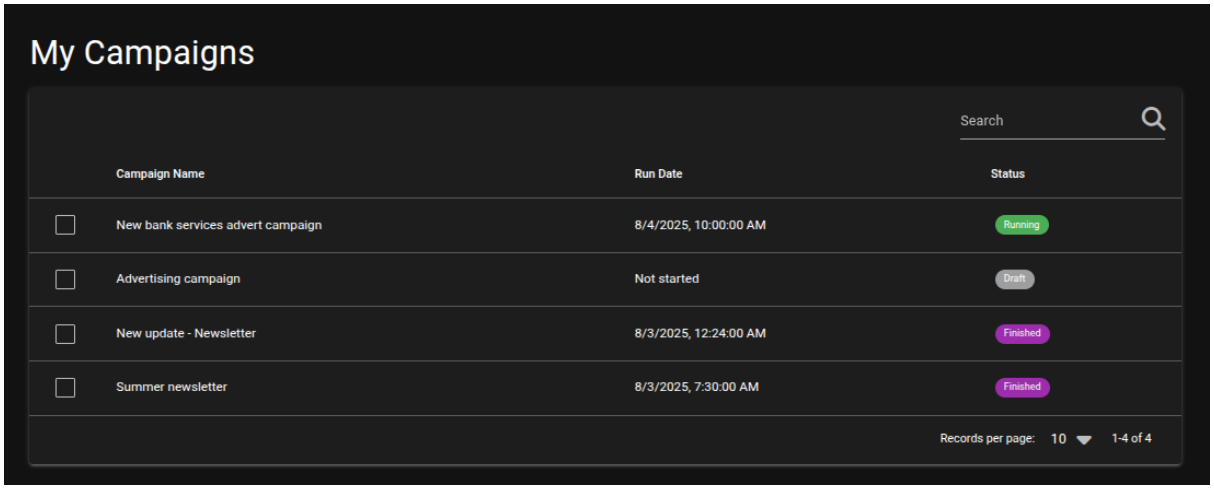


Εικόνα 19: Δημιουργία καμπάνιας (Προγραμματισμός καμπάνιας)

7.6 Σελίδα σύνοψης καμπάνιας

Για κάθε καμπάνια που έχει δημιουργηθεί, ο χρήστης έχει την δυνατότητα να δει μία σύνοψη της καμπάνιας, καθώς και στατιστικά που την αφορούν, όπως το πόσα email στάλθηκαν, πόσα από αυτά έφτασαν στον παραλήπτη, και πόσοι παραλήπτες άνοιξαν το email.

Για να δει την σύνοψη μιας καμπάνιας, ο χρήστης πρέπει πρώτα να επιλέξει από το μενού πλοήγησης την επιλογή *My Campaigns*. Εκεί θα του παρουσιαστεί μία λίστα από όλες τις καμπάνιες που έχει δημιουργήσει.



Εικόνα 20: Λίστα καμπανιών χρήστη

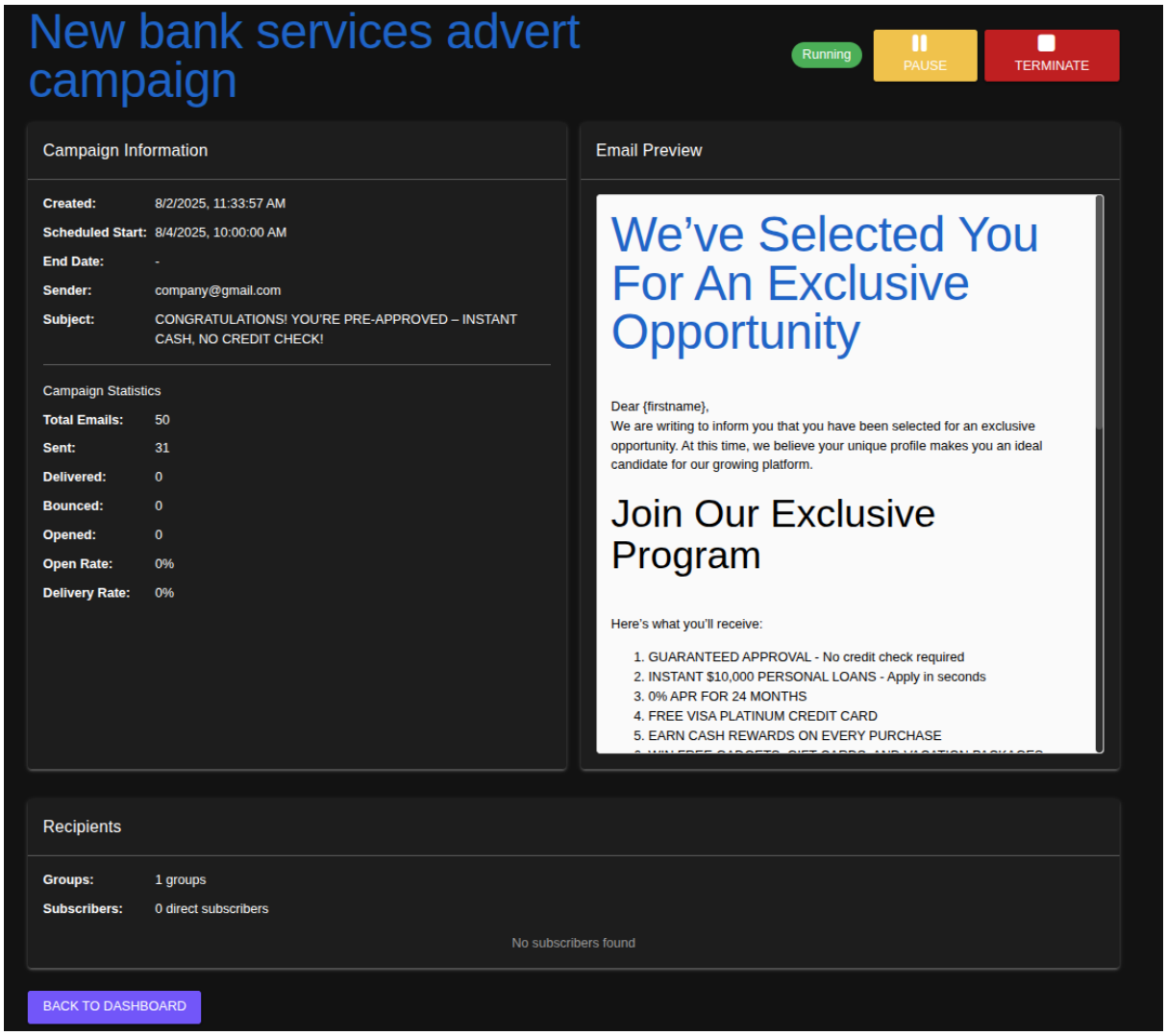
Στην συνέχεια, μπορεί να επιλέξει οποιαδήποτε καμπάνια που δεν είναι σε κατάσταση draft, και να πατήσει την επιλογή *VIEW DETAILS*. Αυτό θα τον μεταφέρει στην σελίδα σύνοψης της επιλεγμένης καμπάνιας.

Στην σελίδα σύνοψης, στα αριστερά παρουσιάζονται όλες οι πληροφορίες της καμπάνιας, όπως το πότε δημιουργήθηκε, πότε είναι προγραμματισμένη για αποστολή, πότε τελείωσε η αποστολή, το θέμα του email και η διεύθυνση αποστολής. Επίσης παρουσιάζονται τα εξής στατιστικά:

- *Συνολικά αριθμός συνδρομητών που αφορούσε η καμπάνια*
- *Πόσα email στάλθηκαν*
- *Πόσα email παραδόθηκαν επιτυχώς στους παραλήπτες τους*
- *Πόσα email απέτυχαν να παραδοθούν*
- *Πόσα email άνοιξαν από τον παραλήπτη τους*

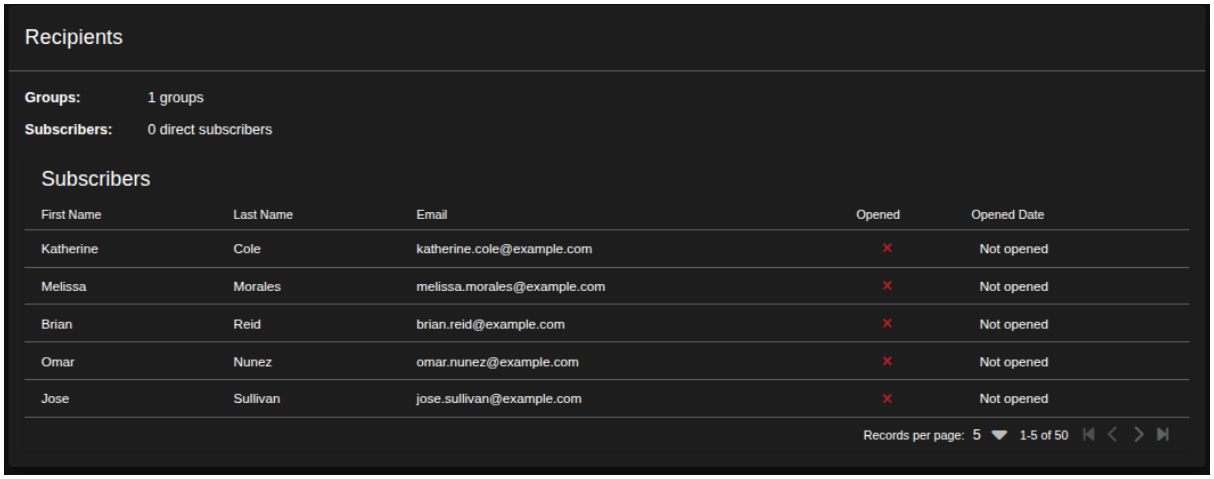
Στα δεξιά παρουσιάζεται το περιεχόμενο του email, όπως αυτό στάλθηκε στους συνδρομητές.

Σε περίπτωση που η επιλεγμένη καμπάνια δεν έχει ολοκληρωθεί αλλά είτε είναι σε εκτέλεση ή είναι προγραμματισμένη για αποστολή, τότε ο χρήστης έχει την δυνατότητα είτε να σταματήσει προσωρινά την καμπάνια χρησιμοποιώντας το κουμπί *PAUSE*, η να τερματίσει την εκτέλεση της με την χρήση του *TERMINATE*.



Εικόνα 21: Σύνοψη καμπάνιας σε εκτέλεση

Στην περίπτωση ολοκληρωμένης καμπάνιας, δίνεται και η δυνατότητα στον χρήστη να δει με λεπτομέρειες ποιοι συνδρομητές άνοιξαν και ποιοι όχι το email που στάλθηκε. Αυτή η λίστα με τους συνδρομητές βρίσκεται στο κάτω μέρος της σελίδας.



Εικόνα 22: Σύνοψη παραληπτών ολοκληρωμένης καμπάνιας

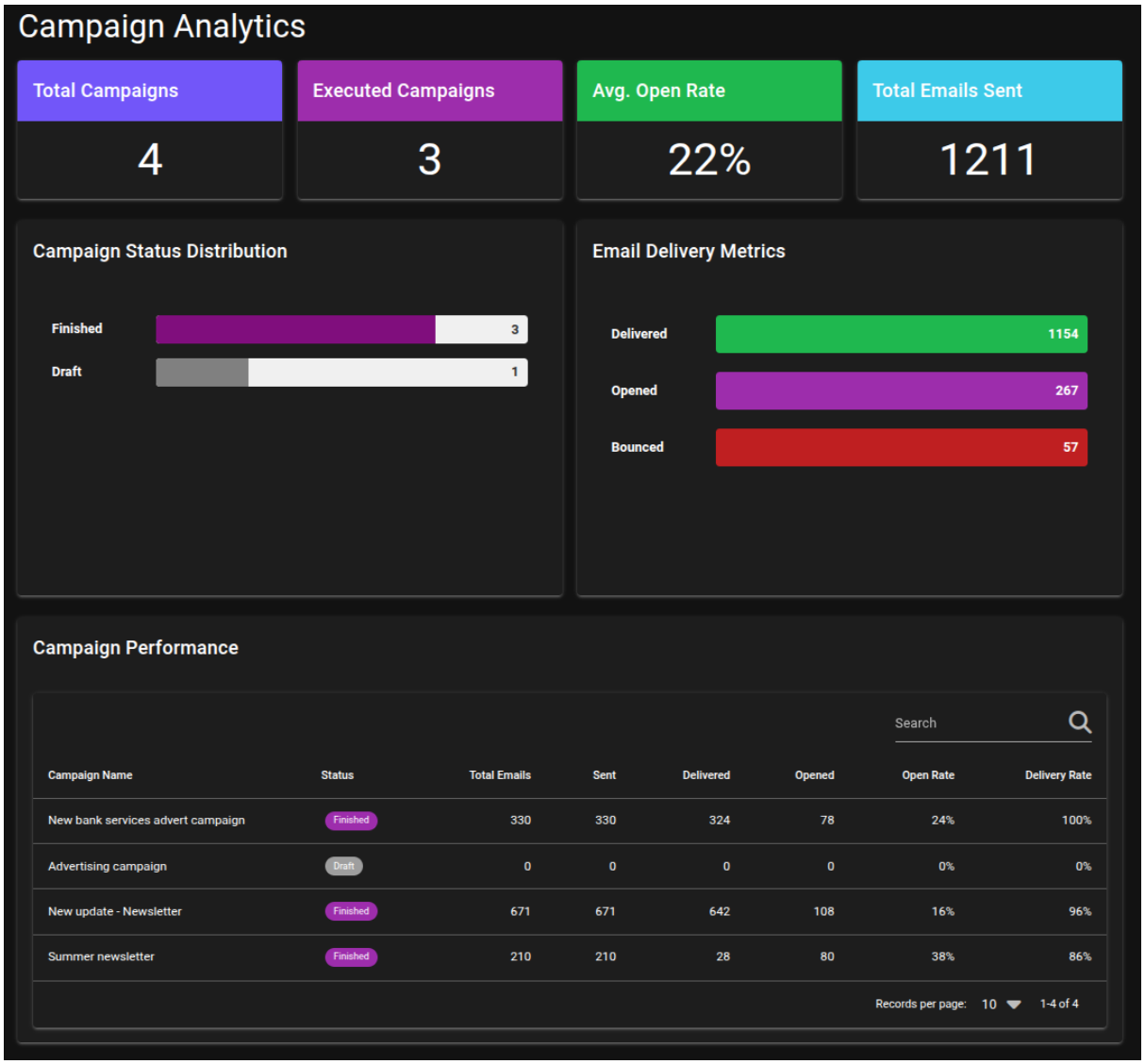
7.7 Σελίδα συνολικών στατιστικών

Ο σκοπός αυτής της σελίδας είναι να παρουσιάσει στον χρήστη τα συνολικά στατιστικά όλων των καμπανιών που έχει στείλει. Για να πλοηγηθεί σε αυτή την σελίδα, ο χρήστης πρέπει να επιλέξει το Analytics από το μενού πλοήγησης που βρίσκεται αριστερά.

Στο πάνω μέρος της σελίδας υπάρχει μια σύνοψη που παρουσιάζει 4 στοιχεία:

- *Το συνολικό αριθμό των καμπανιών που έχουν δημιουργηθεί*
- *Το συνολικό αριθμό των καμπανιών που έχουν αποσταλεί*
- *Το ποσοστιαίο μέσο όρο ανοίγματος των απεσταλμένων email*
- *Τον συνολικό αριθμό email που έχουν αποσταλεί*

Στο κάτω μέρος της σελίδας παρουσιάζεται μια λίστα με όλες τις καμπάνιες, και τα στατιστικά στοιχεία για κάθε καμπάνια ξεχωριστά, όπως τα συνολικά email κάθε καμπάνιας, το πόσα email ανοίχτηκαν και το πόσα απέτυχαν να αποσταλούν.



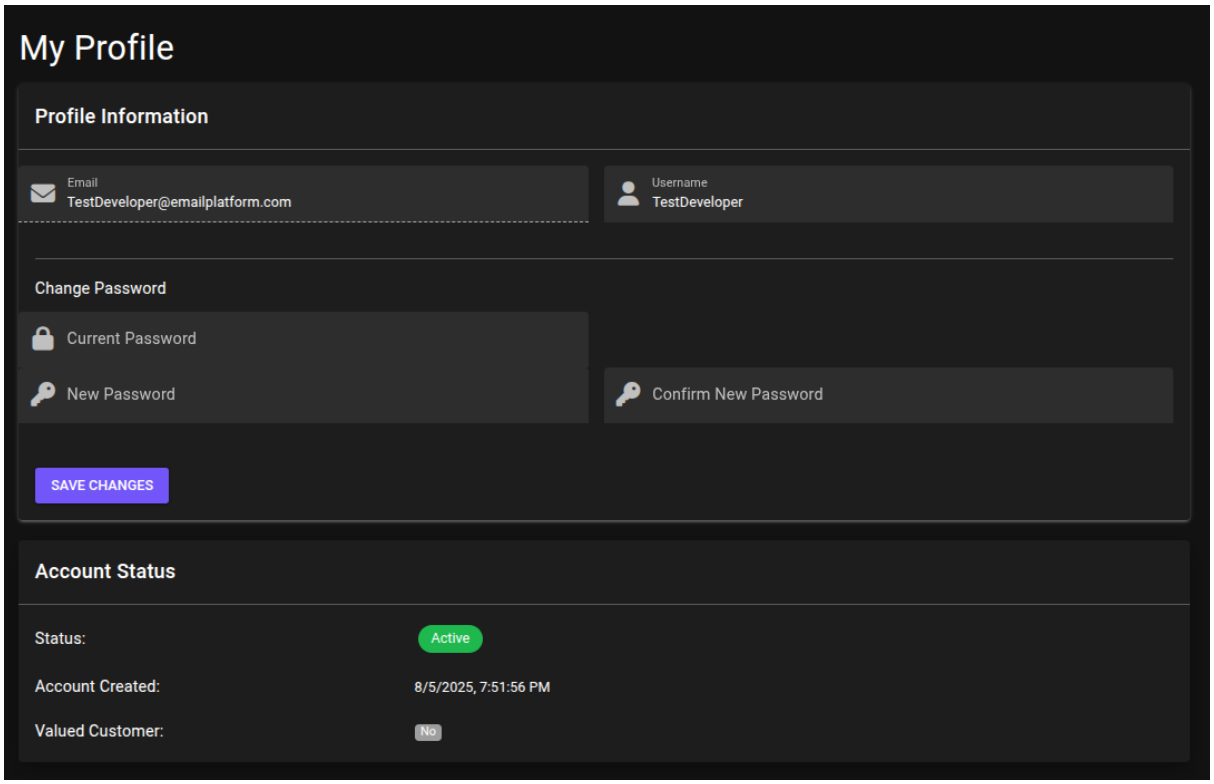
Εικόνα 23: Σύνοψη στατιστικών όλων των καμπανιών

7.8 Σελίδα προφίλ χρήστη

Ο χρήστης για να δει την σελίδα προφίλ πρέπει να χρησιμοποιήσει την επιλογη *Profile* στο μενού πλοήγησης. Αυτή η σελίδα αφορά την διαχείριση του λογαριασμού χρήστη.

Αρχικά, εκεί παρουσιάζονται διάφορα στοιχεία σχετικά με τον λογαριασμό χρήστη, όπως το email, το username, η ημερομηνία δημιουργίας του λογαριασμού καθώς και η κατάσταση του λογαριασμού. Εκτός από την παρουσίαση αυτών των στοιχείων, αυτή η σελίδα προσφέρει και την δυνατότητα στον χρήστη να αλλάξει τον κωδικό πρόσβασης του.

Για να γίνει αλλαγή κωδικού, ο χρήστης πρέπει να συμπληρώσει τον τρέχων κωδικό στο κατάλληλο πεδίο, και επίσης να συμπληρώσει τον νέο κωδικό. Στην συνέχεια με το κουμπί SAVE CHANGES μπορεί να οριστικοποιήσει την αλλαγή.



Εικόνα 24: Προφίλ χρήστη

Κεφάλαιο 8ο: Συμπεράσματα και προτάσεις βελτίωσης

8.1 Συμπεράσματα

Κατά την διάρκεια εκπόνησης της εργασίας εμφανίστηκαν πολλά τεχνικά προβλήματα τα οποία έπρεπε να βρεθούν λύσεις. Ένα από αυτά αποτελούσε ο αλγόριθμος που έπρεπε να φτιάξουμε για την επιλογή του κατάλληλου mail server. Η σωστή επιλογή ποιον παραμέτρων θα χρησιμοποιούταν αποτέλεσε ένα αρκετά μεγάλο πρόβλημα το οποίο χρειάστηκε να γίνει αρκετά μεγάλη ερευνά σε υπάρχουσες πλατφόρμες καθώς και μαζικά emails τα οποία εταιρίες στέλνουν ώστε να καταλήξουμε σε πιες τιμές θα ήταν «μεγάλες» και ποιες «ικανοποιητικές». Παράλληλα, λόγω της πολυπλοκότητας και του φόρτου που θα λάμβανε το σύστημα χρειάστηκε να γίνει πολύ μεγάλη εμβάθυνση στο κομμάτι του Spring Boot Web flux καθώς και στα coroutines που παρέχει η Kotlin με σκοπό την γρηγορότερη και παράλληλη επεξεργασία και αποστολή όλων των email.

Ένα συμπέρασμα επίσης που βγήκε από την εργασία αποτελεί ο εντοπισμός ορίων που παρατηρήθηκε με την χρήση του hibernate το οποίο είναι υπεύθυνο για την επικοινωνία του server με την βάση δεδομένων. Ενώ φαινομενικά αποτελεί ένα πολύ καλό εργαλείο που επιτρέπει στον προγραμματιστή να μην χρειαστεί να γράψει ερωτήματα SQL και να διαχειρίζεται την δημιουργία και ενημέρωση των πινάκων στην βάση δεδομένων, λόγω τις αυτόματης μετάφρασης από κώδικα ή annotations σε περιπλοκά ερωτήματα που υπήρχαν πολλά joins μεταξύ πινάκων παρατηρήθηκε ότι τα ερωτήματα πίσω στην βάση έπαιρναν πολύ περισσότερο χρόνο. Έπειτα από ανάλυση παρατηρήθηκε ότι το hibernate έχτιζε τα ερωτήματα με πολύ πιο περίπλοκο τρόπο από ότι χρειαζόταν. Παράλληλα με το γεγονός ότι είναι σχεδιασμένο ως blocking το παραπάνω πρόβλημα γίνεται πολύ πιο έντονο ειδικά σε περίπτωσης, όπως αυτής της εργασίας, που υπάρχει μεγάλη και συχνή επικοινωνία με τη βάση δεδομένων. Το πρόβλημα αυτό λύθηκε με την χρήση του R2DBC που δεν αποτελεί ORM όπως το hibernate και «αναγκάζει» τον προγραμματιστή να γράψει τα SQL ερωτήματα που αυτός θέλει.

8.2 Προτάσεις βελτίωσης

Υποστήριξη υπηρεσιών όπως Viber, WhatsApp και Messenger. Η εφαρμογή είναι σχεδιασμένη ώστε να μπορεί με σχετικά μεγάλη ευκολία να επεκταθεί περεταίρω, η ενσωμάτωση υποστήριξης κάποιας μηνυματικής υπηρεσίας θα μπορούσε να γίνει αρκετά ευκολά με την απλή προσθήκη ενός νέου Service, την δημιουργία μιας νέας ουράς στο RabbitMQ και την διαχωρίσει στον executor της ουράς αυτής ως μηνυματικής υπηρεσίας. Το τελευταίο πράγμα που θα πρέπει να γίνει είναι αντί μιας SMTP κλήσεις σε έναν mail server να γίνει μια HTTP κλήση ανάλογα με την τεκμηρίωση που παρέχει η εκάστοτε μηνυματική υπηρεσία. Ο λόγος που δεν υλοποιήθηκε είναι το κόστος που έρχεται με την έκδοση API κλειδιών (ειδικά για το Viber) και το γεγονός ότι αποφασίσαμε η πτυχιακή εργασία να έχει ως βάση αντικείμενα ανοιχτού κώδικα.

Ένας τελευταίος τρόπος βελτίωσης της εργασίας αποτελεί η αλλαγή και μετατροπή της αρχιτεκτονικής της εφαρμογής. Ξεκινώντας την εργασία αποφασίστηκε να ακολουθήσουμε την layered αρχιτεκτονική καθώς με αυτή είμασταν πολύ πιο εξοικειωμένοι. Μετά από ερευνά σχετικά με τις αρχιτεκτονικές εφαρμογών που υπάρχουν βάση των χαρακτηριστικών που θέλαμε να έχει η εφαρμογή (microservices, εύκολή επέκταση) εμφανίστηκε η αρχιτεκτονική εξάγωνου που αποτελούσε καλύτερη λύση. Λόγω του όγκου κώδικα που θα έπρεπε να μεταφερθεί και να αλλάξει καθώς και τις πολυπλοκότητας της, αποφασίστηκε να συνεχίσουμε με την layered αρχιτεκτονική. Πλέον καθώς υπάρχει μεγαλύτερη

εξοικείωση με την αρχιτεκτονική εξάγωνου υπάρχει σκοπός να ξανά γραφτεί ο κώδικας με στόχο την μετατροπή του σε αυτή την αρχιτεκτονική ακόμα και μετρά το πέρας της εργασίας.

Τέλος, η μεγαλύτερη αλλαγή και πιθανός βελτιώσει θα μπορούσε να είναι η μετατροπή του object oriented και annotation based κώδικα σε functional. Με την υποστηρίξει πλέον του Spring Boot με το functional endpoints, Bean Definition DSL καθώς και την υποστήριξη της Kotlin, μπορεί η εργασία να μετατραπεί σε functional. Αυτό θα μπορούσε σε συνδυασμό με την μετατροπή της εφαρμογής από layered σε αρχιτεκτονική εξάγωνου να βοηθήσει στην γενίκευση κάποιων σημείων του κώδικα και να βοηθήσει στο ποσό ευκολά μπορεί να τηρηθεί το SOLID και πιο συγκεκριμένα το O το οποίο είναι η open-close αρχή. Η συγκεκριμένη αλλαγή αποτελεί μια από τις πιο δύσκολες που θα μπορούσαν να γίνουν καθώς προϋποθέτει πολύ μεγάλη εξοικείωση και κατανόηση του πως δουλεύει ο functional προγραμματισμός.

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] “Kotlin docs” Accessed: Aug 19, 2025. [Online]. Available: https://kotlinlang.org/docs/home.html

[2] “Kotlin Functional (SAM) interfaces” Accessed: Aug 19, 2025. [Online]. Available: https://kotlinlang.org/docs/fun-interfaces.html

[3] “Kotlin Coroutines”, Accessed: Aug 19, 2025. [Online]. Available: https://kotlinlang.org/docs/coroutines-overview.html

[4] R. Jones, “Object-Oriented Programming LNCS 8586 ARCoSS.”

[5] Alejandro Serrano Mena, “Functional programming Ideas for the Curious Kotliner”

[6] ”InnoDB and the ACID Model” Accessed: Aug 19, 2025. [Online]. Available: https://dev.mysql.com/doc/refman/8.4/en/mysql-acid.html

[7] “MySQL Documentation” Accessed: Aug 19, 2025. [Online]. Available: https://dev.mysql.com/doc/

[8] ”PostgreSQL 17.6 Documentation” Accessed: Aug 19, 2025. [Online]. Available: https://www.postgresql.org/docs/current/index.html

[9] ”Microsoft SQL Server” Accessed: Aug 19, 2025. [Online]. Available: https://learn.microsoft.com/en-us/sql/sql-server/?view=sql-server-ver17

[10] ”Oracle Database Documentation” Accessed: Aug 19, 2025. [Online]. Available: https://docs.oracle.com/en/database/oracle/oracle-database/

[11] “MariaDB Documentation” Accessed: Aug 19, 2025. [Online]. Available: https://mariadb.com/docs

[12] ”Apache Ant 1.10.15 Manual” Accessed: Aug 19, 2025. [Online]. Available: https://ant.apache.org/manual/index.html

[13] ”Apache Maven Project Documentation” Accessed: Aug 19, 2025. [Online]. Available: https://maven.apache.org/guides/index.html

[14] ”Gradle User Manual” Accessed: Aug 19, 2025. [Online]. Available: https://docs.gradle.org/current/userguide/userguide.html

[15] “What’s the Difference Between Kafka and RabbitMQ? ” Accessed: Aug 19, 2025. [Online]. Available: https://aws.amazon.com/compare/the-difference-between-rabbitmq-and-kafka/

[16] “Comparing Version Control Systems: Git, SVN, CVS, and More” Accessed: Aug 19, 2025. [Online]. Available: https://medium.com/@chittaranjansethi/comparing-version-control-systems-git-svn-cvs-and-more-6af27a74685d

[17] “git Documentation” Accessed: Aug 19, 2025. [Online]. Available: https://git-scm.com/doc

[18] “Apache Subversion Manual” Accessed: Aug 19, 2025. [Online]. Available: https://subversion.apache.org/docs/

[19] ”GitLab Runner Documentation” Accessed: Aug 19, 2025. [Online]. Available: https://docs.gitlab.com/runner/

[20] “GitHub Actions Documentation” Accessed: Aug 19, 2025. [Online]. Available: https://docs.github.com/en/actions

[21] Mark Richards, “Software Architecure Patterms”

[22] ”The Clean Architecture by Unclee Bob” Accessed: Aug 19, 2025. [Online]. Available: https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html

[23] ”Apache SpamAssassin Documentation” Accessed: Aug 19, 2025. [Online]. Available: https://spamassassin.apache.org/doc.html

[24] “React Documentation” Accessed: Aug 19, 2025. [Online]. Available: https://react.dev/learn

[25] ”Angular Documentation” Accessed: Aug 19, 2025. [Online]. Available: https://angular.dev/overview

[26] ”Vue.js Documentation” Accessed: Aug 19, 2025. [Online]. Available: https://vuejs.org/guide/introduction

[27] “ReactJs Functional Components” Accessed: Aug 19, 2025. [Online]. Available: https://www.geeksforgeeks.org/reactjs/reactjs-functional-components/

[28] “Quasar API” Accessed: Aug 19, 2025. [Online]. Available: https://quasar.dev/api-explorer

[29] ”Stateful and stateless authentication” Accessed: Aug 19, 2025. [Online]. Available: https://medium.com/@kennch/stateful-and-stateless-authentication-10aa3e3d4986

[30]| ”JSON Web Token (JWT) RFC7519” Accessed: Aug 19, 2025. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc7519

[31] “WebFlux Security SpringBoot” Accessed: Aug 19, 2025. [Online]. Available: https://docs.spring.io/spring-security/reference/reactive/configuration/webflux.html

[32] “JSON Web Token (JWT) IANA Registry” Accessed: Aug 19, 2025. [Online]. Available: https://www.iana.org/assignments/jwt/jwt.xhtml

[33] “Liquibase documentation” Accessed: Aug 19, 2025. [Online]. Available: https://docs.liquibase.com/home.html

[34] “What is REST API? IBM” Accessed: Aug 19, 2025. [Online]. Available: https://www.ibm.com/think/topics/rest-apis

[35] “SpringAI Documentation” Accessed: Aug 19, 2025. [Online]. Available: https://docs.spring.io/spring-ai/reference/index.html

[36] “OpenAI Model Spec” Accessed: Aug 19, 2025. [Online]. Available: https://model-spec.openai.com/2025-04-11.html

[37] “Vue email editor, open source project” Accessed: Aug 19, 2025. [Online]. Available: https://github.com/unlayer/vue-email-editor

[38] “Axios Documentation” Accessed: Aug 19, 2025. [Online]. Available: https://axios-http.com/docs/intro

[39] “Docker Docs” Accessed: Aug 19, 2025. [Online]. Available: https://docs.docker.com/

[40] “Kubernetes Documentation” Accessed: Aug 19, 2025. [Online]. Available: https://kubernetes.io/docs/home/

[41] “RabbitMQ Installation” Accessed: Aug 19, 2025. [Online]. Available: https://www.rabbitmq.com/docs/download

[42] “MinIO Quickstart Guide” Accessed: Aug 19, 2025. [Online]. Available: https://charts.min.io/

[43] “Getting Started with Amazon S3” Accessed: Aug 19, 2025. [Online]. Available: https://aws.amazon.com/s3/getting-started/

[44] “Install Docker Engine” Accessed: Aug 19, 2025. [Online]. Available: https://docs.docker.com/engine/install/

[45] “Windows Subsystem for Linux Documentation” Accessed: Aug 19, 2025. [Online]. Available: https://learn.microsoft.com/en-us/windows/wsl/