ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

Σχολή Τεχνολογίας

Τμήμα Ψηφιακών Συστημάτων

Π.Μ.Σ. Μηχανική Λογισμικού

για Διαδικτυακές & Φορητές Εφαρμογές

Ανάπτυξη Εφαρμογής για την Προβολή Ιστορικών Δεδομένων Διαστημικών Αποστολών

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Μιχελής Νικόλαος (ΑΜ: Μ013123013)

Επιβλέπων: Γεώργιος Κακαρόντζας, Καθηγητής

ΛΑΡΙΣΑ, Μάϊος 2025

UNIVERSITY OF THESSALY

School of Technology

Digital Systems Department

MSc in Software Engineering

for Internet & Mobile Applications

Development of an Application for Viewing Historical Space Mission Data

MASTER THESIS

Nikolaos Michelis (RN:M013123013)

Supervisor: George Kakarontzas, Professor

LARISSA, May 2025

Υπεύθυνη Δήλωση περί Ακαδημαϊκής Δεοντολογίας και Πνευματικών Δικαιωμάτων

Με πλήρη επίγνωση των συνεπειών του νόμου περί πνευματικών δικαιωμάτων, δηλώνω ρητά ότι η παρούσα πτυχιακή εργασία, καθώς και τα ηλεκτρονικά αρχεία και πηγαίοι κώδικες που αναπτύχθηκαν ή τροποποιήθηκαν στα πλαίσια αυτής της εργασίας, αποτελεί αποκλειστικά προϊόν προσωπικής μου εργασίας, δεν προσβάλλει κάθε μορφής δικαιώματα διανοητικής ιδιοκτησίας, προσωπικότητας και προσωπικών δεδομένων τρίτων, δεν περιέχει έργα/εισφορές τρίτων για τα οποία απαιτείται άδεια των δημιουργών/δικαιούχων και δεν είναι προϊόν μερικής ή ολικής αντιγραφής, οι πηγές δε που χρησιμοποιήθηκαν περιορίζονται στις βιβλιογραφικές αναφορές και μόνον, και πληρούν τους κανόνες της επιστημονικής παράθεσης. Τα σημεία όπου έχω χρησιμοποιήσει ιδέες, κείμενο, αρχεία ή/και πηγές άλλων συγγραφέων, αναφέρονται ευδιάκριτα στο κείμενο με την κατάλληλη παραπομπή και η σχετική αναφορά περιλαμβάνεται στο τμήμα των βιβλιογραφικών αναφορών με πλήρη περιγραφή.

Αναλαμβάνω πλήρως, ατομικά και προσωπικά, όλες τις νομικές και διοικητικές συνέπειες που δύναται να προκύψουν στην περίπτωση κατά την οποία αποδειχθεί, διαχρονικά, ότι η εργασία αυτή ή τμήμα της δεν μου ανήκει διότι είναι προϊόν λογοκλοπής.

Ο/Η Δηλών/ούσα

(Υπογραφή)

Νικόλαος Μιχελής

20/05/2025

Εγκρίνεται από την Επιτροπή Εξέτασης:

Επιβλέπων/πουσα	**Γεώργιος Κακαρόντζας** Καθηγητής, Τμήμα Ψηφιακών Συστημάτων, Πανεπιστήμιο Θεσσαλίας
Μέλος	**Ονοματεπώνυμο Μέλους 1** Βαθμίδα/ιδιότητα μέλους 1, Τμήμα/Ίδρυμα μέλους 1
Μέλος	**Ονοματεπώνυμο Μέλους 2** Βαθμίδα/ιδιότητα μέλους 2, Τμήμα/Ίδρυμα μέλους 2

Ημερομηνία έγκρισης: 16-6-2025

Περίληψη

Το αντικείμενο της παρούσας μεταπτυχιακής εργασίας αφορά την ανάπτυξη μιας διαδικτυακής εφαρμογής. Σε πρώτη φάση πραγματοποιήθηκε η ανάλυση των απαιτήσεων της εφαρμογής. Έπειτα, σε δεύτερη φάση πραγματοποιήθηκε η σχεδίαση της εφαρμογής και η υλοποίηση του πρωτότυπου της. Στη συνέχεια, σε τρίτη φάση πραγματοποιήθηκε η υλοποίηση του λογισμικού χρησιμοποιώντας τις τεχνολογίες Spring Boot για το backend και React, SCSS για το frontend. Επίσης, το λογισμικό σχεδιάστηκε με το design pattern Model-View-Controller (MVC) όσον αφορά το backend και η υλοποίηση του frontend με τη μορφή ενός component based Single- Page-Application (SPA). Τέλος, αφού ολοκληρώθηκαν όλα τα παραπάνω πραγματοποιήθηκε η εγκατάσταση της εφαρμογής στη cloud πλατφόρμα Amazon Web Services.

Abstract

The subject of this master's thesis concerns the development of a web application. In the first phase, the requirements of the application were analyzed. In the second phase, the application design was created and the prototype was implemented. Subsequently, in the third phase, the software was developed using the Spring Boot technologies for the backend and React, SCSS for the frontend. Additionally, the software was designed using the Model-View-Controller (MVC) design pattern for the backend, and the frontend was implemented as a component-based Single-Page Application (SPA). Finally, after completing all the above steps, the application was deployed on the cloud platform Amazon Web Services

Ευχαριστίες

Με την ολοκλήρωση της παρούσας εργασίας, φτάνω στο τέλος του μεταπτυχιακού μου και κλείνει έτσι ένας σημαντικός κύκλος σπουδών και προσωπικής εξέλιξης. Ένα μεγάλο ευχαριστώ στον επιβλέποντα καθηγητή κύριο Κακαρόντζα Γεώργιο για την καθοδήγηση και την πολύτιμη βοήθεια που μου πρόσφερε κατά τη διάρκεια της μεταπτυχιακής εργασίας, ώστε να επιτύχω το καλύτερο δυνατό αποτέλεσμα. Στη συνέχεια, θα ήθελα να ευχαριστήσω τους The Space Devs για την παροχή των δεδομένων αφού η απόφαση τους για την παροχή των δεδομένων τους δημόσια έκανε δυνατή την υλοποίηση αυτής της εργασίας. Τέλος, θα ήθελα να ευχαριστήσω την οικογένεια μου για τη στήριξη και τη βοήθειά τους.

Μιχελής Νικόλαος

ημερομηνία

Περιεχόμενα

1 Εισαγωγή

Σε αυτή την ενότητα παρουσιάζεται το αντικείμενο της μεταπτυχιακής εργασίας, η παρούσα κατάσταση και τα δομικά της στοιχεία.

1.1 Αντικείμενο Εργασίας

Η ραγδαία τεχνολογική εξέλιξη των τελευταίων ετών, έχει οδηγήσει σε αύξηση του αριθμού των διαστημικών εκτοξεύσεων που πραγματοποιούνται ετησίως. Ο αυξανόμενος ρυθμός των αποστολών, έχει ως αποτέλεσμα την πραγματοποίηση τους σε εβδομαδιαία βάση, γεγονός που οδηγεί σε σημαντική αύξηση του όγκου των παραγόμενων δεδομένων. Σε αυτό το πλαίσιο, η ανάπτυξη μιας εφαρμογής που παρακολουθεί σε σχεδόν πραγματικό χρόνο πότε θα πραγματοποιηθεί μια διαστημική αποστολή αποκτά ιδιαίτερη σημασία.

Η παρούσα εργασία επικεντρώνεται στην ανάπτυξη μίας διαδικτυακής εφαρμογής για την προβολή δεδομένων διαστημικών αποστολών. Η συγκεκριμένη εφαρμογή αποτελεί λογισμικό εκπαιδευτικού σκοπού το οποίο εκμεταλλεύεται εξωτερικές πηγές δεδομένων κάνοντας χρήση κατάλληλου (API). Επιπλέον, η εφαρμογή έχει σχεδιαστεί έτσι ώστε να παρέχει μια φιλική προς τον χρήστη διεπαφή, εύκολη πλοήγηση και εργαλεία για αναζήτηση εξατομικευμένου περιεχομένου.

Αναλυτικότερα, με την εφαρμογή που αναπτύχθηκε, οι χρήστες έχουν τη δυνατότητα να παρακολουθούν σε πραγματικό χρόνο πότε θα γίνει η επόμενη εκτόξευση ή να αναζητούν παλαιότερες μέσω των διαθέσιμων φίλτρων. Επιπλέον υπάρχουν αρκετές ενότητες σχετικών δεδομένων όπως πληροφορίες για αστροναύτες, πληροφορίες για τη δομή ενός πυραύλου δηλαδή από ποιο προωθητήρα και διαστημόπλοιο αποτελείται και πληροφορίες για την τοποθεσία της εκτόξευσης αλλά και σχετικές ειδήσεις. Τέλος, οι χρήστες μπορούν να αποθηκεύουν τις εκτοξεύσεις που τους ενδιαφέρουν και να διαβάσουν το αντίστοιχο άρθρο αργότερα.

1.2 Υφιστάμενη Κατάσταση

Υπάρχουν ήδη εφαρμογές οι οποίες μπορούν να ενημερώσουν τους χρήστες σχετικά με τις διαστημικές αποστολές όπως το Next Space Flight [10], Space Launch Schedule [11] αλλά και οι σελίδες των επίσημων φορέων όπως της NASA [12], ESA [13], SpaceX [14]. Το μειονέκτημα των μικρότερων εφαρμογών είναι ότι δεν προσφέρουν εξατομικευμένο περιεχόμενο στους χρήστες και η αναζήτηση των πληροφοριών είναι δύσχρηστος. Επίσης, οι εφαρμογές των επίσημων φορέων παρέχουν μεγάλο όγκο πληροφοριών και πιθανότατα κάποιος μέσος χρήστης να χάσει το ενδιαφέρον του. Ο βασικός στόχος της εφαρμογής, είναι να προσφέρει σε όλους τους χρήστες μια ευχάριστη εμπειρία ανάγνωσης, χωρίς να τον βομβαρδίζει ταυτόχρονα με πολλές πληροφορίες. Τέλος, θα έχει τη δυνατότητα να αποθηκεύει άρθρα στην ψηφιακή του βιβλιοθήκη έτσι ώστε να τα διαβάσει αργότερα, πράγμα που δεν έχουν άλλες παρόμοιες εφαρμογές.

1.3 Δομή Εργασίας

Η παρούσα μεταπτυχιακή πτυχιακή εργασία χωρίζεται σε οκτώ θεματικές ενότητες. Συγκεκριμένα:

Στην 1η ενότητα γίνεται μια σύντομη παρουσίαση του θέματος της εργασίας, παρουσιάζονται οι λόγοι που οδήγησαν στην υλοποίηση της εφαρμογής.

Στην 2η ενότητα αναλύονται οι απαιτήσεις της εφαρμογής, παρουσιάζεται το διάγραμμα περιπτώσεων χρήσης και η αρχιτεκτονική τριών επιπέδων.

Στην 3η ενότητα παρουσιάζεται η υλοποίηση του frontend. Αρχικά αναλύεται η αρχιτεκτονική του και παρουσιάζεται το πρωτότυπο της γραφικής διασύνδεσης. Επιπλέον, παρουσιάζονται οι βιβλιοθήκες τρίτων μερών που αποτέλεσαν πιλοτικό κομμάτι της υλοποίησης.

Στην 4η ενότητα παρουσιάζεται η υλοποίηση του backend. Αρχικά αναλύεται η αρχιτεκτονική του και παρουσιάζεται το διάγραμμα ακολουθίας. Έπειτα, αναλύονται οι μηχανισμοί και οι βιβλιοθήκες που χρησιμοποιήθηκαν. Επιπλέον, αναλύεται ο μηχανισμός για την εξαγωγή, τον μετασχηματισμό και τη φόρτωση των δεδομένων στη βάση δεδομένων.

Στην 5ο ενότητα παρουσιάζεται η υλοποίηση του μηχανισμού αυθεντικοποίησης χρήστη. Αρχικά παρουσιάζεται το διάγραμμα της αρχιτεκτονικής του. Στην συνέχεια αναλύεται η υλοποίηση τόσο από πλευρά του frontend όσο και από την πλευρά του backend.

Στην 6ο ενότητα παρουσιάζεται η διαδικασία της εγκατάστασης της εφαρμογής στη cloud πλατφόρμα Amazon Web Services. Αρχικά παρουσιάζεται η αρχιτεκτονική του frontend στο cloud. Έπειτα, παρουσιάζεται η αρχιτεκτονική του backend στο cloud.

Στη Βιβλιογραφία παρουσιάζονται οι βιβλιογραφικές αναφορές.

Στα Παραρτήματα παρουσιάζονται κάποια βασικά κομμάτια κώδικα τόσο του backend όσο και του frontend

2 Απαιτήσεις Εφαρμογής

Σε αυτή την ενότητα αναλύονται οι λειτουργικές, μη λειτουργικές απαιτήσεις αλλά και οι τεχνικές απαιτήσεις. Επίσης, περιγράφονται οι μελλοντικές επεκτάσεις, αναλύονται βασικές περιπτώσεις χρήσης και περιγράφεται η αρχιτεκτονική τριών επιπέδων.

2.1 Λειτουργικές Απαιτήσεις

Στο Πίνακας 1 παρουσιάζονται οι βασικές απαιτήσεις της εφαρμογής. Αυτές οι λειτουργίες αφορούν όλους τους χρήστες και δεν απαιτούν αυθεντικοποίηση (authorization) ή κάποια εξειδικευμένη εξουσιοδότηση (authorization) για την πρόσβαση στα δεδομένα.

Πίνακας 1: Λειτουργικές απαιτήσεις.

Λ/Α	Λειτουργική Απαίτηση	Περιγραφή	Προτεραιότητα
ΛΑ-01	Αναζήτηση εκτοξεύσεων, αστροναυτών, πυραύλων, διαστημόπλοιων και προωθητήρων με φίλτρα.	Ο χρήστης μπορεί να εφαρμόζει φίλτρα (ημερομηνία, χώρα, οργανισμό, barcode) για εξατομικευμένα αποτελέσματα.	Υψηλή
ΛΑ-02	Προβολή λεπτομερειών εκτοξεύσεων, αστροναυτών, πυραύλων, διαστημόπλοιων και προωθητήρων.	Ο χρήστης μπορεί να δει αναλυτικές πληροφορίες για κάθε εκτόξευση, αστροναύτη, πρόγραμμα, πύραυλο, διαστημόπλοιο και προωθητήρα.	Υψηλή
ΛΑ-03	Προβολή των εκτοξεύσεων που έχει χρησιμοποιηθεί ο κάθε πύραυλος, προωθητήρας και διαστημόπλοιο.	Ο χρήστης μπορεί να δει όλες τις εκτοξεύσεις για κάθε πύραυλο, προωθητήρα και διαστημόπλοιο.	Υψηλή
ΛΑ-04	Αναζήτηση διαστημικών σταθμών με φίλτρα.	Ο χρήστης μπορεί να εφαρμόζει φίλτρα για εξατομικευμένα αποτελέσματα	Μέση
ΛΑ-05	Προβολή λεπτομερειών διαστημικών σταθμών.	Ο χρήστης μπορεί να δει αναλυτικές πληροφορίες για κάθε διαστημικό σταθμό	Μέση
ΛΑ-06	Προβολή σε χάρτη όλων των πλατφορμών εκτοξεύσεων.	Ο χρήστης μπορεί να δει σε ένα χάρτη τις ενεργές και μη ενεργές πλατφόρμες εκτόξευσης	Υψηλή
ΛΑ-07	Προβολή λεπτομερειών των πλατφορμών εκτοξεύσεων.	Ο χρήστης μπορεί να δει αναλυτικές πληροφορίες για κάθε πλατφόρμα εκτόξευσης	Υψηλή
ΛΑ-08	Αναζήτηση σχετικών ειδήσεων με φίλτρα.	Ο χρήστης μπορεί να εφαρμόζει φίλτρα για εξατομικευμένα αποτελέσματα	Υψηλή

Στη συνέχεια, στο Πίνακας 2 παρουσιάζονται ποιες από τις παραπάνω απαιτήσεις ολοκληρώθηκαν με επιτυχία και ποιες δεν ολοκληρώθηκαν, καθώς και κάποιες παρατηρήσεις.

Πίνακας 2: Ολοκληρωμένες Λειτουργικές Απαιτήσεις.

Λ/Α	Προτεραιότητα	Κατάσταση	Παρατηρήσεις
ΛΑ-01	Υψηλή	Ολοκληρώθηκε	Περιορισμένος αριθμό φίλτρων για την αναζήτηση πυραύλων, διαστημόπλοιων και προωθητήρων
ΛΑ-02	Υψηλή	Ολοκληρώθηκε	
ΛΑ-03	Υψηλή	Ολοκληρώθηκε	
ΛΑ-04	Μέση	Δεν Ολοκληρώθηκε	Χρειάζεται περισσότερη παραμετροποίηση στο backend.
ΛΑ-05	Μέση	Ολοκληρώθηκε	
ΛΑ-06	Υψηλή	Ολοκληρώθηκε	
ΛΑ-07	Υψηλή	Ολοκληρώθηκε	
ΛΑ-08	Υψηλή	Ολοκληρώθηκε	Περιορισμένος αριθμό φίλτρων, αναζήτηση στο search bar και με βάση ημερομηνία δημοσίευσης κατά φθίνουσα ή αύξουσα σειρά.

Έπειτα, στο Πίνακας 3 παρουσιάζονται οι απαιτήσεις της εφαρμογής που απαιτούν αυθεντικοποίηση (authorization) και εξειδικευμένη εξουσιοδότηση (authorization) για την πρόσβαση στα δεδομένα.

Πίνακας 3: Λειτουργικές απαιτήσεις συνδεδεμένου χρήστη.

Λ/Α	Λειτουργική Απαίτηση	Περιγραφή	Προτεραιότητα
ΛΑ-01	Μηχανισμός εγγραφής και σύνδεσης.	Οι χρήστες μπορούν να γίνουν μέλη κάνοντας εγγραφή στη σελίδα και μπορούν να συνδεθούν ξανά οποιαδήποτε στιγμή στον προσωπικό τους λογαριασμό.	Υψηλή
ΛΑ-02	Μηχανισμός για ανάκτηση κωδικού και αλλαγή κωδικού.	Οι χρήστες που έχουν ξεχάσει τον κωδικό τους μπορούν να ανακτούν των κωδικό τους ακόμη και αν δεν έχουν συνδεθεί στο λογαριασμό τους. Επίσης μπορούν να αλλάζουν τον κωδικό μέσα από το profile τους.	Υψηλή
ΛΑ-03	Προσωπικό profile με την προβολή των στοιχείων τους αλλά και τις δυνατότητες διαχείρισης του λογαριασμού τους.	Οι χρήστες έχουν προσωπικό profile όπου τους παρουσιάζονται τα προσωπικά τους στοιχεία. Επίσης, όλες οι δυνατές ρυθμίσεις που μπορούν να επιλέξουν για τον λογαριασμό τους.	Υψηλή
ΛΑ-04	Δυνατότητα διαγραφής λογαριασμού.	Ο χρήστης έχει τη δυνατότητα να διαγράφει τον λογαριασμό του.	Μέση
ΛΑ-05	Πίνακας ελέγχου για την προβολή μηνυμάτων από τη φόρμα επικοινωνίας, τα μέλη της σελίδας καθώς και άλλες λειτουργίες.	Οι χρήστες με ανώτερο ρόλο μπορούν να δουν στον πίνακα ελέγχου δεδομένα τα οποία δεν μπορούν να δουν απλή χρήστες.	Μέση
ΛΑ-06	Μηχανισμός αρχειοθέτησης εκτοξεύσεων.	Οι χρήστες έχουν τη δυνατότητα να αρχειοθετούν εκτοξεύσεις για να τις δουν αργότερα.	Υψηλή

Οι λειτουργικές απαιτήσεις του Πίνακας 3 ολοκληρώθηκαν με επιτυχία και δεν υπήρξε κάποιο πρόβλημα κατά της υλοποίηση τους το οποίο θα μπορούσε να διαμορφώσει την τελική απαίτηση από αυτό που είχε οριστεί αρχικά.

2.2 Μη Λειτουργικές Απαιτήσεις

Οι μη λειτουργικές απαιτήσεις καθορίζουν τα ποιοτικά χαρακτηριστικά και τους περιορισμούς του συστήματος. Αυτές οι απαιτήσεις δε σχετίζονται με συγκεκριμένες λειτουργίες αλλά επηρεάζουν την απόδοση, ασφάλεια και την επεκτασιμότητα.

Πίνακας 4: Μη λειτουργικές απαιτήσεις.

ΜΛ/Α	Μη Λειτουργική Απαίτηση	Περιγραφή	Προτεραιότητα
ΜΛ/Α-01	Απόκριση backend	Ο χρόνος απόκρισης του backend δεν πρέπει να υπερβαίνει το όριο του ενός δευτερολέπτου.	Υψηλή
ΜΛ/Α-02	Απόκριση frontend	Η διεπαφή χρήστη πρέπει να προσαρμόζεται σε όλες τις συσκευές.	Υψηλή
ΜΛ/Α-03	Παγκόσμιος διαμοιρασμός στατικών στοιχείων του frontend.	Ο διαμοιρασμός των εικόνων και στατικών στοιχείων πρέπει να γίνεται παγκόσμια και αποδοτικά.	Υψηλή
ΜΛ/Α-04	Συχνή αναβάθμιση δεδομένων	Τα δεδομένα για τις διαστημικές εκτοξεύσεις και τα ιστορικά στοιχεία πρέπει να ενημερώνονται σε πραγματικό χρόνο ή τουλάχιστον κάθε 1 ώρα.	Υψηλή
ΜΛ/Α-05	Ασφάλεια	Τα δεδομένα χρήστη να αποστέλλονται μέσω ασφαλούς σύνδεσης (HTTPS) και να χρησιμοποιείται αυθεντικοποίηση JWT.	Υψηλή
ΜΛ/Α-06	Έλεγχος πρόσβασης δεδομένων	Τα endpoints του backend προστατεύονται μέσω role-based access control (RBAC).	Μέση
ΜΛ/Α-07	Ευχρηστία	Η διεπαφή χρήστη να είναι φιλική και εύχρηστη για διαφορετικές ηλικιακές ομάδες.	Μέση
ΜΛ/Α-08	Συμβατότητα	Η εφαρμογή να λειτουργεί σε όλους τους σύγχρονους browsers όπως Chrome, Firefox, Edge.	Υψηλή
ΜΛ/Α-09	Επεκτασιμότητα	Το σύστημα να μπορεί να υποστηρίξει μελλοντική επέκταση με νέες λειτουργίες χωρίς σημαντική τροποποίηση της υπάρχουσας υποδομής.	Υψηλή
ΜΛ/Α-10	Ευχρηστία αναζήτησης	Τα φίλτρα αναζήτησης να είναι εύκολα κατανοητά και προσβάσιμα, ώστε να εξυπηρετούν χρήστες με περιορισμένη εμπειρία σε τέτοιου είδους εφαρμογές.	Μέση
ΜΛ/Α-11	Όριο προϋπολογισμού cloud	Το σύστημα φιλοξενείται σε περιβάλλον cloud με περιορισμένο προϋπολογισμό.	Υψηλή
ΜΛ/Α-12	Βέλτιστη χρήσης πόρων	Οι υπηρεσίες του συστήματος (βάση δεδομένων, APIs) θα πρέπει να σχεδιαστούν με γνώμονα τη μέγιστη αποδοτικότητα, περιορίζοντας την κατανάλωση πόρων και αποφεύγοντας περιττά κόστη.	Υψηλή

Σύμφωνα με τον Πίνακας 4, διαπιστώνεται ότι απαιτείται ιδιαίτερη έμφαση στην απόδοση του συστήματος. Η ταχύτητα απόκρισης ειδικά κατά την αναζήτηση εκτοξεύσεων, αποτελεί κρίσιμο παράγοντα για τη συνολική εμπειρία του χρήστη. Επιπλέον, η εφαρμογή οφείλει να παρέχει ομαλή εμπειρία χρήσης σε διαφορετικές συσκευές προσαρμόζοντας δυναμικά τη διεπαφή της. Επίσης, η προσβασιμότητα και η ευχρηστία αποτελούν επίσης βασικές προτεραιότητες για την ικανοποίηση των τελικών χρηστών.

2.3 Τεχνικές Απαιτήσεις

Στο Πίνακας 5 περιγράφονται οι τεχνικές απαιτήσεις που αφορούν το backend.

Πίνακας 5: Τεχνικές απαιτήσεις backend.

T/A	Τεχνική Απαίτηση	Περιγραφή
TR-001	Υποδομή cloud	Η εφαρμογή θα φιλοξενείται σε AWS Lightsail instance με 2 vCPUs, 2GB RAM και 60GB SSD.
TR-002	Βάση δεδομένων	Χρήση MySQL 8.0 για αποθήκευση δεδομένων εκτοξεύσεων, αστροναυτών και πολλά άλλα
TR-003	Backend framework	Ανάπτυξη backend με Spring Boot 3.0 με RESTful APIs.
TR-004	Docker containers	Οι Υπηρεσίες του backend θα πρέπει να βρίσκονται μέσα σε containers για καλύτερη διαχείριση πόρων και ασφάλεια.
TR-005	Πιστοποιητικό SSL	Χρήση Let's Encrypt SSL μέσω Nginx Reverse Proxy για ασφαλή πρόσβαση στην εφαρμογή.
TR-006	Μηχανισμός αυθεντικοποίησης	Χρήση JWT (JSON Web Tokens) για τη διαχείριση sessions και την προστασία των endpoints.
TR-007	Αναβάθμιση δεδομένων	Ο μηχανισμός ETL εκτελείται τουλάχιστον κάθε 1-2 ώρες για την ανανέωση των δεδομένων εκτοξεύσεων.
TR-008	Παρακολούθηση συστήματος	Χρήση Grafana και Prometheus για την παρακολούθηση της απόδοσης και τη συλλογή στατιστικών.

Στη συνέχεια, στο Πίνακας 6 παρουσιάζονται οι τεχνικές απαιτήσεις που αφορούν το frontend.

Πίνακας 6: Τεχνικές απαιτήσεις frontend.

T/A	Τεχνική Απαίτηση	Περιγραφή
TR-001	Frontend Framework	Χρήση React με Vite για την ανάπτυξη του frontend.
TR-002	CSS Preprocessor	Χρήση του SASS CSS preprocessor για την ανάπτυξη των γραφικών στοιχείων.
TR-003	S3 Cloud Storage	Χρήση του Amazon S3 cloud storage για την αποθήκευση στατικών στοιχείων του

		frontend (όπως αρχεία HTML, CSS, JS) και εικόνων σχετικών με τη λειτουργία της εφαρμογής.
TR-004	CloudFrond	Χρήση του Amazon CloudFront για την παγκόσμια διανομή περιεχομένου μέσω ενός Content Delivery Network (CDN).
TR-005	Lambda@edge	Χρήση Lambda@edge συναρτήσεων για την ανακατεύθυνση των αιτημάτων από crawlers των μέσων κοινωνικής δικτύωσης
TR-006	Πιστοποιητικό TSL/SSL X.509	Χρήση πιστοποίησης X.509 SSL για ασφαλής εσωτερική επικοινωνία των υπηρεσιών Amazon Web Services.

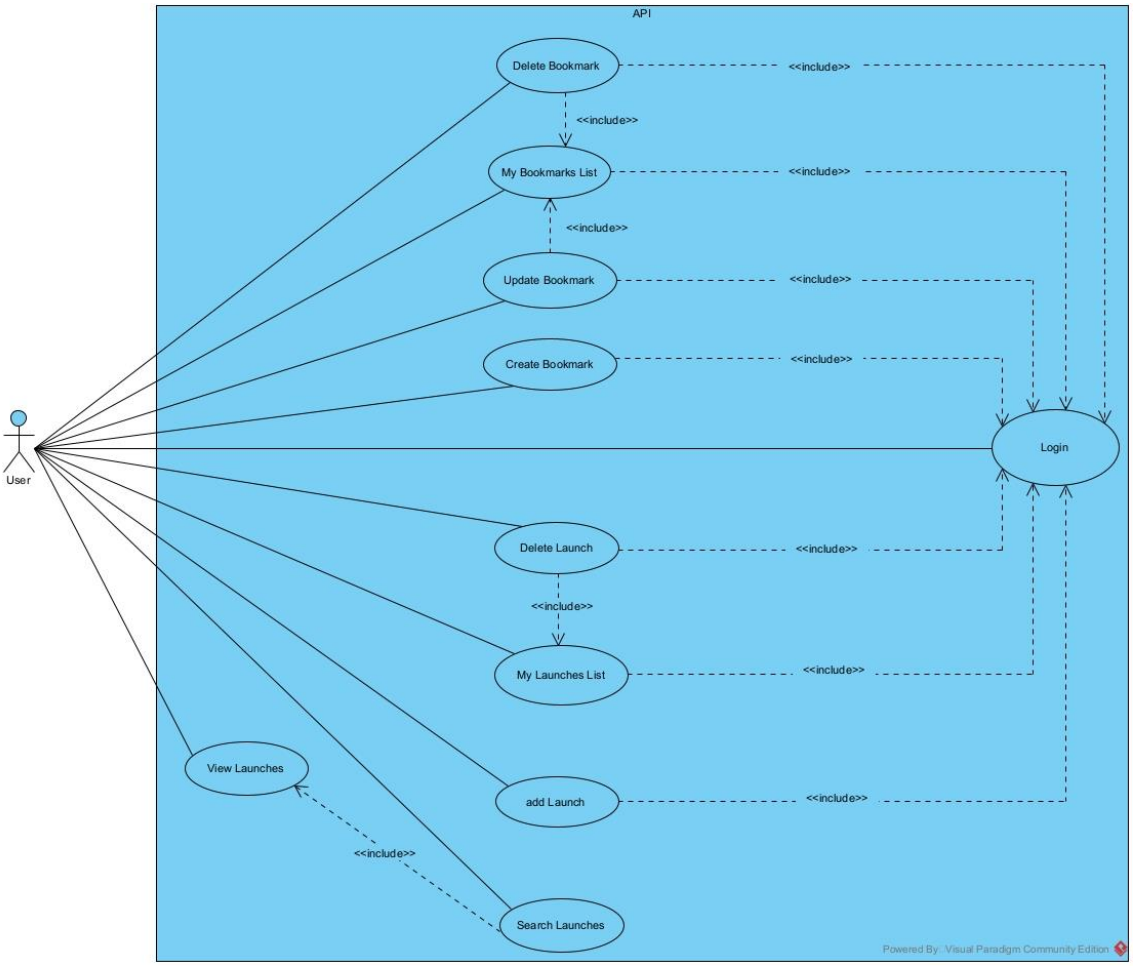
2.4 Μελλοντικές Επεκτάσεις

Στην ενότητα αυτή, παρουσιάζονται λειτουργίες και χαρακτηριστικά που πρόκειται να επεκτείνουν την εφαρμογή στο μέλλον, εμπλουτίζοντας το περιεχόμενο της εφαρμογής με νέες λειτουργίες και εξασφαλίζοντας στους χρήστες νέο περιεχόμενο. Οι μελλοντικές επεκτάσεις έχουν ως εξής:

- Παρακολούθηση καιρικών συνθηκών για συγκεκριμένες πλατφόρμες εκτοξεύσεων: Προσθήκη βοηθητικού παραθύρου (widget) για την παρακολούθηση των καιρικών συνθηκών για τις πιο γνωστές πλατφόρμες εκτοξεύσεων. Προσθήκη ενότητας διαστημικών σταθμών: Θα παρουσιάζονται αναλυτικές πληροφορίες σχετικά με τους ενεργούς και μη ενεργούς διαστημικούς σταθμούς.
- Προσθήκη ενότητας με όλα τα αναμενόμενα γεγονότα: Προσθήκη ενότητας για την παρακολούθηση όλων των αναμενόμενων γεγονότων.
- Προσθήκη ειδοποιήσεων: Οι χρήστες θα μπορούν να λαμβάνουν ειδοποιήσεις λίγα λεπτά πριν γίνει η εκτόξευση.
- Βελτιστοποίηση απόδοσης της διαδικασίας εξαγωγής μετασχηματισμού και φόρτωσης των δεδομένων (ETL): Βελτιστοποίηση της απόδοσης του API που εκτελεί τη διαδικασία της εξαγωγής μετασχηματισμού και φόρτωσης των δεδομένων. Έτσι ώστε να διευκολυνθεί η μελλοντική επέκταση των δεδομένων αλλά και η συχνότερη αναβάθμιση τους.
- Ολοκλήρωση του μηχανισμού 0Auth2: Ολοκλήρωση του μηχανισμού OAuth2 για την αυθεντικοποίηση χρηστών μέσω τρίτων παρόχων (π.χ. Google, Facebook κ.λπ.), επιτρέποντας στους χρήστες να εγγραφούν και να συνδεθούν στην εφαρμογή με ασφάλεια και ευκολία.

- Βελτιστοποίηση για τις μηχανές αναζήτησης (SEO): Βελτιστοποίηση για τις μηχανές αναζήτησης προσθέτοντας δυναμικά sitemaps για γρηγορότερη ευρετηρίαση των σελίδων. Επίσης, βελτιστοποίηση του υπάρχοντος μηχανισμού Server Side Rendering (SSR), με στόχο την καλύτερη διανομή της σελίδας σε περισσότερα μέσα κοινωνικής δικτύωσης.

2.5 Διάγραμμα Περιπτώσεων Χρήσης



Εικόνα 1: Διάγραμμα περιπτώσεων χρήσης.

Στην Εικόνα 1 παρουσιάζονται τα βασικότερα σενάρια χρήσης της εφαρμογής. Επιπλέον, για τους σκοπούς του παραδείγματος θα υποθέσουμε ότι ο χρήστης έχει ήδη λογαριασμό στην εφαρμογή.

Σενάριο χρήσης 1 - Αναζήτηση εκτοξεύσεων

Ο χρήστης εισάγει τα απαραίτητα φίλτρα για την αναζήτηση εξατομικευμένων αποτελεσμάτων.

Συμμετέχοντες

- Χρήστης (Actor)
- Εφαρμογή.

Βασική Ροή

1. Ο χρήστης μεταβαίνει στην ιστοσελίδα μέσω του www.moonkeyeu.com.
2. Ο χρήστης εφαρμόζει φίλτρα αναζήτησης χρησιμοποιώντας το εργαλείο με όλα τα διαθέσιμα φίλτρα ή πληκτρολογώντας στη μπάρα αναζήτησης το όνομα της εκτόξευσης.
3. Το σύστημα ελέγχει εάν υπάρχουν εκτοξεύσεις με τα φίλτρα αναζήτησης και τις επιστρέφει.

Το παραπάνω σενάριο χρήσης ισχύει για όλες τις ενότητες της εφαρμογής όπου μπορεί ο χρήστης να εφαρμόσει φίλτρα αναζήτησης (π.χ. αναζήτηση αστροναυτών, αναζήτηση προγραμμάτων κ.λπ.).

Εναλλακτικές Ροές

- 4a. Αν δεν υπάρχουν σχετικά αποτελέσματα, το σύστημα εμφανίζει σχετικό μήνυμα.

Σενάριο χρήσης 2 - Σύνδεση Χρήστη

Ο χρήστης εισάγει τα στοιχεία του για να συνδεθεί στην εφαρμογή έτσι ώστε να αποκτήσει πρόσβαση στις λειτουργίες που απαιτούν αυθεντικοποίηση χρήστη.

Συμμετέχοντες

- Χρήστης (Actor)
- Εφαρμογή.

Προϋποθέσεις

Ο χρήστης πρέπει να έχει ήδη λογαριασμό στο σύστημα.

Βασική Ροή

1. Ο χρήστης ανακατευθύνεται στη σελίδα πληκτρολογώντας τον σύνδεσμο www.moonkeyeu.com.
2. Ανοίγει τη φόρμα σύνδεσης.
3. Εισάγει το email και τον κωδικό του.
4. Το σύστημα ελέγχει εάν τα διαπιστευτήρια είναι σωστά.
5. Το σύστημα δημιουργεί συνεδρία για τον χρήστη μέσω ενός cookie όπου αποθηκεύετε το refresh token και τον ανακατευθύνει στην κύρια σελίδα με τις εκτοξεύσεις.

Εναλλακτικές Ροές

- 4a. Αν τα διαπιστευτήρια είναι λανθασμένα, το σύστημα εμφανίζει μήνυμα λάθους και ζητά από τον χρήστη να επαναλάβει τη διαδικασία.
- 4b. Αν ο χρήστης εισάγει λανθασμένα διαπιστευτήρια τρείς φορές, τότε το σύστημα μπλοκάρει τον λογαριασμό του για δεκαπέντε λεπτά και κάθε επόμενη φορά που ξεπερνά αυτό τον αριθμό αυξάνεται κλιμακωτά.

Σενάριο χρήσης 3 - Αρχειοθέτηση εκτοξεύσεων για αργότερη ανάγνωση

Ο χρήστης δημιουργεί ένα σελιδοδείκτη και αποθηκεύει σε αυτό όλες τις εκτοξεύσεις που θέλει να διαβάσει αργότερα.

Συμμετέχοντες

- Χρήστης (Actor)
- Εφαρμογή.

Προϋποθέσεις

Ο χρήστης πρέπει να είναι συνδεδεμένος στο σύστημα.

Βασική Ροή

1. Ο χρήστης ανοίγει τη φόρμα δημιουργίας σελιδοδείκτη.
2. Ο χρήστης εισάγει το όνομα του σελιδοδείκτη.
3. Το σύστημα αποθηκεύει τον σελιδοδείκτη στη βάση δεδομένων.
4. Το σύστημα εμφανίζει μήνυμα επιτυχούς δημιουργίας του σελιδοδείκτη.
5. Ο χρήστης επιλέγει σε πιο σελιδοδείκτη θα αποθηκεύσει μια εκτόξευση.
6. Το σύστημα εμφανίζει μήνυμα επιτυχούς αποθήκευσης της εκτόξευσης.

Εναλλακτικές Ροές

- 3a. Αν ο σελιδοδείκτης υπάρχει ήδη τότε το σύστημα εμφανίζει μήνυμα λάθους.
- 5a. Αν η εκτόξευση υπάρχει ήδη στη λίστα του σελιδοδείκτη τότε εμφανίζει μήνυμα λάθους.

Σενάριο χρήσης 4 - Διαγραφή αρχειοθετημένων εκτοξεύσεων

Ο χρήστης διαγράφει από τον σελιδοδείκτη εκτοξεύσεις που δεν τον ενδιαφέρουν.

Συμμετέχοντες

- Χρήστης (Actor)
- Εφαρμογή.

Προϋποθέσεις

Ο χρήστης πρέπει να είναι συνδεδεμένος στο σύστημα.

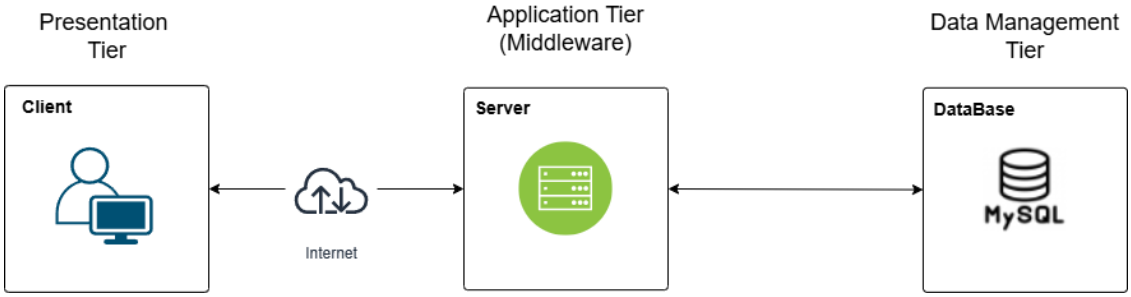
Βασική Ροή

1. Ο χρήστης ανακατευθύνεται στη σελίδα με τους σελιδοδείκτες.
2. Ο χρήστης επιλέγει τον σελιδοδείκτη που τον ενδιαφέρει.
3. Ο χρήστης ανακατευθύνεται στη σελίδα με τις εκτοξεύσεις του, όπου μπορεί να επιλέξει αυτές που θέλει να διαγράψει.

Εναλλακτικές Ροές

- 1a. Αν ο χρήστης δεν είναι συνδεδεμένος εμφανίζεται κατάλληλο μήνυμα προτροπής για σύνδεση.
- 2a. Αν δεν υπάρχουν σελιδοδείκτες τότε εμφανίζεται κατάλληλο μήνυμα.
- 3a. Αν δεν υπάρχουν εκτοξεύσεις στον σελιδοδείκτη τότε εμφανίζεται κατάλληλο μήνυμα.

2.6 Αρχιτεκτονική Τριών Επιπέδων



Εικόνα 2: Αρχιτεκτονική τριών επιπέδων.

Η εφαρμογή ακολουθεί την αρχιτεκτονική τριών επιπέδων (3-tier) και διαχωρίζει τη λειτουργικότητα σε τρία επίπεδα, με στόχο την επεκτασιμότητα, συντηρησιμότητα αλλά και την επαναχρησιμοποίηση του κώδικα:

- Επίπεδο Παρουσίασης (Presentation Tier): Υλοποιείται μέσω μιας εφαρμογής React, η οποία αποτελεί το frontend. Το επίπεδο αυτό είναι υπεύθυνο για την εμφάνιση του περιεχομένου στον τελικό χρήστη και τη διαχείριση της αλληλεπίδρασής του με την εφαρμογή.
- Επίπεδο Λογικής Εφαρμογής (Application/Logic Tier): Υλοποιείται μέσω ενός API σε Spring Boot, το οποίο χειρίζεται τη ροή των δεδομένων μέσω JSON αιτημάτων και απαντήσεων (request/response). Επιπλέον υλοποιεί τη λογική της εφαρμογής και τη διασύνδεση μεταξύ frontend και βάσης δεδομένων. Η λογική ακολουθεί το πρότυπο MVC (Model-View-Controller), με διαχωρισμένα controller, cervice και repository επίπεδα.

- Επίπεδο Δεδομένων (Data Tier): Υλοποιείται μέσω μιας βάσης δεδομένων MySQL, η οποία αποθηκεύει όλα τα δεδομένα της εφαρμογής, όπως χρήστες, αποστολές, bookmarks και πολλά αλλά.

Έτσι, η αρχιτεκτονική 3-tier προσφέρει καθαρό διαχωρισμό ευθυνών και διευκολύνει την ανεξάρτητη ανάπτυξη, διασφάλιση ποιότητας και συντήρηση κάθε επιπέδου.

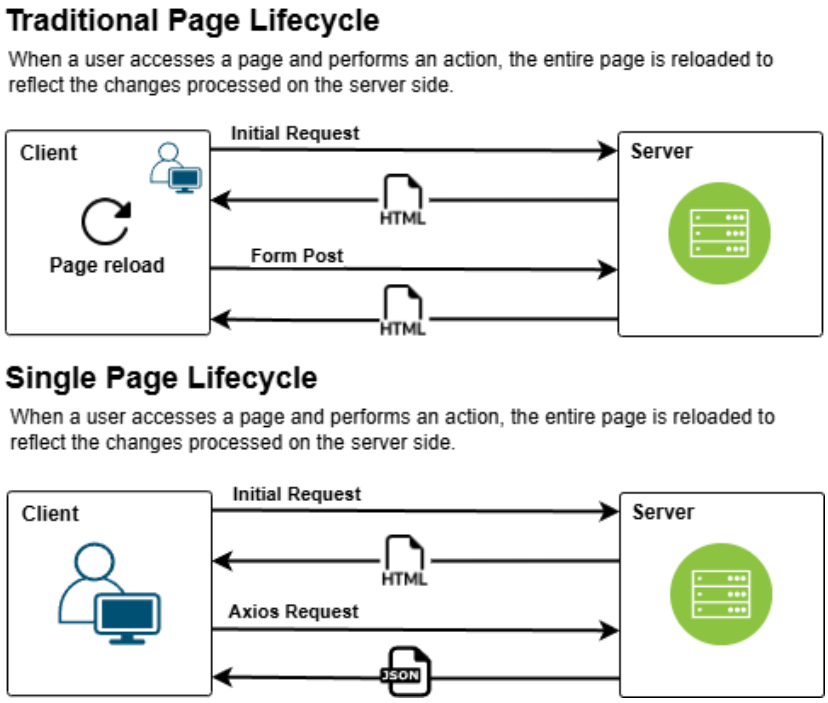
3 Ανάλυση και Υλοποίηση Frontend

Σε αυτή την ενότητα αναλύουμε την αρχιτεκτονική του frontend, παρουσιάζεται η υλοποίηση του πρωτοτύπου του και οι βιβλιοθήκες τρίτου μέρους όπου αποτελούν ζωτικής σημασίας για την υλοποίηση του.

3.1 Ανάλυση Αρχιτεκτονικής Frontend

Σε αυτή την υποενότητα αναλύεται η αρχιτεκτονική του frontend. Αρχικά παρουσιάζονται τα Single Page Application και αναλύεται η αρχιτεκτονική Component Based. Έπειτα παρουσιάζονται τα Frameworks και οι βιβλιοθήκες που υποστηρίζουν αυτή την η αρχιτεκτονική και αναφέρονται οι λόγοι για τους οποίους επιλέχθηκε η βιβλιοθήκη React.

3.1.1 Ανάλυση των Single Page Applications



Εικόνα 3: Λειτουργικότητα των Single Page Applications.

Πριν προχωρήσουμε στην ανάλυση της αρχιτεκτονικής Component Based αρχικά θα πρέπει κατανοηθεί τι είναι ένα Single Page Application. Ένα Single Page Application

(SPA) είναι ένας τύπος web εφαρμογής που μπορεί να ανανεώνει δυναμικά το περιεχόμενο του χωρίς να απαιτείται πλήρης ανανέωση της σελίδας. Σε αντίθεση με τα παραδοσιακά Multi Page Applications (MPA), όπου κάθε νέα αλληλεπίδραση απαιτεί πλήρες αίτημα στον διακομιστή και επαναφόρτωση σελίδας, ένα SPA ανακτά και ενημερώνει μόνο τα απαραίτητα δεδομένα.

Επιπλέον, ένα από τα κύρια πλεονεκτήματά του είναι ότι ο browser φορτώνει εξαρχής όλα τα στατικά στοιχεία της εφαρμογής όπως JavaScript, CSS και HTML με ένα μόνο αίτημα, βελτιώνοντας έτσι την απόδοση και μειώνοντας τους χρόνους φόρτωσης. Στη συνέχεια, υπάρχουν αρκετή τρόποι με τους οποίους ένα Single Page Application παράγει μια HTML σελίδα, οι οποίοι είναι οι εξής:

- Τεχνική Client Side Rendering (CSR): Η τεχνική Client-Side-Rendering (CSR) επιτρέπει την παραγωγή HTML σελίδων με την οποία ο browser ζητά ένα βασικό αρχείο HTML από τον διακομιστή μαζί με τα απαραίτητα αρχεία CSS και JavaScript. Κατά την αρχική φόρτωση, ο χρήστης βλέπει μια κενή σελίδα ή έναν loader μέχρι να εκτελεστεί η JavaScript που ανακτά τα δεδομένα και δημιουργεί τις προβολές (views) όπου τις εισάγει δυναμικά στο DOM. Αν και το CSR μπορεί να είναι πιο αργό σε απλές ιστοσελίδες λόγω της χρήσης πόρων από τη συσκευή του χρήστη, είναι ιδανικό για εφαρμογές με έντονη δραστηριότητα μειώνοντας το φόρτο εργασίας του backend. Σε περιπτώσεις που απαιτείται καλύτερη βελτιστοποίηση για διαμοιρασμό σε μέσα κοινωνική δικτύωσης (social sharing) ή για λόγους SEO τεχνικές όπως το Server-Side Rendering (SSR) ή το Static Site Generation (SSG) είναι πιο κατάλληλες.
- Τεχνική Server Side Rendering (SSR): Η τεχνική Server-Side-Rendering (SSR) επιτρέπει την παραγωγή HTML σελίδων και η δημιουργία των προβολών (views) πραγματοποιείται στην πλευρά του διακομιστή πριν αποσταλεί στον browser. Κατά τη διαδικασία αυτή, ο διακομιστής λαμβάνει το αίτημα από τον browser, συγκεντρώνει τα απαραίτητα δεδομένα, κατασκευάζει τη σελίδα και την αποστέλλει ως HTML στον χρήστη. Αυτό επιτρέπει την άμεση προβολή του περιεχομένου χωρίς κενές οθόνες ή loaders, βελτιώνοντας την ταχύτητα εμφάνισης και τη βελτιστοποίηση για τις μηχανές αναζήτησης (SEO).
- Τεχνική Static Site Generation (SSG): Η τεχνική Static-Site-Generation (SSG) επιτρέπει την παραγωγή HTML σελίδων οι οποίες δημιουργούνται στατικά και αποθηκεύονται ως έτοιμα HTML αρχεία στον διακομιστή. Όταν ο browser ζητά την ιστο-

σελίδα, ο διακομιστής απλώς σερβίρει την προκατασκευασμένη σελίδα, επιτυγχάνοντας πολύ γρήγορη φόρτωση. Αυτή η προσέγγιση είναι ιδανική για ιστοσελίδες με στατικό περιεχόμενο, καθώς οι σελίδες είναι ήδη έτοιμες και δεν απαιτείται δυναμική δημιουργία τους σε πραγματικό χρόνο. Ωστόσο, δεν είναι τόσο κατάλληλη για εφαρμογές με συχνές αλλαγές δεδομένων, καθώς απαιτεί εκ νέου δημιουργία (rebuild) για να ενημερωθεί το περιεχόμενο.

Αφού αναλύθηκαν οι τεχνικές παραγωγής HTML σελίδων στη συνέχεια αναφέρονται τα πλεονεκτήματα και τα μειονεκτήματα των SPA. Τα πλεονεκτήματα τους είναι τα εξής:

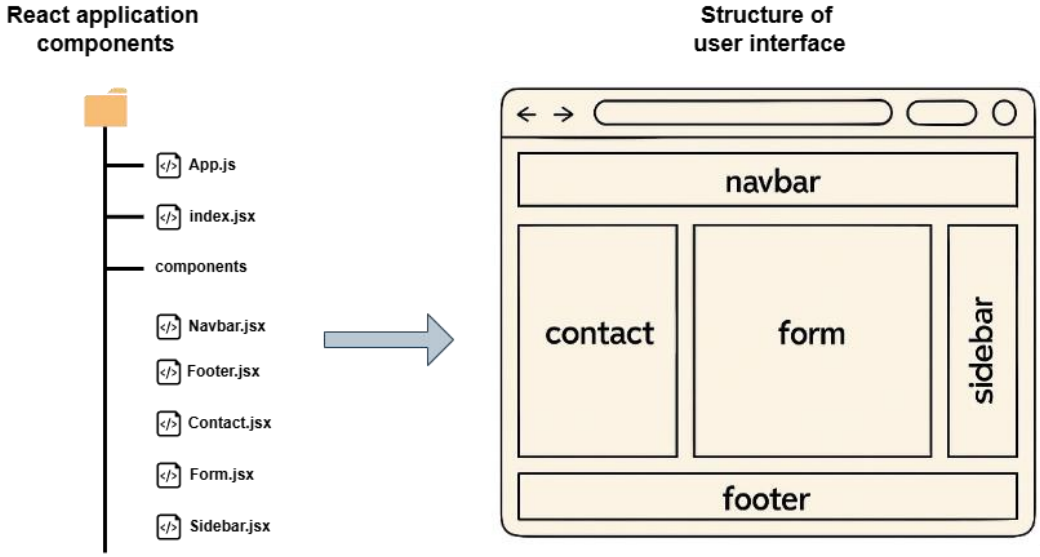
- Ταχύτερη εμπειρία χρήστη: Τα SPA φορτώνουν μόνο μία φορά και ανακτούν μόνο τα απαραίτητα δεδομένα, μειώνοντας τις πλήρεις ανανεώσεις της σελίδας και προσφέροντας μια πιο ομαλή και άμεση εμπειρία.
- Caching για πρόσβαση εκτός σύνδεσης: Τα SPA μπορούν να υλοποιήσουν στρατηγικές Caching, επιτρέποντας στους χρήστες να έχουν πρόσβαση σε συγκεκριμένα τμήματα της εφαρμογής ακόμη και όταν είναι εκτός σύνδεσης.
- Βελτιωμένη απόδοση: Με την ελαχιστοποίηση των αιτημάτων στον διακομιστή και την ενημέρωση μόνο των απαραίτητων στοιχείων, τα SPA μειώνουν σημαντικά το φορτίο στους διακομιστές, με αποτέλεσμα ταχύτερους χρόνους φόρτωσης και καλύτερη συνολική απόδοση.
- Μειωμένη χρήση πόρων δικτύου (bandwidth): Επειδή τα SPA ανακτούν μόνο τα δεδομένα που χρειάζονται, μειώνεται η ποσότητα δεδομένων που μεταφέρεται μεταξύ πελάτη και διακομιστή, εξοικονομώντας πόρους δικτύου και βελτιώνοντας την απόδοση.
- Βελτιωμένη απόκριση: Επιτρέπουν δυναμικές ενημερώσεις περιεχομένου χωρίς να απαιτείται πλήρης ανανέωση της σελίδας.
- Ομαλή πλοήγηση χρήστη: Τα SPA χρησιμοποιούν client side routing, επιτρέποντας ομαλή πλοήγηση μεταξύ των τμημάτων της εφαρμογής χωρίς να χρειάζεται πλήρης ανανέωση της σελίδας.

Εκτός από τα πλεονεκτήματα που έχουν τα Single Page Application τα οποί προσφέρουν μι ομαλή εμπειρία στο χρήστη κατά τη χρήση της εφαρμογής έχουν επίσης και κάποια μειονεκτήματα:

- Πιο αργή αρχική φόρτωση: Μπορεί να είναι πιο αργά κατά την αρχική φόρτωση, επηρεάζοντας αρνητικά τους χρήστες με αργή σύνδεση στο διαδίκτυο.

- Δυσκολότερη διαχείριση του Search Engine Optimization (SEO): Η βελτιστοποίηση για τις μηχανές αναζήτησης (SEO) είναι πιο δύσκολη λόγω της μεγάλης εξάρτησης από τη JavaScript.
- Περιορισμένη υποστήριξη από browsers: Ορισμένα προηγμένα χαρακτηριστικά μπορεί να μη λειτουργούν σωστά σε παλαιότερους browsers.
- Κίνδυνοι ασφαλείας: Είναι ευάλωτα σε ζητήματα ασφαλείας, όπως το Cross-Site Scripting (XSS) και απαιτούνται τεχνικές προστασίας που αναλύονται στην ενότητα 5.
- Υψηλή κατανάλωση πόρων στην πλευρά του πελάτη: Τα SPA επιβαρύνουν περισσότερο τη συσκευή του χρήστη, επηρεάζοντας την απόδοση σε παλαιότερες συσκευές.
- Πολύπλοκη ανάπτυξη: Η ανάπτυξη ενός SPA είναι πιο περίπλοκη και απαιτεί μεγαλύτερη εξοικείωση με τεχνολογίες frontend.

3.1.2 Μοντέλο Αρχιτεκτονικής Component Based



Εικόνα 4: Αρχιτεκτονική Component Based.

Η αρχιτεκτονική Component Based είναι ένα πρότυπο σχεδίασης όπου μια εφαρμογή χρησιμοποιεί αυτόνομα επαναχρησιμοποιήσιμα components. Κάθε ένα component ενσωματώνει ένα συγκεκριμένο κομμάτι λειτουργικότητας ή διεπαφής χρήστη. Επιπλέον, αυτά τα components μπορούν να ενωθούν και να δημιουργήσουν μια ολοκληρωμένη εφαρμογή.

Παρατηρώντας την την Εικόνα 4 το παράθυρο της γραφικής διασύνδεσης αποτελείται από πέντε components, το navbar, footer, form, contact και το sidebar. Ενώνοντας όλα

αυτά τα components κατασκευάζεται η γραφική διασύνδεση χρήστη. Επίσης, μπορούν να επαναχρησιμοποιηθούν και σε άλλα σημεία της εφαρμογής.

Επιπλέον, τα Single Page Applications (SPA) ωφελούνται από την αρχιτεκτονική Component Based διότι από τη φύση τους αποτελούνται από μια σελίδα και κατασκευάζοντας ανεξάρτητα components επιτυγχάνονται τα ακόλουθα:

- Διευκόλυνση της συντηρησιμότητας: Τα components διαχειρίζονται και αναβαθμίζονται πιο εύκολα αφού είναι μεμονωμένα και δεν επηρεάζουν ολόκληρη την εφαρμογή.
- Επιτυγχάνεται η δυνατότητα επαναχρησιμοποίησης των components: Σωστά σχεδιασμένα components μπορούν να επαναχρησιμοποιηθούν σε ολόκληρη την εφαρμογή.
- Απλοποιούν τη διαδικασία της ορθότητας λειτουργίας: Σε κάθε component μπορεί να διαπιστωθεί πιο εύκολα η ορθότητα λειτουργίας του, αφού είναι ανεξάρτητο από την υπόλοιπη εφαρμογή.

Στο πλαίσιο της εργασία, χρησιμοποιήθηκε η αρχιτεκτονική Component Based Single Page Application με τις τεχνικές παραγωγής HTML σελίδων CSR και SSR. Η τεχνική CSR χρησιμοποιήθηκε για την παραγωγή των HTML σελίδων από την πλευρά των πελατών. Αντίθετα, η τεχνική SSR χρησιμοποιήθηκε για λόγους διαμοιρασμού του περιεχομένου σε μέσα κοινωνικής δικτύωσης. Αφού ένα Single Page Application από την φύση του δεν μπορεί να καλύψει αυτή την ανάγκη διότι κατά την πρώτη φόρτωση του περιεχομένου του η σελίδα είναι κενή και οι crawler των μέσων κοινωνική δικτύωσης δε θα μπορέσουν να ανακτήσουν σωστά το περιεχόμενο της σελίδας. Περαιτέρω ανάλυση της υλοποίησης του μηχανισμού SSR αναλύεται στην ενότητα 6.

3.1.3 Υποστηριζόμενα Frameworks και Βιβλιοθήκες

Υπάρχουν αρκετά frameworks και βιβλιοθήκες που υποστηρίζουν τη Component-Based αρχιτεκτονική, όπως Vue, Angular και Svelte. Ωστόσο, για την υλοποίηση του frontend επιλέχθηκε η βιβλιοθήκη React για τους εξής λόγους:

- Ευκολία εκμάθησης: Σε σύγκριση με άλλα frameworks, η React έχει μια πιο απλή και κατανοητή προσέγγιση, ειδικά αν κάποιος έχει ήδη γνώσεις JavaScript.
- Υψηλή απόδοση: Η χρήση του virtual DOM (Document Object Model) αποτελεί ένας από τους βασικότερους παράγοντες διαφοροποίησης σε σχέση με τα αλλά frameworks. Ο virtual DOM ελαχιστοποιεί τις περιττές ενημερώσεις του

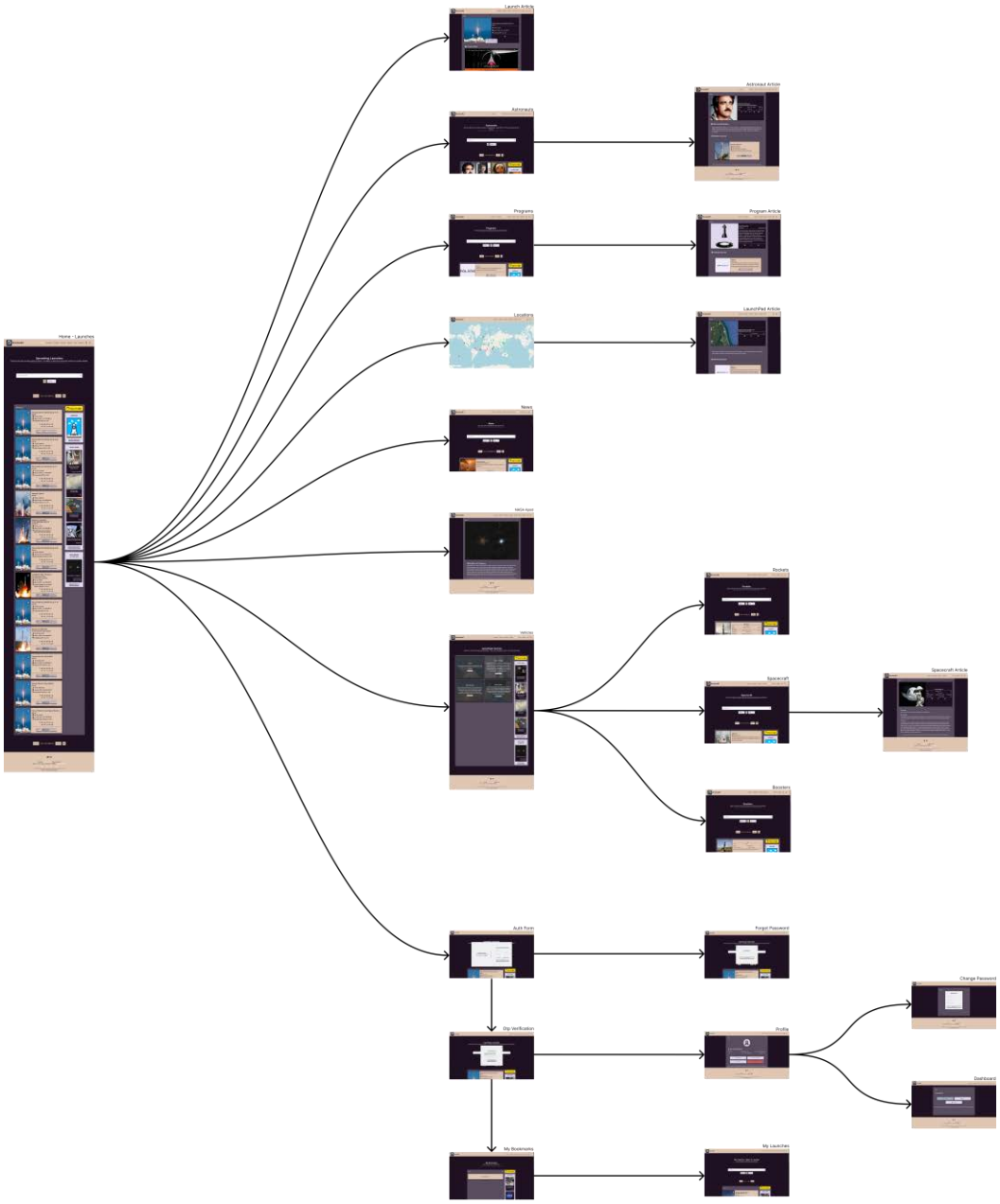
πραγματικού DOM, βελτιώνοντας την ταχύτητα και την αποδοτικότητα της εφαρμογής.

- Μεγάλη κοινότητα και οικοσύστημα: Η React υποστηρίζεται από μια τεράστια κοινότητα προγραμματιστών και έχει πλούσιο οικοσύστημα βιβλιοθηκών, όπως React Router, Redux, React TanStack και πολλές άλλες βιβλιοθήκες.
- Μεγάλος αριθμός εφαρμογών που την υποστηρίζουν: Η React μπορεί να χρησιμοποιηθεί για την ανάπτυξη διαδικτυακών αλλά και κινητών εφαρμογών. Επίσης, υποστηρίζεται από μεγάλες εταιρίες στο χώρο όπως το Facebook, Twitter, Netflix και πολλές άλλες.
- Διαθεσιμότητα εργαλείων για αποσφαλματοποίησης (Debugging): Η React προσφέρει εργαλεία αποσφαλματοποίησης τα οποία μπορούν να επιταχύνουν τη διαδικασία της ανάπτυξης. Κάποια από τα πιο δημοφιλή εργαλεία είναι React Develop Tools και Redux Developer Tools το οποίο χρησιμοποιείται για εφαρμογές που βασίζονται στη βιβλιοθήκη Redux.

3.2 Υλοποίηση Πρωτοτύπου UI/UX στο Figma

Εικόνα 5: Λογότυπο Figma [17].

Το Figma είναι ένα εργαλείο σχεδίασης που χρησιμοποιείται κυρίως για τη δημιουργία διαπαφών χρήστη (UI) και προσομοιάζει την εμπειρία (UX) του χρήστη σε αυτή. Επίσης, χρησιμοποιείται για την ανάπτυξη όλων των ειδών εφαρμογών όπως ιστότοπους και εφαρμογές για κινητά. Επιπλέον, ο cloud-based χαρακτήρας του Figma σημαίνει ότι είναι προσβάσιμο από οπουδήποτε και πολλά μέλη της ομάδας μπορούν να εργάζονται ταυτόχρονα στο ίδιο έργο. Στη συνέχεια παρουσιάζεται το πρωτότυπο της εφαρμογής το οποίο υλοποιήθηκε με το Figma και ο σκοπός του είναι να προσφέρει μία πρώτη εικόνα για το πως θα είναι η εφαρμογή.



Εικόνα 6: Πλοήγηση χρήστη στην εφαρμογή.

- Διάταξη σελίδων: Αρχικά έχουμε το header με το menu της σελίδας. Στη συνέχεια, υπάρχει το βασικό περιεχόμενο, όπου στο κέντρο έχουμε το περιεχόμενο που αφορά κάθε κατηγορία και στην άκρη του υπάρχει ένα sidebar το οποίο δείχνει συνοπτικά όλα τα τελευταία νέα. Επίσης, στο τέλος της κάθε σελίδας εκτός από την κατηγορία locations υπάρχει το footer στο οποίο ο χρήστης μπορεί να βρει διάφορες πληροφορίες σχετικά με τη σελίδα.
- Διάταξη καρτών: Κάθε κάρτα έχει ένα τίτλο με αυτό που περιγραφή και άλλες διάφορες πληροφορίες. Επίσης, στις κάρτες των εκτοξεύσεων υπάρχει ένα χρονόμετρο το οποίο προσδιορίζει πότε θα γίνει η εκτόξευση. Στο κάτω μέρος των

καρτών υπάρχουν συνήθως τα κουμπιά για την προβολή λεπτομερειών (Info), το κουμπί (Watch) για το βίντεο της εκτόξευσης, το κουμπί (View Launch) την προβολή των εκτοξεύσεων που αφορούν το συγκεκριμένο όχημα ή είδηση. Επίσης, τα κουμπιά (Wiki) και (Share) για την πληροφόρηση του χρήστη από τη σελίδα wiki και τον διαμοιρασμό των λεπτομερειών που αφορούν τη συγκεκριμένη κάρτα.

- Αρχική σελίδα (home ή launches): Όταν ο χρήστης εισέρχεται στην εφαρμογή μέσω του βασικού συνδέσμου, μεταφέρεται στη σελίδα με όλες τις προσεχής εκτοξεύσεις που πρόκειται να γίνουν. Επίσης, υπάρχουν διάφορα εργαλεία όπως το πεδίο αναζήτησης και διάφορα φίλτρα που μπορεί να εφαρμόσει για εξατομικευμένη αναζήτηση.
- Άρθρο για εκτόξευση (launches/{id}): Όταν ο χρήστης πατήσει το κουμπί (Info) σε μία κάρτα τότε μεταφέρεται στη σελίδα με όλες τις λεπτομέρειες που αφορούν αυτή την εκτόξευση.
- Σελίδα με αστροναύτες (astronauts): Στο κοινό menu το οποίο βρίσκεται στο πάνω μέρος της σελίδας (header) επιλέγοντας την επιλογή astronauts ο χρήστης μεταφέρεται στη σελίδα με όλους τους αστροναύτες. Επίσης, υπάρχουν διάφορα εργαλεία όπως το πεδίο αναζήτησης και διάφορα φίλτρα που μπορεί να εφαρμόσει για εξατομικευμένη αναζήτηση.
- Άρθρο για αστροναύτη (astronaut/{id}): Όταν ο χρήστης πατήσει το κουμπί (Info) σε μία κάρτα τότε μεταφέρεται στη σελίδα με όλες τις λεπτομέρειες που αφορά τον συγκεκριμένο αστροναύτη.
- Σελίδα με οργανισμούς (agencies): Στο κοινό menu επιλέγοντας την επιλογή agencies ο χρήστης μεταφέρεται στη σελίδα με όλους τους πιο σημαντικούς οργανισμούς.
- Άρθρο για οργανισμό (agency/{id}): Όταν ο χρήστης πατήσει το κουμπί (Info) σε μία κάρτα τότε μεταφέρεται στη σελίδα με όλες τις λεπτομέρειες που αφορούν αυτό τον οργανισμό.
- Σελίδα με προγράμματα (programs): Στο κοινό menu επιλέγοντας την επιλογή programs ο χρήστης μεταφέρεται στη σελίδα με όλα τα προγράμματα. Επίσης, υπάρχουν διάφορα εργαλεία όπως το πεδίο αναζήτησης και διάφορα φίλτρα που μπορεί να εφαρμόσει για εξατομικευμένη αναζήτηση.
- Άρθρο για πρόγραμμα (program/{id}): Όταν ο χρήστης πατήσει το κουμπί (Info) σε μία κάρτα τότε μεταφέρεται στη σελίδα με όλες τις λεπτομέρειες που αφορούν αυτό το πρόγραμμα.

- Σελίδα με όλα τα διαστημικά οχήματα (vehicles): Στο κοινό menu πατώντας την επιλογή vehicles ο χρήστης μεταφέρεται στη σελίδα με όλες τι κατηγορίες διαστημικών οχημάτων όπως πυραύλους (rockets), προωθητήρες (rocket boosters) και διαστημόπλοια (spacecraft).
- Σελίδες πυραύλων (rockets), προωθητήρων (launchers) και διαστημόπλοιων (spacecraft): Στη σελίδα με τις κατηγορίες διαστημικών οχημάτων (vehicles) ο χρήστης μπορεί να επιλέξει μια από αυτές τις κατηγορίες και να δει τη λίστα με όλα τα διαστημικά οχήματα αυτής της κατηγορίας. Επιπλέον, στις κάρτες των οχημάτων που αφορούν τους προωθητήρες αλλά και τους πυραύλους υπάρχει το κουμπί (View Launch). Αν πατηθεί από τον χρήστη, θα μεταφερθεί στη σελίδα με τις εκτοξεύσεις. Ενώ στο σύνδεσμο θα έχει προστεθεί το προσδιοριστικό (identifier) του συγκεκριμένου οχήματος και θα εμφανιστούν οι εκτοξεύσεις που έχει χρησιμοποιηθεί το συγκεκριμένο όχημα.
- Σελίδα με νέα (news): Στο κοινό menu πατώντας την επιλογή news ο χρήστης μεταφέρεται στη σελίδα με όλα τελευταία νέα. Επίσης, υπάρχουν διάφορα εργαλεία όπως το πεδίο αναζήτησης και διάφορα φίλτρα που μπορεί να εφαρμόσει για εξατομικευμένη αναζήτηση.
- Σελίδα με πλατφόρμες εκτοξεύσεων (locations): Στο κοινό menu επιλέγοντας την επιλογή locations ο χρήστης μεταφέρεται σε ένα χάρτη ο οποίος έχει κάποια points που προσδιορίζουν το που βρίσκεται η κάθε πλατφόρμα. Επιπλέον, κάθε point είναι χρωματισμένο με κόκκινο εάν η πλατφόρμα είναι ανενεργή και με πράσινο εάν είναι ενεργή. Επίσης, υπάρχουν διάφορα εργαλεία όπως το πεδίο αναζήτησης και διάφορα φίλτρα που μπορεί να εφαρμόσει για εξατομικευμένη αναζήτηση.
- Σελίδα αστρονομικής εικόνας της ημέρας από την NASA: Σε αυτή τη σελίδα ο χρήστης μπορεί να δει καθημερινά νέες αστρονομικές εικόνες από το NASA Apod API.
- Δυνατότητα εγγραφής και σύνδεσης: Στο menu της σελίδας υπάρχει το εικονίδιο του χρήστη όπου ο χρήστης μπορεί να το πατήσει για να κάνει εγγραφή ή σύνδεση εάν έχει λογαριασμό. Επίσης, υπάρχει η επιλογή για επαναφορά του κωδικού σε περίπτωση που ξεχάσει το κωδικό του λογαριασμού του.
- Σελίδα profile χρήστη: Αφού ο χρήστης συνδεθεί με επιτυχία στον λογαριασμό του, τότε πατώντας ξανά στο εικονίδιο του χρήστη θα μεταφερθεί στη σελίδα του προσωπικού του profile. Σε αυτή τη σελίδα μπορεί να δει διάφορες πληροφορίες

σχετικά με τον λογαριασμό του όπως το username, email, role αλλά και διάφορες επιλογές όπως την αλλαγή του κωδικού, τη διαγραφή του λογαριασμού του, και την επιλογή για αποσύνδεση. Επίσης, εάν ο χρήστης είναι moderator υπάρχει η επιλογή dashboard στην οποία μπορεί να δει διάφορες λεπτομέρειες σχετικά με την εφαρμογή.

- Σελίδα σελιδοδεικτών και λίστα αποθήκευσης εκτοξεύσεων (bookmarks - my launches): Αφού ο χρήστης συνδεθεί με επιτυχία μπορεί να αποθηκεύσει μια εκτόξευση έτσι ώστε να διαβάσει τις πληροφορίες σχετικά με αυτή αργότερα. Επίσης, μπορεί να δημιουργεί σελιδοδείκτες με τα ονόματα της αρεσκείας του, έτσι ώστε να διατηρεί τις εκτοξεύσεις που τον ενδιαφέρουν οργανωμένες.

3.3 Βασικές Βιβλιοθήκες που Χρησιμοποιήθηκαν

Σε αυτή την ενότητα αναλύονται οι βιβλιοθήκες τρίτου μέρους που χρησιμοποιήθηκαν και αποτελούν ζωτικής σημασίας για την υλοποίηση του frontend.

3.3.1 Βιβλιοθήκη React TanStack

Εικόνα 7: Λογότυπο React TanStack [18].

Η βιβλιοθήκη TanStack Query (πρώην γνωστή ως React Query) είναι μία ισχυρή βιβλιοθήκη για τη διαχείριση της κατάστασης του διακομιστή (server state) σε web εφαρμογές. Απλοποιεί τις διαδικασίες της ανάσυρσης (fetching), του μηχανισμού caching, του συγχρονισμού αλλά και της ανανέωσης των δεδομένων από τον διακομιστή, καθιστώντας την εφαρμογή ταχύτερη πιο συντηρήσιμη και επεκτάσιμη.

Τα περισσότερα web frameworks δεν προσφέρουν μια συγκεκριμένη λύση για τη λήψη ή την ενημέρωση δεδομένων, αναγκάζοντας τους προγραμματιστές είτε να φτιάξουν τις δικές τους λύσεις είτε να χρησιμοποιούν γενικές βιβλιοθήκες διαχείρισης κατάστασης. Ωστόσο, οι παραδοσιακές βιβλιοθήκες διαχείρισης κατάστασης είναι εξαιρετικές για τη διαχείριση client side state (κατάσταση πελάτη), αλλά δυσκολεύονται να χειριστούν την κατάστασης του διακομιστή (server state) η οποία έχει εντελώς διαφορετικά χαρακτηριστικά:

- Η κατάσταση του διακομιστή (server state) μπορεί να είναι αποθηκευμένη εξωτερικά, σε μια τοποθεσία που δεν ελέγχουμε όπως για παράδειγμα εξωτερικές πηγές δεδομένων (APIs).
- Απαιτεί ασύγχρονη επικοινωνία για λήψη και ενημέρωση των δεδομένων.
- Τα δεδομένα μπορεί να είναι κοινά και να αλλάζουν συνεχώς από άλλους χρήστες.
- Τα δεδομένα μπορεί να αλλάζουν συνέχεια, με αποτέλεσμα η εφαρμογή να έχει παλιά δεδομένα εάν δε γίνει σωστή διαχείριση της κατάστασης του διακομιστή (server state).

Στη συνέχεια αφού κατανοηθούν όλα τα παραπάνω προβλήματα τότε προκύπτει η ανάγκη για τους εξής μηχανισμούς:

- Μηχανισμός Caching.
- Διαγραφή πολλαπλών αιτήματα για τα ίδια δεδομένα.
- Μηχανισμός για την ενημέρωση των δεδομένων.
- Μηχανισμός για τον εντοπισμό των παλαιών δεδομένων
- Άμεση προβολή των αλλαγών στα δεδομένα.
- Βελτιστοποίησης ταχύτητας όπως pagination και lazy loading.
- Διαχείριση μνήμης με την απομάκρυνση των καταστάσεων του διακομιστή που δε χρησιμοποιούνται (Garbage Collection).

Η βιβλιοθήκη TanStack Query, βοηθά στην επίλυση όλων των παραπάνω προβλημάτων, αφού ενσωματώνει όλους τους παραπάνω μηχανισμούς και η εγκατάσταση της απαιτεί μηδενική παραμετροποίηση. Στη συνέχεια ακολουθούν κάποια παραδείγματα για την ανάσυρση δεδομένων με τη βιβλιοθήκη TanStack Query αλλά και παραδείγματα χωρίς αυτή τη βιβλιοθήκη.

```
function GitHubStats() {
  const [data, setData] = useState(null);
  const [loading, setLoading] = useState(true);

  useEffect(() => {
    fetch('https://api.github.com/repos/query')
      .then(response => response.JSON())
      .then(data => {
        setData(data);
        setLoading(false);
      });
  }, []);

  if (loading) {
    return <div>Loading...</div>;
  }
  return (
    <div>
      <h1>{data.name}</h1>
      <p>{data.description}</p>
    </div>
  );
}
```

Εικόνα 8: Αίτημα χωρίς TanStack Query.

```
function GitHubStats() {
  const {data, isLoading } = useQuery(
    ['githubStats'],
    () => fetch('https://api.github.com/repos).then(res => res.JSON())
  );

  if (isLoading) {
    return <div>Loading...</div>;
  }

  return (
    <div>
      <h1>{data.name}</h1>
      <p>{data.description}</p>
    </div>
  );
}
```

Εικόνα 9: Αίτημα με TanStack Query.

Σύμφωνα με τα παραπάνω αποσπάσματα κώδικα την Εικόνα 8 και την Εικόνα 9 συμπεραίνουμε πως ενώ έχουμε ίδιο αριθμό γραμμών κώδικα η ανάσυρση των δεδομένων με τη βιβλιοθήκη React Query μας προσφέρει περισσότερες δυνατότητες όπως το caching πράγμα το οποίο δε βλέπουμε στο πρώτο παράδειγμα. Επιπλέον, μπορούν να προστεθούν και άλλες λειτουργίες στο Hook useQuery, όπως η δυνατότητα αποθήκευσης προηγούμενων δεδομένων (KeepPreviousData), η δυνατότητα ενημέρωσης των δεδομένων όταν ο χρήστης επιστρέψει ξανά στην καρτέλα της εφαρμογής (refetchOnWindowFocus). Επίσης, υπάρχουν και άλλες λειτουργίες που

για την υλοποίηση τους μόνο με React θα ήταν αρκετά δύσκολη και θα απαιτούσε αρκετό χρόνο.

3.3.2 Βιβλιοθήκη Axios

Εικόνα 10: Λογότυπο Axios [19].

Η Βιβλιοθήκη Axios χρησιμοποιείται για την αποστολή HTTP αιτημάτων και είναι πιο ευέλικτη σε σχέση με την προεπιλεγμένη συνάρτηση fetch. Υποστηρίζει σημαντικά χαρακτηριστικά όπως interceptors, τα οποία επιτρέπουν την τροποποίηση αιτημάτων και απαντήσεων, καθώς και την αυτόματη μετατροπή των απαντήσεων σε JSON. Επίσης, διευκολύνει τη διαχείριση σφαλμάτων και την εκτέλεση παράλληλων αιτημάτων, παρέχοντας μια πιο ολοκληρωμένη λύση για τη διαχείριση HTTP αιτημάτων.

```
function GitHubStats() {
  const { data, isLoading } = useQuery(
    ['githubStats'],
    () => axios.get('https://api.github.com/repos').then(res => res.data)
  );

  if (isLoading) {
    return <div>Loading...</div>;
  }

  return (
    <div>
      <h1>{data.name}</h1>
      <p>{data.description}</p>
    </div>
  );
```

Εικόνα 11: Αίτημα με Axios.

Στο πλαίσιο της εργασίας, χρησιμοποιήθηκε σε συνδυασμό με τη βιβλιοθήκη TanStack Query, παρέχοντας έτσι ένα πανίσχυρο τρόπο για την ανάσυρση αλλά και την αποστολή δεδομένων προς τις REST υπηρεσίες.

3.3.3 Βιβλιοθήκη React Hook Form

Η βιβλιοθήκη React Hook Form χρησιμοποιείται για τη διαχείριση των φορμών με ευκολότερο και αποδοτικότερο τρόπο. Πιο συγκεκριμένα προσφέρει τα εξής πλεονεκτήματα:

- Εύκολη σύνδεση με τα πεδία εισόδου δεδομένων (input filed): Η βιβλιοθήκη παρέχει εύκολο τρόπο σύνδεσης των πεδίων της φόρμας χωρίς να χρειάζεται η διαχείριση πολλαπλών καταστάσεων.
- Περιορίζει των αριθμό των ανανεώσεων (re-renders) της διεπαφής χρήστη (UI): Διαχειρίζεται την κατάσταση της φόρμας χωρίς να προκαλεί περιττές ανανεώσεις ολόκληρης της διεπαφής χρήστη (UI). Αλλά μόνο συγκεκριμένα κομμάτια της φόρμας που έχουν αλλάξει τα δεδομένα τους.
- Ευκολότερη επικύρωση των δεδομένων: Προσφέρει ενσωματωμένο μηχανισμό επικύρωσης των δεδομένων της φόρμας, απλοποιώντας τη διαδικασία αφού δεν απαιτείται να υλοποιηθεί εξολοκλήρου από τον προγραμματιστή.
- Μικρό μέγεθος βιβλιοθήκης: Η βιβλιοθήκη React Hook Form έχει περιορισμένο μέγεθος και δεν εξαρτάτε από άλλες βιβλιοθήκες.
- Δυνατότητα δημιουργίας ανεξάρτητων components: Η κατάσταση της φόρμας είναι ανεξάρτητη από αλλά components της εφαρμογής. Έτσι, καθιστά τον κώδικα της φόρμας επαναχρησιμοποιήσιμο σε πολλαπλά σημεία της εφαρμογής (reusable component).

Στη συνέχεια βλέπουμε ένα παράδειγμα φόρμας το οποίο υλοποιήθηκε με τη βιβλιοθήκη React Hook Form:

```
function UserForm() {
  const {register, handleSubmit, formState: { errors }} = useForm();

  // Υποβολή δεδομένων φόρμας
  const onSubmit = (data) => {
    console.log(data);
  };

  return (
    <form onSubmit={handleSubmit(onSubmit)}>
      <div>
        <label htmlFor="name">Όνομα:</label>
        <input
          id="name"
          {...register('name', { required: 'Το όνομα είναι υποχρεωτικό' })}
        />
        {errors.name && <p>{errors.name.message}</p>}
      </div>
      <div>
        <label htmlFor="email">Ηλεκτρονικό Ταχυδρομείο:</label>
        <input
          id="email"
          type="email"
          {...register('email', {
            required: 'Η διεύθυνση ηλεκτρονικού ταχυδρομείο είναι υποχρεωτική',
            pattern: {
```

```
              value: /^[A-Z0-9._%+-]+@[A-Z0-9.-]+\.[A-Z]{2,}$/i,
              message: 'Μη έγκυρη διεύθυνση διεύθυνση ηλεκτρονικού ταχυδρομείο'
            }
          })}
        />
        {errors.email && <p>{errors.email.message}</p>}
      </div>
      <button type="submit">Υποβολή</button>
    </form>
  );
}
```

Εικόνα 12: Ορισμός φόρμας με την βιβλιοθήκη React Hook Form.

- Το Hook useForm: Χρησιμοποιείται για την πρόσβαση στα δεδομένα της φόρμας και στις λειτουργίες της.
- Η συνάρτηση register: Η συνάρτηση register συνδέει τα πεδία της φόρμας με τη βιβλιοθήκη και τα επικυρώνει.
- Η συνάρτηση handleSubmit: Η συνάρτηση χειρίζεται την υποβολή της φόρμας και καλεί τη συνάρτηση onSubmit όταν η φόρμα είναι έγκυρη.
- Κατάσταση formState.errors: Ελέγχει για τυχόν σφάλματα (errors) και τα εμφανίζει δίπλα στα πεδία αν υπάρχουν.

Χωρίς τη βιβλιοθήκη React Hook Form, κάθε πεδίο της φόρμας θα χρειαζόταν να διαχειριστούμε την κατάσταση του αλλά και την επικύρωση του μέσω πολλών useState hooks. Η βιβλιοθήκη καθιστά αυτόν τον κώδικα πιο καθαρό και εύκολο στην κατανόηση, ενώ παράλληλα βελτιώνει την απόδοση.

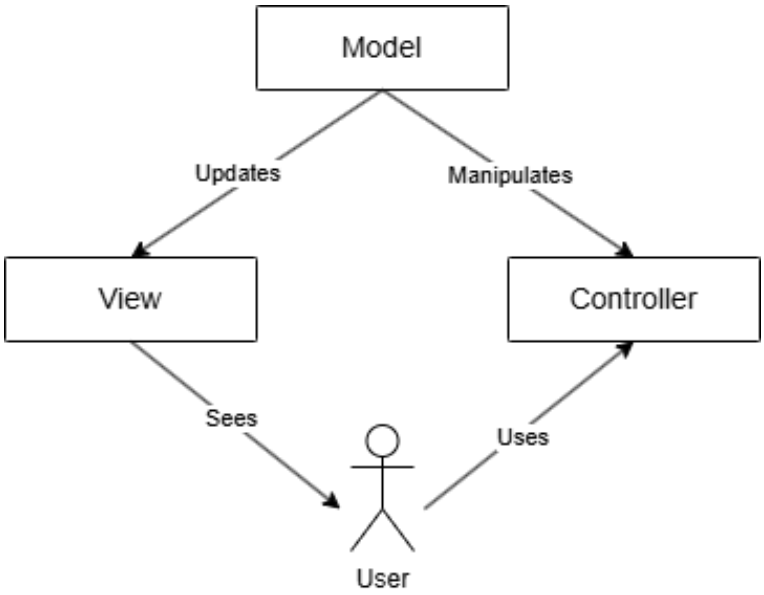
4 Ανάλυση και Υλοποίηση Backend

Σε αυτή την ενότητα αναλύεται η αρχιτεκτονική του backend. Αρχικά περιγράφονται τα πρότυπα της αρχιτεκτονικής, τα διαγράμματα της αρχιτεκτονικής και επιπλέον μηχανισμοί που υλοποιήθηκαν.

4.1 Ανάλυση Αρχιτεκτονικής

Σε αυτή την υποενότητα αναλύεται η αρχιτεκτονική του Spring Boot. Αρχικά παρουσιάζεται το πρότυπο σχεδίασης λογισμικού Model View Controller (MVC). Επιπλέον, παρουσιάζονται τα Framework που το υποστηρίζουν και ποιο επιλέχθηκε για την υλοποίηση του backend και αναλύεται η αρχιτεκτονική του.

4.1.1 Μοντέλο Αρχιτεκτονικής Model View Controller



Εικόνα 13: Μοντέλο αρχιτεκτονικής Model View Controller (MVC)

Το Model-View-Controller (MVC) είναι ένα θεωρητικό πρότυπο σχεδίασης λογισμικού το οποίο οργανώνει τη λογική μιας εφαρμογής σε διακριτά επίπεδα (layers), καθένα εκ των οποίων εκτελεί ένα συγκεκριμένο σύνολο λειτουργιών. Επιπλέον, τα επίπεδα αυτά συνεργάζονται μεταξύ τους ώστε να διασφαλιστεί η ομαλή λειτουργία του λογισμικού και ο διαχωρισμός αυτός συμβάλλει καθοριστικά στη βελτίωση της συντηρησιμότητας,

της αναγνωσιμότητας και της επεκτασιμότητας του κώδικα. Τα επίπεδα αναλύονται ως εξής:

- Model (Μοντέλο): Το μοντέλο είναι υπεύθυνο για τη διαχείριση της λογικής των δεδομένων.
- View (Όψη): Η όψη είναι υπεύθυνη για την απεικόνιση των δεδομένων στον τελικό χρήστη.
- Controller (Ελεγκτής): Ο ελεγκτής λειτουργεί ως ενδιάμεσος και αλληλοεπιδρά με το χρήστη μέσω της όψης και του μοντέλου.

4.1.2 Υποστηριζόμενα Frameworks σε Java

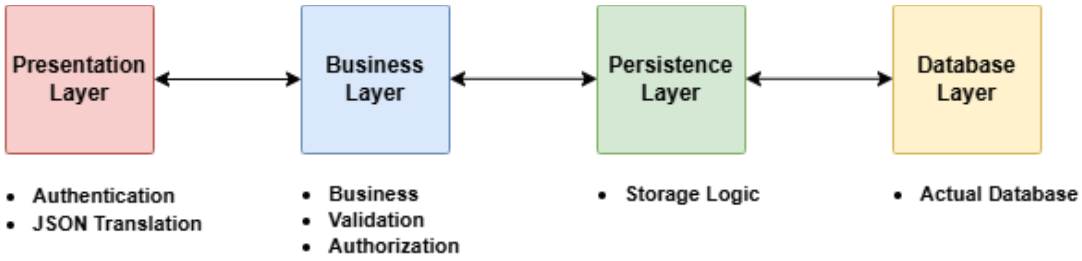
Εικόνα 14: Λογότυπο Spring Boot [20].

Υπάρχουν αρκετά frameworks που υποστηρίζουν την αρχιτεκτονική Model View Controller (MVC), όπως Spring Boot 3.0, Quarkus και JakartaEE και πολλά αλλά. Ωστόσο, για την υλοποίηση του backend επιλέχθηκε το framework Spring Boot για τους εξής λόγους:

- Απλοποίηση παραμετροποίησης (Auto-configuration): Το Spring Boot μειώνει δραστικά την ανάγκη για χειροκίνητη ρύθμιση (configuration), καθώς παρέχει έτοιμες προεπιλεγμένες ρυθμίσεις για συχνά χρησιμοποιούμενα modules.
- Ενσωματωμένος web server: Περιλαμβάνει ενσωματωμένους application servers όπως Tomcat ή Jetty, επιτρέποντας την εκτέλεση της εφαρμογής ως απλό JAR αρχείο, χωρίς να απαιτείται εξωτερική παραμετροποίηση ενός application server.
- Υποστήριξη REST APIs: Παρέχει υποστήριξη για την ανάπτυξη RESTful web services διευκολύνοντας την επικοινωνία με frontend εφαρμογές.
- Πλούσιο οικοσύστημα και κοινότητα: Το Spring Boot υποστηρίζεται από μια τεράστια κοινότητα έχει πλούσιο documentation και πληθώρα βιβλιοθηκών, γεγονός που διευκολύνει την ανάπτυξη και την επίλυση προβλημάτων.

- Επεκτασιμότητα και ευελιξία: Η εφαρμογή μπορεί εύκολα να επεκταθεί με επιπρόσθετα modules όπως το Spring Security, Spring Data και Spring Batch, χωρίς να απαιτούνται ριζικές αλλαγές στον κώδικα.
- Συμβατότητα με μοντέρνες αρχιτεκτονικές: Υποστηρίζει σύγχρονες αρχιτεκτονικές όπως microservices, REST APIs, και reactive programming, γεγονός που το καθιστά ευέλικτη επιλογή για μελλοντικές επεκτάσεις.

4.1.3 Αρχιτεκτονική του Spring Boot



Εικόνα 15: Τέσσερα επίπεδα οργάνωσης κώδικα.

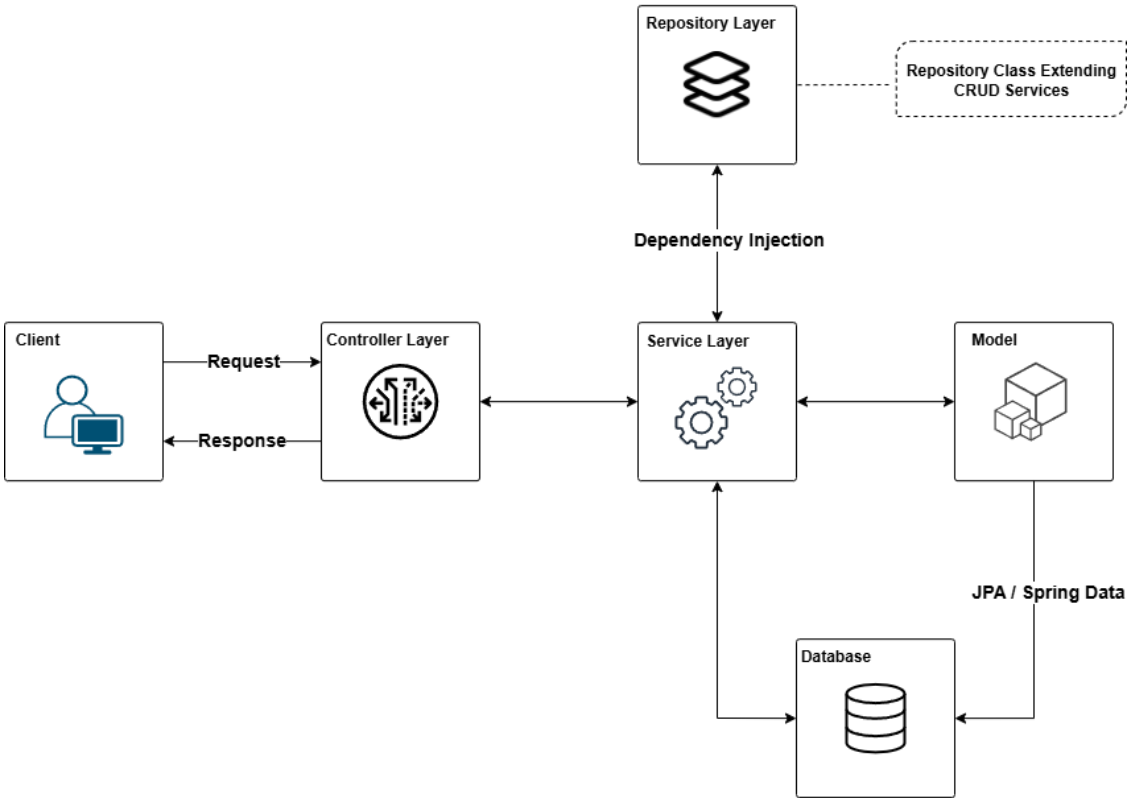
Πριν προχωρήσουμε στην ανάλυση της ροής της αρχιτεκτονικής του Spring Boot πρέπει να αναλυθούν τα τέσσερα επίπεδα οργάνωσης του κώδικα. Αναλυτικότερα, αποτελείται από τα επίπεδα:

- Presentation Layer: Το Presentation Layer αποτελεί το ανώτερο επίπεδο της αρχιτεκτονικής του Spring Boot. Αποτελείται κυρίως από REST controllers τα οποία είναι υπεύθυνα για τη διαχείριση HTTP αιτημάτων. Επίσης, επικυρώνει τα δεδομένα και είναι υπεύθυνο για τη σειριοποίηση (serialization) και την αποσειριοποίηση (deserialization) του JSON αιτήματος. Έπειτα, αφού ολοκληρωθεί η επεξεργασία του αιτήματος το προωθεί στο Business Layer.
- Business Layer: Το Business Layer είναι υπεύθυνο για την υλοποίηση της βασικής λογικής του συστήματος. Αποτελείται από κλάσεις υπηρεσιών οι οποίες εκτελούν τις ενέργειες επεξεργασίας και επικύρωσης των δεδομένων, διαχειρίζονται την αυθεντικοποίηση (authentication) και την εξουσιοδότηση (authorization) των χρηστών μέσω της βιβλιοθήκη Spring Security και διαχειρίζονται τις συναλλαγές εφαρμόζοντας το annotation @Transactional. Τέλος, αλληλεπιδρούν με το Persistence Layer τόσο για την αποθήκευση όσο και την ανάσυρση των δεδομένων.
- Persistence Layer: Το Persistence Layer διαχειρίζεται συναλλαγές με τη βάση δεδομένων και τη λογική αποθήκευσης των δεδομένων. Αποτελείται από Repository κλάσεις που χρησιμοποιούν τις βιβλιοθήκες Spring Data JPA, Hibernate ή R2DBC για

την πρόσβαση στα δεδομένα. Αναλυτικότερα, αντιστοιχούν τα αντικείμενα της Java στους αντίστοιχους πίνακες της βάσης δεδομένων με τη χρήση του Object Relational Mapping (ORM) και πραγματοποιούν βασικές ενέργειες CRUD (Create, Read, Update, Delete).

- Database Layer: Το Database Layer περιέχει τη βάση δεδομένων όπου τα δεδομένα αποθηκεύονται.

Συνεπώς, σύμφωνα με την παραπάνω οργάνωση του κώδικα σε τέσσερα επίπεδα και το πρότυπο σχεδίασης λογισμικού Model View Controller (MVC) το οποίο αναλύθηκε στην υποενότητα 4.1.1 προκύπτει η αρχιτεκτονική του Spring Boot.



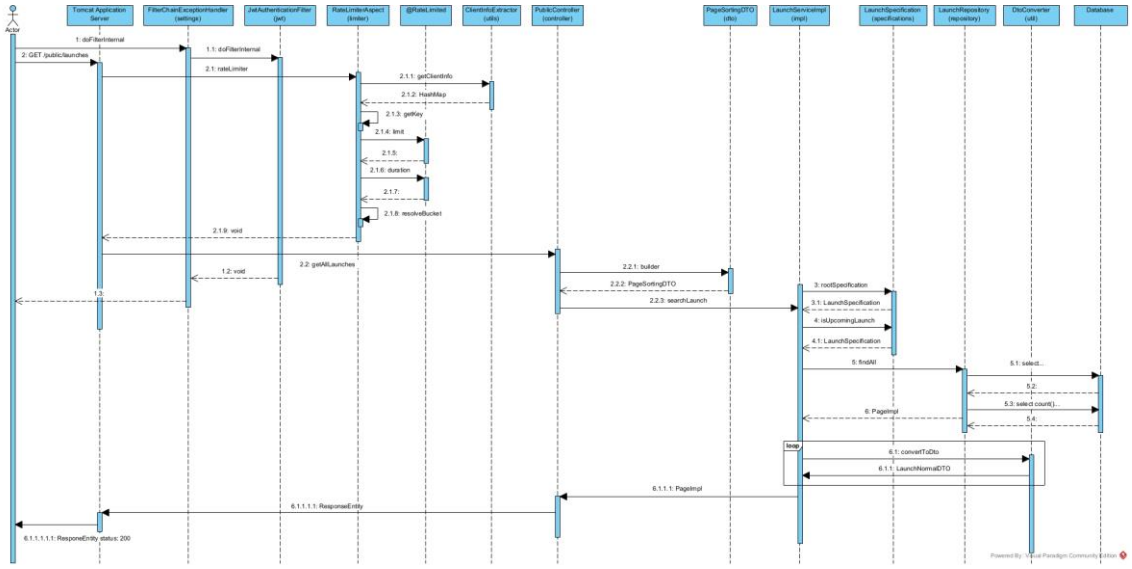
Εικόνα 16: Ροή αρχιτεκτονικής Spring Boot.

Αρχικά ο πελάτης (π.χ. React εφαρμογή) πραγματοποιεί ένα HTTP αίτημα προς στο backend. Τότε το backend διαχειρίζεται αυτό το αίτημα μέσω του Controller Layer και το ανακατευθύνει στην αντίστοιχη μέθοδο. Έπειτα, το Service Layer εκτελεί τη βασική λογική της εφαρμογής και επικοινωνεί με το Persistence Layer τόσο για την ανάσυρση όσο και για την τροποποίηση των δεδομένων.

Στη συνέχεια, το Persistence Layer με τη χρήση της βιβλιοθήκης Spring Data JPA που την υλοποιεί μια κλάση Repository αλληλεπιδρά με τη βάση δεδομένων για την

πραγματοποίηση CRUD (Create, Read, Update, Delete) ενεργειών. Τέλος, αφού ολοκληρωθεί η επεξεργασία του αιτήματος επιστρέφεται η απάντηση με τα δεδομένα με τη μορφή JSON.

4.2 Διάγραμμα Ακολουθίας Αναζήτηση Δεδομένων



Εικόνα 17: Διάγραμμα ακολουθίας αναζήτηση δεδομένων.

Στην Εικόνα 17 παρουσιάζεται το διάγραμμα ακολουθίας που αφορά την ροή ενός αιτήματος το οποίο σχετίζεται με την αναζήτηση εκτοξεύσεων. Η ροή του αιτήματος έχει ως εξής:

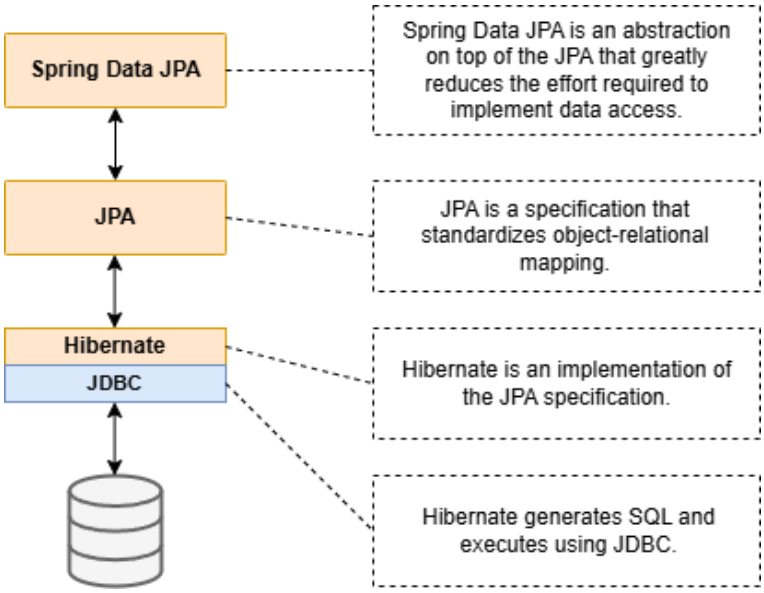
- FilterChainExceptionHandler: Εκτελείται σε κάθε αίτημα και αναλαμβάνει τις εξαιρέσεις (Exceptions) που μπορεί να προκύψουν στο επίπεδο κάποιου φίλτρου όπως το JwtAuthenticationFilter που θα αναλυθεί στη συνέχεια.
- JwtAuthenticationFilter: Εκτελείται μετά το FilterChainExceptionHandler και ελέγχει εάν απαιτείται αυθεντικοποίηση χρήστη (Authentication) για την ολοκλήρωση του αιτήματος.
- RateLimiter: Περιορίζει τον αριθμό τον αιτημάτων που γίνονται από κάποια διεύθυνση χρήστη (IP Address) ανά λεπτό. Για παράδειγμα, ο PublicController που επιστρέφει τις διαστημικές εκτοξεύσεις, στολίζεται με το annotation RateLimited και περιορίζεται σε εκατό αιτήματα (request) ανά λεπτό.
- PageSortingDTO: Αποτελεί μια κλάση Data Transfer Object (DTO) γίνεται build από τον PublicController και περιέχει τους παραμέτρους αναζήτησης μιας εκτόξευσης που ορίζονται από τον χρήστη.

- LaunchServiceImpl: Αποτελεί μια υπηρεσία με όλες τις μεθόδους αναζήτησης εκτοξεύσεων, όπως τη μέθοδο searchLaunch για την αναζήτηση δεδομένων με βάση κάποια κριτήρια τα οποίο περιέχει το αντικείμενο PageSortingDTO και δέχεται ως παράμετρο. Επίσης, επιστρέφει σελιδοποιημένα (Paginated) αποτελέσματα τύπου Page και κάθε ένα από αυτά είναι τύπου του interface DTOEntity. Επιπλέον, τη μέθοδο getLaunchById η οποία επιστρέφει ένα Optional αντικείμενο τύπου DTOEntity με βάση το αναγνωριστικό κάποιας εκτόξευσης. Επίσης, είναι Optional πράγμα που σημαίνει ότι θα επιστραφεί μόνο ένα αποτέλεσμα αλλά μπορεί και να μην επιστραφεί και κανένα αποτέλεσμα.
- LaunchSpecification: Αποτελεί μία κλάση όπου κάθε μέθοδος που περιέχει είναι ένα κριτήριο αναζήτησης. Στο παραπάνω διάγραμμα βλέπουμε την κλάση LaunchSpecification όπου έχει τις μεθόδους rootSpecification και isUpcomingLaunch. Αφού επιστραφεί ένα αντικείμενο τύπου Specification<Class> όπου class ο τύπος της οντότητας του model δηλαδή τύπου Launch. Τότε, αυτό το αντικείμενο θα μπει ως παράμετρος στη μέθοδο finadAll του repository LaunchRepository και θα επιστραφούν σελιδοποιημένα (Paginated) αποτελέσματα τύπου Page και κάθε ένα από αυτά είναι τύπου της οντότητας Launch από τη βάση δεδομένων.

4.3 Επιπλέον Μηχανισμοί και Βιβλιοθήκες

Σε αυτή την υποενότητα παρουσιάζεται η βιβλιοθήκη Spring Data JPA, ο μηχανισμός Caching αλλά και οι καλές πρακτικές HATEOAS.

4.3.1 Ανάλυση της Βιβλιοθήκης Spring Data JPA



Εικόνα 18: Βιβλιοθήκη Spring Data JPA.

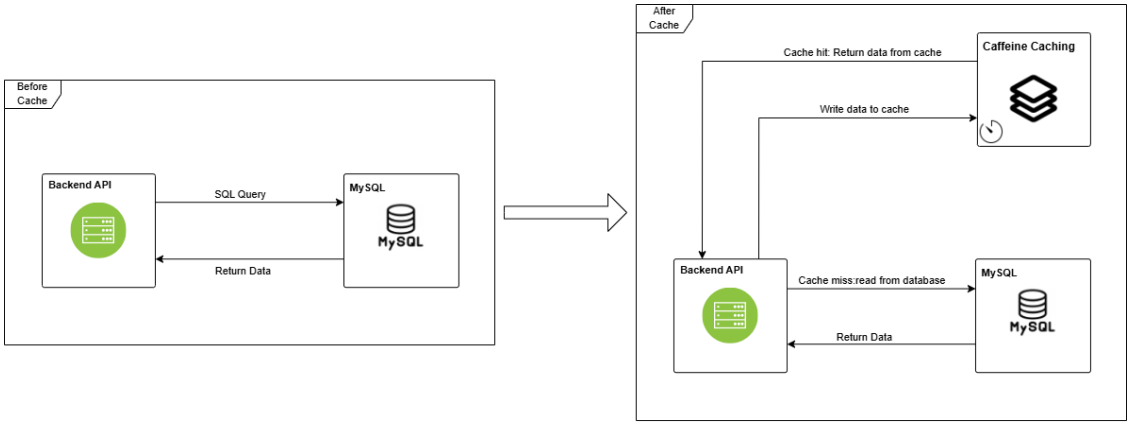
Η βιβλιοθήκη Spring Data JPA αποτελεί κομμάτι του Spring Data Project και διευκολύνει την αλληλεπίδραση με τις σχεσιακές βάσεις δεδομένων μέσω του JPA (Jakarta Persistence API) προσφέροντας έτοιμες μεθόδους για τις βασικές CRUD ενέργειες. Επιπλέον, ένα από τα βασικότερα πλεονεκτήματα είναι η ευκολία με την οποία μπορούν να υλοποιηθούν πολύπλοκες λειτουργίες με λιγότερο boilerplate κώδικα, βελτιώνοντας έτσι την αναγνωσιμότητα και συντηρησιμότητα της εφαρμογής.

Ένα ακόμα ισχυρό χαρακτηριστικό του Spring Data JPA είναι το Specification API, το οποίο επιτρέπει τη δυναμική κατασκευή ερωτημάτων βάσει κριτηρίων. Το API αυτό βασίζεται στο πρότυπο Specification Design Pattern και αξιοποιεί την κλάση Specification<T> για τη σύνθεση των φίλτρων. Μέσω αυτής της προσέγγισης, μπορούν να δημιουργηθούν πολύπλοκα και δυναμικά ερωτήματα ανάλογα με τα κριτήρια που επιλέγει ο χρήστης, χωρίς την ανάγκη δημιουργίας διαφορετικών repository μεθόδων για κάθε συνδυασμό παραμέτρων.

Στο πλαίσιο της εργασίας, η βιβλιοθήκη Spring Data JPA χρησιμοποιήθηκε για τη δημιουργία Repositories των οντοτήτων που αντιστοιχούν σε πίνακες της βάσης δεδομένων, διευκολύνοντας την επιστροφή των δεδομένων και τις βασικές CRUD ενέργειες. Σε συνδυασμό με τη βιβλιοθήκη Hibernate, επετεύχθη βελτιστοποίηση στον ορισμό των JPQL (Java Persistence Query Language) ερωτημάτων, καθώς και αποφυγή του προβλήματος των N+1 ερωτημάτων, το οποίο συχνά οδηγεί σε σημαντικές καθυστερήσεις όταν υπάρχουν σχέσεις μεταξύ των οντοτήτων.

Επιπλέον, όπως αναφέρεται και στις λειτουργικές απαιτήσεις της ενότητας δύο, ο χρήστης πρέπει να εφαρμόζει φίλτρα για εξατομικευμένη αναζήτηση αποτελεσμάτων. Για την υλοποίηση αυτής της απαίτησης, η χρήση του Specification API αποδείχθηκε καθοριστική, επιτρέποντας τη δυναμική και ευέλικτη σύνθεση των ερωτημάτων. Με τον τρόπο αυτό, η εφαρμογή απέκτησε μεγαλύτερη αποδοτικότητα και καλύτερη εμπειρία χρήσης, επιτρέποντας εύκολη και αποδοτική διαχείριση πολύπλοκων κριτηρίων αναζήτησης.

4.3.2 Ανάλυση της Βιβλιοθήκης Caffeine Caching



Εικόνα 19: Λειτουργία μηχανισμού Caching.

Ο μηχανισμός Caching είναι μια διαδικασία η οποία αποθηκεύει τα αποτελέσματα ενός αιτήματος (request) σε διαφορετικό μέρος από αυτό το οποίο επιστράφηκαν στην αρχή. Έτσι ώστε να μην τρέχει κάθε φορά ο κώδικας που μπορεί να είναι χρονοβόρος και να καταναλώνει πόρους. Για παράδειγμα, εάν το αίτημα εκτελείται συχνά και απαιτεί να επιστραφούν τα δεδομένα από τη βάση δεδομένων, τότε τα αποτελέσματα θα πρέπει να γίνουν Caching διότι ισχύουν τα εξής:

- Γλιτώνουμε τον χρόνο επικοινωνίας: Η επικοινωνία με τη βάση δεδομένων είναι χρονοβόρα.
- Γλιτώνουμε κόστος: Γλιτώνουμε κόστος αφού τα περισσότερα συστήματα διαχείρισης βάσεων δεδομένων (ΣΔΒΔ) στο cloud (π.χ. AWS RDS, Google Cloud SQL) χρεώνουν και με βάση το φόρτο στην ανταλλαγή δεδομένων.
- Γλιτώνουμε το χρόνο εκτέλεσης του κώδικα: Γλιτώνουμε το χρόνο που απαιτεί ο κώδικάς για να τρέξει, διότι η απάντηση στο αίτημα μπορεί να απαιτεί πολλά ερωτήματα (SQL Queries) ή σύνθετες υπολογιστικές ενέργειες.

Στην ενότητα δύο αναφέρθηκε πως η υπηρεσία API του backend θα πρέπει να έχει χρόνους απόκρισης κάτω 1 second. Για το λόγο αυτό χρησιμοποιείται η βιβλιοθήκη Caffeine για την υλοποίηση του μηχανισμού In-Memory Caching. Η οποία χρησιμοποιείται για την αποθήκευση των αποτελεσμάτων των συχνότερων ερωτημάτων προς τη βάση δεδομένων. Έτσι, ο χρόνος απόκρισης του backend μετά την πρώτη εκτέλεση του ερωτήματος κυμαίνεται μεταξύ 600 – 800 ms.

4.3.3 Περιγραφή Καλών Πρακτικών HATEOAS

Η πρακτική Hypermedia as the Engine of Application State (HATEOAS) αποτελεί τρόπος σχεδιασμού REST υπηρεσιών και οι πελάτες της υπηρεσίας μπορούν να αλληλοεπιδρούν με αυτή χρησιμοποιώντας υπερσυνδέσμους που αφορούν τις οντότητες που επιστρέφει.

Οι κανόνες HATEOAS εξασφαλίζουν ότι οι χρήστες δε χρειάζεται να έχουν γνώση για την υποδομή της υπηρεσίας πέρα από τον αρχικό υπερσύνδεσμο (URL). Αντίθετα, η υπηρεσία παρέχει τις σχετικές ενέργειες μέσω υπερσυνδέσμων. Έτσι, αυτοί οι σύνδεσμοι καθοδηγούν τον πελάτη σχετικά με τις διαθέσιμες ενέργειες και τον τρόπο εκτέλεσής τους, γεγονός που συμβάλλει στη διατήρηση μιας ενιαίας διεπαφής και μειώνει τη στενή σύζευξη μεταξύ πελάτη και διακομιστή. Αυτοί οι σύνδεσμοι μπορούν να αφορούν τα εξής:

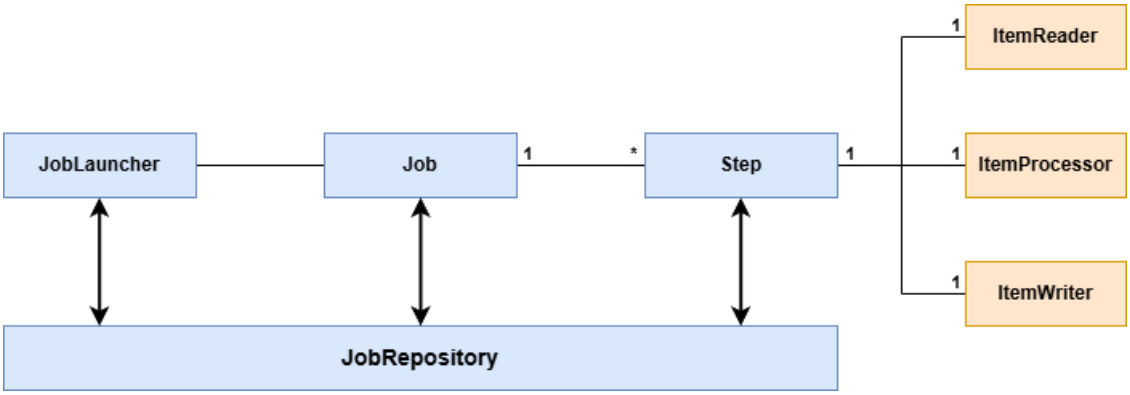
- Σύνδεσμοι που δείχνουν σε άλλους πόρους: Για παράδειγμα μπορεί να έχουμε μία εκτόξευση και να υπάρχει σύνδεσμος για τον σχετικό οργανισμό.
- Φόρμες: Όταν ο χρήστης προσθέτει μια εκτόξευση σε μια λίστα μπορεί να υπάρχουν σύνδεσμοι για το πως μπορεί να διαγράψει αυτή την εκτόξευση από τη λίστα του.
- Αναζήτηση: Μπορεί να περιέχει οδηγίες για το πως μπορεί να γίνει αναζήτηση για τις εκτοξεύσεις.
- Σχετικοί Σύνδεσμοι: Περιέχει σχετικούς συνδέσμους (π.χ. επόμενες και προηγούμενες εκτοξεύσεις)

4.4 Υλοποίηση μηχανισμού Extract Transform Load

Σε αυτή την υποενότητα αναλύεται ο μηχανισμός της εξαγωγής, μετασχηματισμού και φόρτωσης των δεδομένων στη βάση δεδομένων. Επιπλέον, παρουσιάζεται η βιβλιοθήκη

με την οποία γίνεται η ανάσυρση των δεδομένων αλλά και η βάση δεδομένων που χρησιμοποιήθηκε.

4.4.1 Ανάλυση της Βιβλιοθήκης Spring Batch Processing



Εικόνα 20: Αρχιτεκτονική της βιβλιοθήκης Spring Batch.

Η βιβλιοθήκη Spring Batch αποτελεί κομμάτι του Spring Framework και χρησιμοποιείται για τη μαζική επεξεργασία δεδομένων (Batch Processing). Παρέχει επαναχρησιμοποιήσιμες μεθόδους οι οποίες είναι απαραίτητες για την επεξεργασία μεγάλου όγκου δεδομένων όπως το διάβασμα, την εγγραφή και τον προγραμματισμό των εργασιών της επεξεργασίας τους. Επίσης, τα κύρια χαρακτηριστικά του είναι τα εξής:

- Διαχείριση εργασιών (Job Management): Ορίζει και διαχειρίζεται τις εργασίες.
- Επεξεργασία σε κομμάτια (Chunk Processing): Επεξεργάζεται τα δεδομένα σε κομμάτια (Chunks) βελτιώνοντας την απόδοση μειώνοντας τη χρήση μνήμης.
- Διαχείριση συναλλαγών (Transaction Management): Διασφαλίζει την ακεραιότητα των δεδομένα μέσω των συναλλαγών.
- Διαχείριση αποτυχιών (Error Handling): Υλοποιεί στρατηγικές retry και skip για τη διαχείριση των αποτυχιών.
- Παρακολούθηση εργασιών (Job Repository): Υλοποιεί μηχανισμό για την αποθήκευση του ιστορικού των εκτελεσμένων εργασιών στη βάση δεδομένων.

Επιπλέον, η βιβλιοθήκη Spring Batch αποτελείται από τα εξής κύρια κομμάτια:

1. Εργασία (Job): Μια εργασία αντιπροσωπεύει μια διαδικασία μαζικής επεξεργασίας (batch process), η οποία αποτελείται από ένα ή περισσότερα Βήματα (Steps). Κάθε εργασία μπορεί να ρυθμιστεί ώστε να εκτελείται μία φορά ή επαναλαμβανόμενα, ανάλογα με τις ανάγκες της εφαρμογής.

2. Βήμα (Step): Το βήμα είναι ένα επιμέρους στάδιο της εργασίας. Υλοποιεί τη λογική επεξεργασίας των δεδομένων, η οποία συνήθως περιλαμβάνει, ανάγνωση δεδομένων, επεξεργασία δεδομένων (π.χ. φιλτράρισμα ή μετασχηματισμός) αλλά και αποθήκευση των αποτελεσμάτων.
3. Φόρτωση δεδομένων (ItemReader): Υπεύθυνο για την ανάγνωση δεδομένων από κάποια πηγή, όπως ένα αρχείο, μια βάση δεδομένων ή ένα API. Επεξεργασία δεδομένων (ItemProcessor): Εκτελεί τη διαδικασία της επεξεργασίας των δεδομένων. Μπορεί να εφαρμόσει κανόνες φιλτραρίσματος, μετασχηματισμού ή επικύρωσης.
4. Εγγραφή δεδομένων (ItemWriter): Υπεύθυνο για την εγγραφή των επεξεργασμένων δεδομένων στην έξοδο, όπως σε μια βάση, ένα αρχείο ή άλλο σύστημα. Στην παρούσα εργασία η βιβλιοθήκη Spring Batch εφαρμόστηκε εν μέρει δηλαδή εφαρμόστηκαν οι μηχανισμοί για την καταγραφή (JPA Repository) τον ορισμό των εργασιών (Job) αλλά και τα βήματα εκτέλεσης (Steps). Δεν εφαρμόστηκαν όμως οι μέθοδοι για τη φόρτωση, επεξεργασία και γράψιμο δεδομένων διότι απαιτείται περεταίρω μελέτη για την υλοποίηση τους.

4.4.2 Ανάλυση της Βιβλιοθήκης WebClient

Εικόνα 21: Λογότυπο βιβλιοθήκης WebClient [24].

Η βιβλιοθήκη WebClient χρησιμοποιείται για την εκτέλεση HTTP αιτημάτων σε εξωτερικές υπηρεσίες. Αποτελεί κομμάτι του πακέτου Spring WebFlux και δημιουργήθηκε για να αντικαταστήσει την παλαιότερη βιβλιοθήκη RestTemplate. Επίσης, υποστηρίζει reactive programming που σημαίνει ότι δεν μπλοκάρει το νήμα κατά την εκτέλεση αιτημάτων. Επιπλέον, αξίζει να σημειωθεί ότι παρέχει υποστήριξη τόσο για σύγχρονες (Sychronized) όσο και ασύγχρονες (Asyncronized) λειτουργίες καθιστώντας τη βιβλιοθήκη κατάλληλη και για εφαρμογές που εκτελούνται σε Servlet Stack.

Στην παρούσα εργασία χρησιμοποιήθηκε για την πραγματοποίηση αιτημάτων σε εξωτερική υπηρεσία και συγκεκριμένα στο The Space Devs API για την ανάκτηση των απαραίτητων δεδομένων.

4.4.3 Σχεσιακή Βάση Δεδομένων MySQL

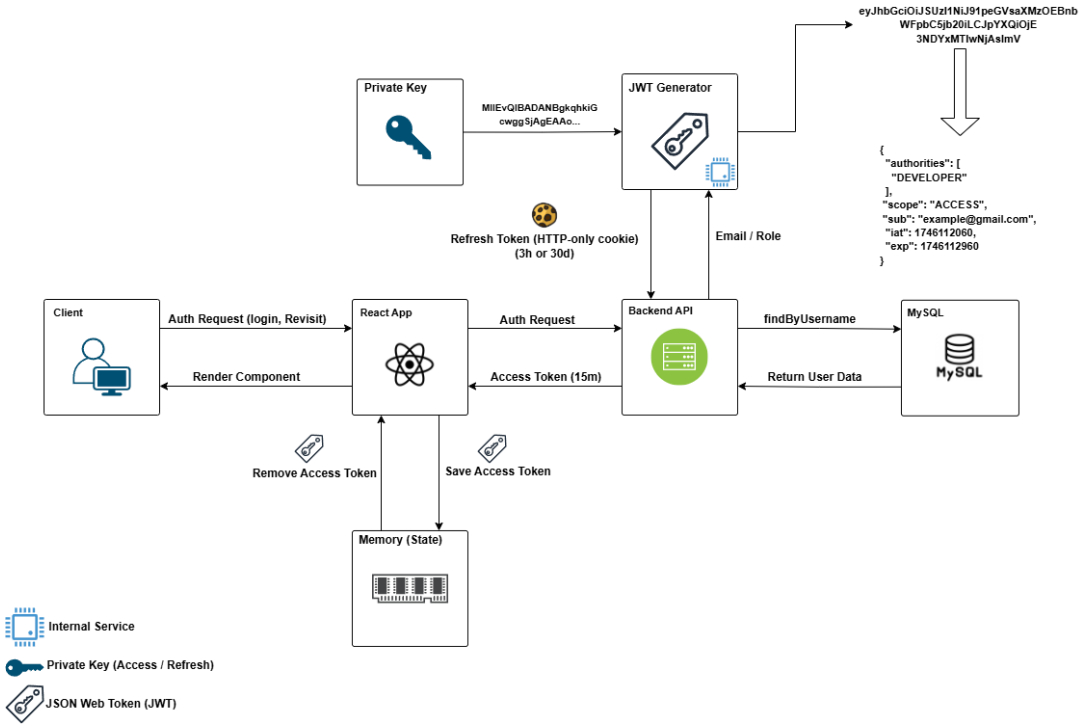
Εικόνα 22: Λογότυπο MySQL [25].

Όπως είδαμε στην υποενότητα 4.3.1 χρησιμοποιήθηκε η βιβλιοθήκη Batch Processing για την εξαγωγή των δεδομένων τον μετασχηματισμό και τη φόρτωση σε μια βάση δεδομένων. Στο πλαίσιο της εργασίας χρησιμοποιήθηκε η σχεσιακή βάση δεδομένων MySQL για την εξυπηρέτηση των αναγκών της εφαρμογής. Επιπλέον, αναπτύχθηκε σύμφωνα με το σχεσιακό μοντέλο μιας και τα δεδομένα μπορούσαν να σπάσουν σε υποπίνακες ακολουθώντας του κανόνες κανονικοποίησης των σχέσεων.

5 Ανάλυση και Υλοποίηση Μηχανισμού Αυθεντικοποίησης

Σε αυτή την ενότητα αναλύεται ο μηχανισμός αυθεντικοποίησης χρήστη. Αρχικά παρουσιάζεται υλοποίηση του κομματιού που αφορά το frontend και στη συνέχεια το κομμάτι που αφορά το backend.

5.1 Ανάλυση Μηχανισμού Αυθεντικοποίησης



Εικόνα 23: Αρχιτεκτονική μηχανισμού αυθεντικοποίησης.

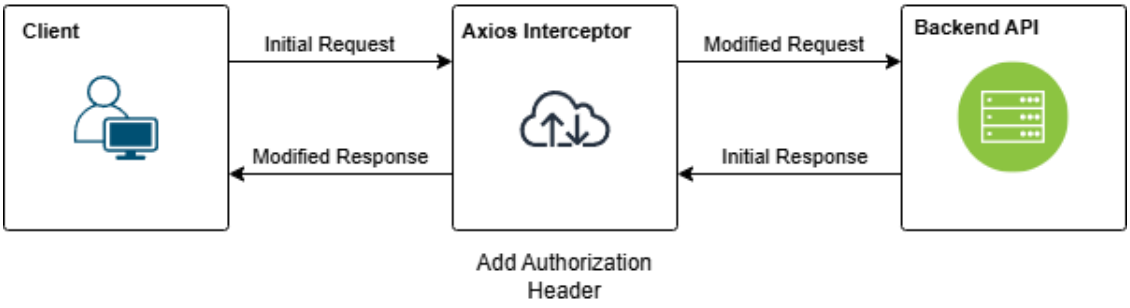
Στην Εικόνα 23 παρουσιάζεται η υλοποίηση του μηχανισμού αυθεντικοποίησης χρήστη. Ακολουθεί μια αρχική ανάλυση της ροής του διαγράμματος, ενώ περισσότερες λεπτομερείς σχετικά με τη λειτουργία και την υλοποίηση του κάθε μηχανισμού, τόσο του frontend όσο και του backend, θα παρουσιαστούν στις επόμενες υποενότητες. Η ροή των αιτημάτων έχει ως εξής:

- Πελάτης (Client): Αλληλοεπιδρά με τη React εφαρμογή, στην περίπτωση αυτή πραγματοποιεί ενέργειες που αφορούν την αυθεντικοποίηση του. Για παράδειγμα, σύνδεση ή επανασύνδεση όταν επιστρέφει ξανά στην εφαρμογή.
- Εφαρμογή (React): Η εφαρμογή React μετατρέπει την αλληλεπίδραση του χρήστη σε αιτήματα προς το backend και κάθε φορά που πραγματοποιείται κάποιο αίτημα προσθέτει τα κατάλληλα διαπιστευτήρια (credentials).
- Εφαρμογή (Backend API): Η εφαρμογή Spring Boot (Backend API) επεξεργάζεται τα αιτήματα (requests) και αν τα διαπιστευτήρια του χρήστη είναι σωστά, τότε παράγει δύο JSON Web Tokens. Αναλυτικότερα, παράγονται το refresh Token το οποίο αποθηκεύεται ως Http-only cookie στον browser και το access Token το οποίο επιστρέφεται στη React εφαρμογή μέσω της απάντησης (response) του Backend API.

5.2 Υλοποίηση Μηχανισμού Αυθεντικοποίησης στο Frontend

Σε αυτή την υποενότητα παρουσιάζεται ο μηχανισμός αυθεντικοποίησης χρήστη από την πλευρά του frontend.

5.2.1 Χρήση Axios Interceptors



Εικόνα 24: Ροή αιτημάτων με τα Axios Interceptors.

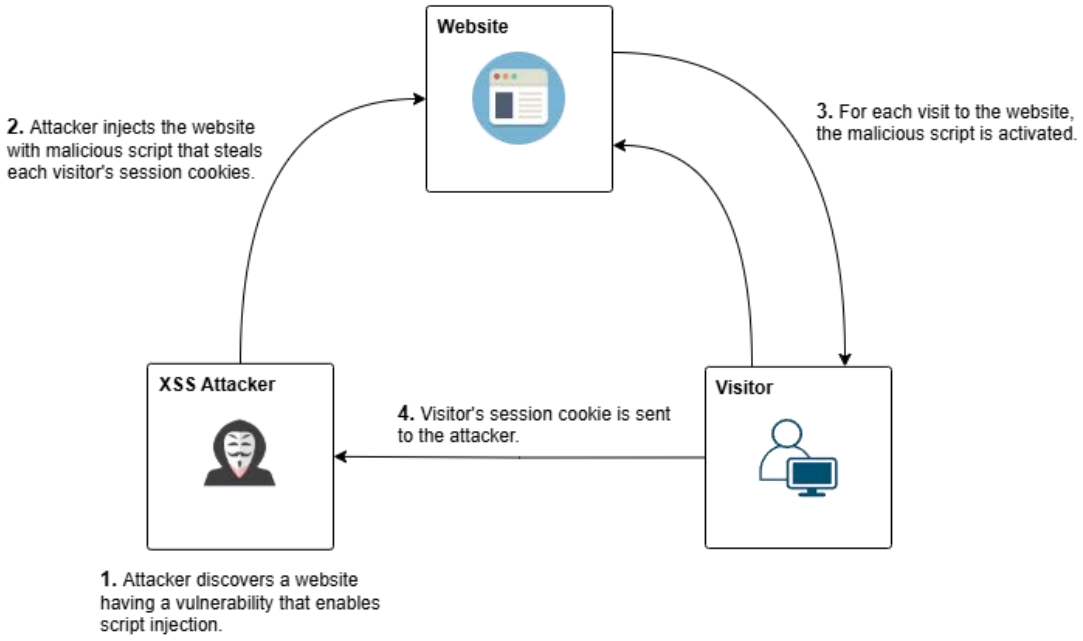
Στην ενότητα τρία παρουσιάστηκε η βιβλιοθήκη Axios η οποία αποτελεί ένας μοντέρνος τρόπος για την πραγματοποίηση αιτημάτων σε εξωτερικές υπηρεσίες. Σε αυτή την υποενότητα παρουσιάζονται τα Axios Interceptors τα οποία χρησιμοποιούνται για την αναχαίτηση των αιτημάτων έτσι ώστε σε κάθε αίτημα να προστίθενται τα κατάλληλα διαπιστευτήρια χρήστη.

Σύμφωνα με την Εικόνα 24 τα Interceptors χρησιμοποιούνται για την προσθήκη του access token αλλά και του CSRF Token σε κάθε αίτημα. Παράλληλα, υλοποιείται μηχανισμός για την ανανέωση του access token όταν αυτό λήξει. Η διαδικασία αυτή επιτυγχάνεται μέσω μιας ουράς αποτυχημένων αιτημάτων. Όταν το access token λήξει, τα αποτυχημένα αιτήματα προστίθενται στην ουρά και μετά την ανανέωσή του, εκτελούνται εκ νέου με το νέο access token, χωρίς ο τελικός χρήστης να αντιληφθεί κάτι.

5.2.2 Αντιμετώπιση Απειλών Cross Site Scripting (XSS)

Υπάρχουν αρκετοί τρόποι για την αποθήκευση ενός session στον browser όπως το Local Storage, Session Storage και indexedDB. Οι συγκεκριμένοι τρόποι αποθήκευσης στον browser είναι προσβάσιμοι με τη γλώσσα JavaScript με αποτέλεσμα να είναι ευάλωτοι σε επιθέσεις Cross Site Scripting (XSS). Έτσι, για το λόγο αυτό το refresh token αποθηκεύεται ως ένα Http-only cookie διότι έχει μεγαλύτερη διάρκεια ζωής από τρείς ώρες μέχρι τριάντα ημέρες.

Αντίθετα το access token, έχει μικρότερη διάρκεια ζωής περίπου δεκαπέντε λεπτά το οποίο όπως προ αναφέραμε επιστρέφεται στη απάντηση (response) του Backend API. Επίσης, αποθηκεύεται στη μνήμη της συσκευής του πελάτη ως κατάσταση (state) χρησιμοποιώντας το Hook useState της React. Αναλυτικότερα, επιλέχθηκε αυτή η προσέγγιση διότι αποτελεί ο ασφαλέστερος τρόπος για την αποθήκευση ενός ευαίσθητου Token διότι περιέχει προσωπικά στοιχεία του χρήστη. Με αυτόν τον τρόπο, η πρόσβαση στη μνήμη της συσκευής του χρήστη καθίσταται πολύ πιο δύσκολη για έναν κακόβουλο χρήστη, μειώνοντας έτσι την πιθανότητα υποκλοπής του access token. Επίσης, στο browser Storage, η υποκλοπή μπορεί να γίνει ευκολότερα μέσω της εκτέλεσης ενός απλού JavaScript κώδικα.



Εικόνα 25: Αντιμετώπιση απειλών Cross Site Scripting (XSS).

Αφού αναλύθηκαν οι λόγοι για τους οποίους δεν αποθηκεύουμε τα tokens στο storage API του browser, παρουσιάζεται ένα παράδειγμα επίθεσης Cross Site Scripting (XSS) σύμφωνα με την Εικόνα 25 το οποίο έχει σκοπό την υποκλοπή του session του χρήστη και συγκεκριμένα του access token.

Αρχικά ο επιτιθέμενος ανακαλύπτει κάποια σελίδα η οποία είναι ευαίσθητή σε επιθέσεις XSS. Έπειτα, εμφυτεύει ένα script το οποίο του επιτρέπει να κλέβει session cookies, στη δική μας περίπτωση όλα τα cookies είναι τύπου http-only και δεν είναι προσβάσιμα από τη JavaScript. Οπότε για το συγκεκριμένο σενάριο θα υποθέσουμε ότι προσπαθεί να υποκλέψει το session από το localStorage του browser. Στη συνέχεια, για κάθε αίτημα που πραγματοποιεί ο επισκέπτης της σελίδας τότε ενεργοποιείται και αυτό το script το οποίο επικοινωνεί με το backend του επιτιθέμενου και τελικά το session token του χρήστη καταλήγει στον κακόβουλο χρήστη.

Παρακάτω παρουσιάζεται η μορφή ενός script το οποίο υποκλέπτει το session από το localStorage του browser.

```
<div style="padding: 10px; margin-top: 10px;">
    <h4>Subscribe to our Newsletter!</h4>
    <form onsubmit="
            event.preventDefault();

            fetch('https://example.com/steal', {
                method: 'POST',
                headers: {
                    'Content-Type': 'application/JSON'
                },
```

```
                body: JSON.stringify({
                    token: localStorage.getItem('access_token'),
                    email: this.email.value
                })
            }).then(response => {
                if (response.ok) {
                    alert('Thank you for subscribing!');
                }
            }).catch(error => {
                console.error('Error:', error);
            });
        "data-gtm-form-interact-id="0">
        <input type="email" name="email" placeholder="Enter your email" data-gtm-form-
interact-field-id="0">
        <button type="submit">Subscribe</button>
    </form>
</div>
```

Εικόνα 26: Επίθεση Cross Site Scripting (XSS)

Αναλυτικότερα παρουσιάζεται μια πλαστή φόρμα newsletter την οποία θα μπορούσε να ενσωματωθεί σε κάποιο μέρος της σελίδας και σε κάθε υποβολή της συγκεκριμένης φόρμας υποκλέπτει το access token από το localStorage και το αποστέλλει στο backend του επιτιθέμενου. Επίσης, στην περίπτωση μας ο παραπάνω κίνδυνος δεν υπάρχει αφού όπως προαναφέραμε το access token του χρήστη αποθηκεύεται στη μνήμη της συσκευής του και το refresh token αποθηκεύεται ως ένα http-only cookie. Επιπλέον, αυτές οι επιθέσεις είναι συχνό να μην είναι ορατές και να ενσωματώνονται σε κρυφά σημεία της εφαρμογής.

5.3 Ανάλυση Μηχανισμών Αυθεντικοποίησης

Σε αυτή την υποενότητα παρουσιάζονται οι μηχανισμοί αυθεντικοποίησης χρήστη έτσι ώστε να βοηθήσει στην κατανόηση των διαγραμμάτων που θα δούμε στην υποενότητα 5.4.

5.3.1 Μηχανισμός JSON Web Token (JWT)

Εικόνα 27: Δομή JSON Web Token (JWT).

Το JSON Web Token (JWT) αποτελεί ένας αυτόνομος τρόπο για τη μετάδοση πληροφοριών μεταξύ δύο ή περισσότερων συστημάτων ή χρηστών με τη μορφή ενός αντικειμένου JSON. Παράλληλα, οι πληροφορίες που περιέχουν τα JSON Web Token μπορούν να επαληθευτούν και να θεωρηθούν αξιόπιστες αφού φέρουν ψηφιακή υπογραφή και υπογράφονται με ένα ζεύγος κλειδιών το ιδιωτικό (Private) και το δημόσιο (Public) τα οποίο παράγονται με κάποιο αλγόριθμο όπως τον HMAC ή RSA και πολλούς άλλους. Αναλυτικότερα ένα JSON Web Token αποτελείται από τα εξής:

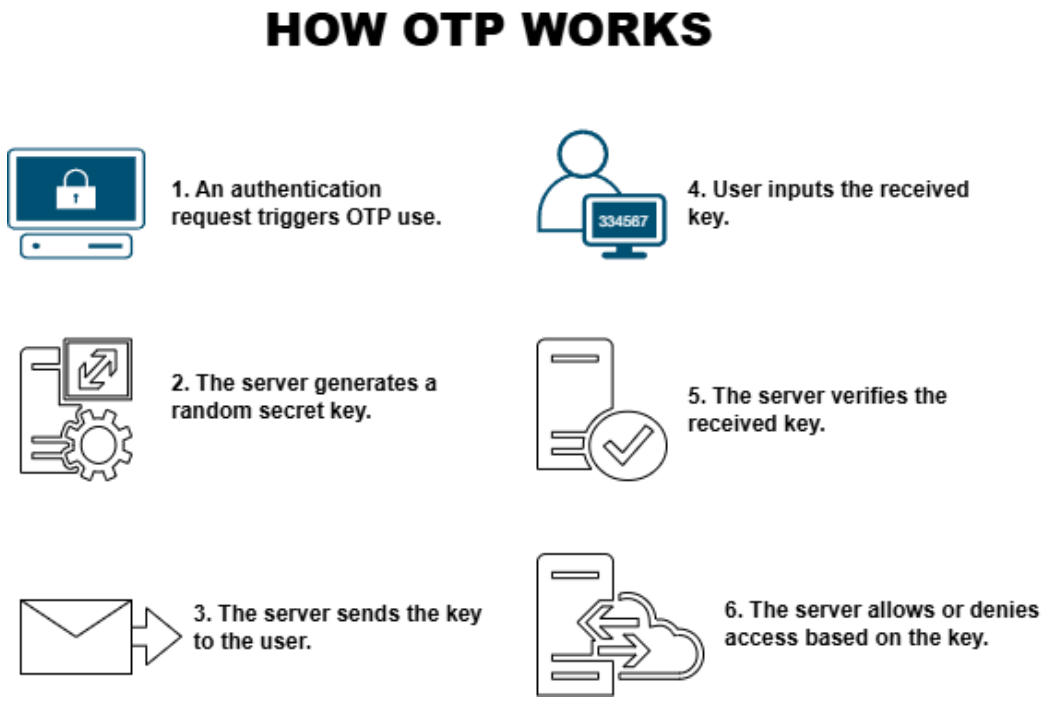
- Headers: Οι ρυθμίσεις του Header αποτελούνται από τον τύπο του token και το αλγόριθμο της ψηφιακής υπογραφής.
- Payload: Το Payload του token περιέχει τις πληροφορίες που φέρει γνωστές ως claims.
- Signature: Η ψηφιακή υπογραφή (signature) δημιουργείται συνδυάζοντας το κωδικοποιημένο header, το κωδικοποιημένο payload, ένα μυστικό κλειδί (secret) και τον αλγόριθμο που έχει οριστεί στο header. Ο ρόλος της υπογραφής είναι να επαληθεύει ότι το μήνυμα δεν έχει αλλοιωθεί κατά τη μεταφορά του. Επιπλέον, στην περίπτωση που το token έχει υπογραφεί με ιδιωτικό κλειδί, η υπογραφή μπορεί επίσης να επιβεβαιώσει ότι ο αποστολέας του JWT είναι πράγματι αυτός που ισχυρίζεται.

Έπειτα, συνδυάζοντας τα παραπάνω προκύπτει το ακόλουθο JSON Web Token.

Εικόνα 28: JSON Web Token (JWT) [27].

Στο πλαίσιο της εργασία χρησιμοποιήθηκαν δύο JSON Web Token το access token και το refresh token. Το access token έχει μικρότερη διάρκεια ζωής περίπου δεκαπέντε λεπτά, ενώ το refresh token έχει μεγαλύτερη διάρκεια ζωής μεταξύ τριών ωρών και τριάντα ημερών και χρησιμοποιείται για την ανανέωση του access token εφόσον αυτό λήξη. Επίσης, αξίζει να σημειωθεί πως οι ψηφιακές υπογραφές τους έχουν παραχθεί με διαφορετικά ζεύγη ιδιωτικών και δημόσιων κλειδιών.

5.3.2 Μηχανισμός One Time Password (OTP)



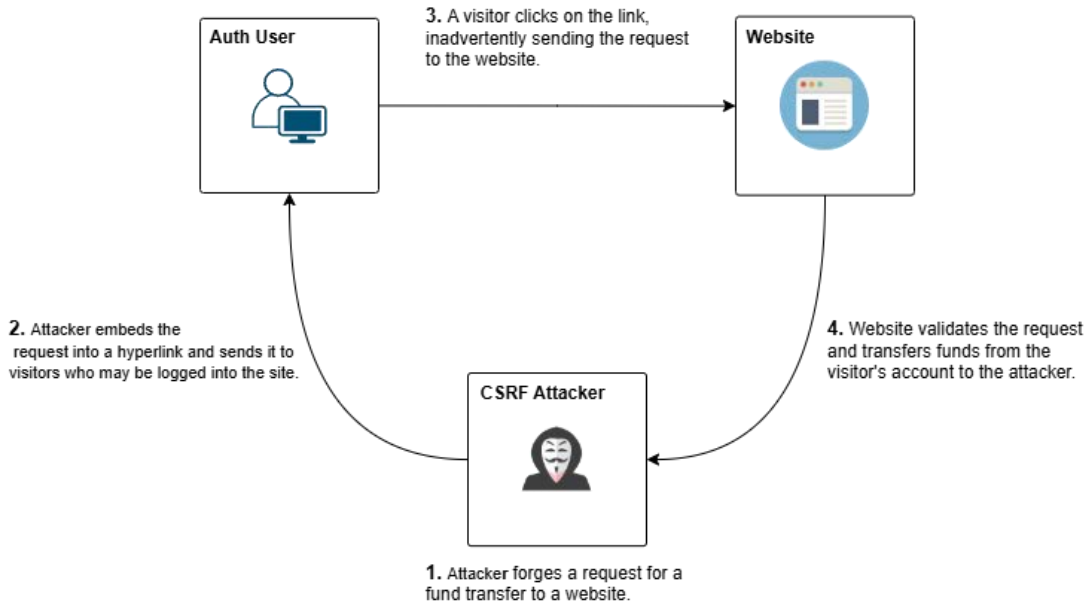
Εικόνα 29: Μηχανισμός One Time Password (OTP).

Ένα One Time Password (OTP) είναι ένας μοναδικός κωδικός ο οποίο παράγεται με ασφαλή τρόπο. Σε αντίθεση με τους παραδοσιακούς κωδικούς οι οποίοι παραμένουν σταθερή για μεγάλη χρονική διάρκεια, ένα One Time Password (OTP) δεν παραμένει σταθερό διότι μπορεί να έχει περιορισμένο χρόνο εξαργύρωσης και μπορεί να χρησιμοποιηθεί μόνο μια φορά. Έτσι, εάν κάποιος κακόβουλος χρήστης αποχτήσει αυτό

το κωδικό τότε δε θα μπορεί να το χρησιμοποιήσει διότι πιθανότατα θα έχει εξαργυρωθεί ή λήξη. Επίσης, αξίζει να σημειωθεί πως ένα OTP μπορεί να προστατέψει μια μην εξουσιοδοτημένη πρόσβαση σε ένα λογαριασμό ακόμη και αν ο κωδικός του λογαριασμού έχει διαρρεύσει.

Υπάρχουν αρκετοί τρόποι για την παραγωγή ενός One Time Password (OTP) όπως HMAC-based One-Time Password (HOTP) και Time-based One-Time Password (TOTP), παρόλο αυτά στο πλαίσιο της εργασίας χρησιμοποιήθηκε μια απλούστερη μέθοδος για την παραγωγή ενός OTP. Πιο συγκεκριμένα, για την παραγωγή του OTP χρησιμοποιήθηκε η κλάση SecureRandom της Java SE στην οποία ορίζεται ένας προκαθορισμένος αριθμός ψηφίων και το ζεύγος των αριθμών που παράγεται είναι δύσκολο να προβλεφθεί σε αντίθεση με την παραδοσιακή Random κλάση. Η υλοποίηση και η λειτουργία του μηχανισμού OTP αναλύεται στην υποενότητα 5.4.

5.3.3 Μηχανισμός Cross Site Request Forgery Token (CSRF)



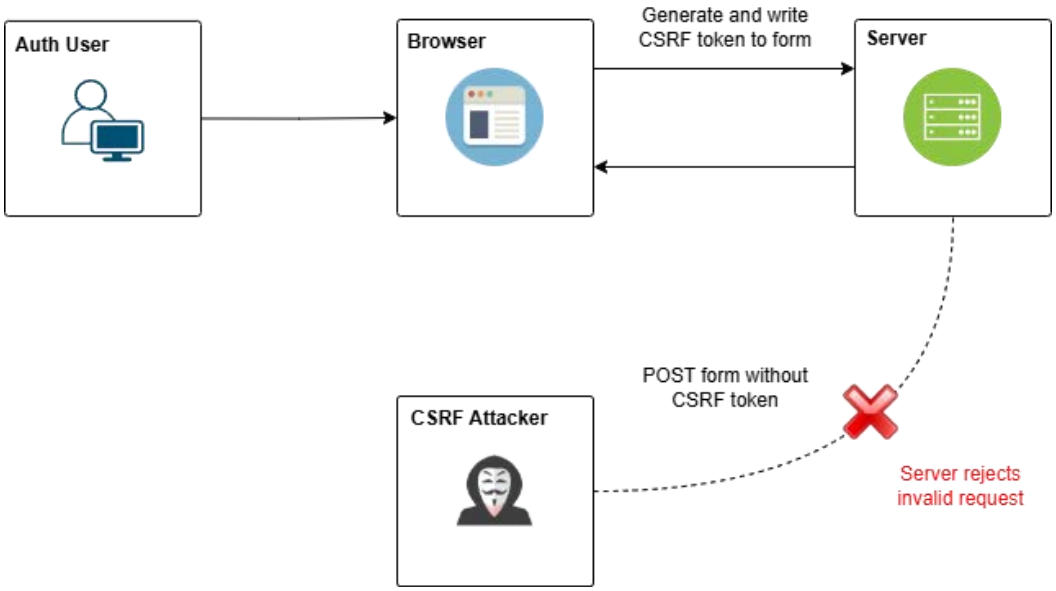
Εικόνα 30: Επίθεση Cross Site Request Forgery (CSRF).

Το Cross Site Request Forgery (CSRF) είναι ένα είδος επίθεσης όπου μην εξουσιοδοτημένες ενέργειες πραγματοποιούνται από αξιόπιστους χρήστες χωρίς τη θέλησή τους. Αυτό σημαίνει ότι ο κακόβουλος χρήστης εκμεταλλεύεται το γεγονός πως σε κάθε αίτημα ενός συνδεδεμένου χρήστη προστίθενται αυτόματα κάποια cookies τα οποία μπορούν να περιέχουν τα διαπιστευτήρια του για την πρόσβαση στην εφαρμογή.

```
<!-- Attempt to delete a user's account -->
<img src="https://backend.moonkeyeu.com/user/account/deactivate" />
```

Εικόνα 31: Παράδειγμα επίθεσης Cross Site Request Forgery (CSRF).

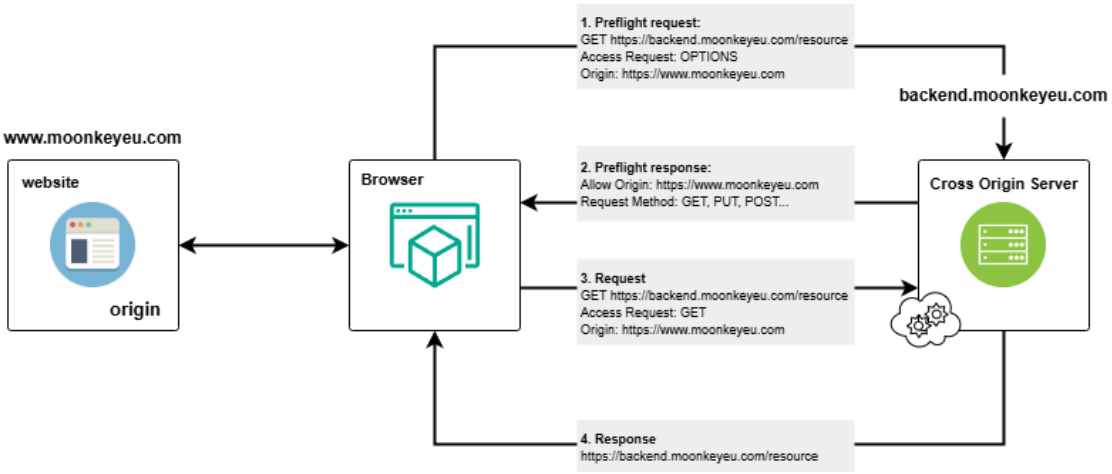
Παραπάνω βλέπουμε μία μορφή μιας CSRF επίθεσης, όταν ο χρήστης επισκεφτεί μια σελίδα που περιέχει το παραπάνω HTML στοιχείο τότε ο browser θα προσπαθήσει να κάνει ένα GET αίτημα στο παραπάνω URL. Παράλληλα, εάν ο χρήστης είναι συνδεδεμένος, ο browser θα παράσχει τα απαραίτητα cookies στο αίτημα και η προσπάθεια διαγραφής του λογαριασμού θα είναι επιτυχής αφού πρόκειται για μια ενέργεια από ένα αξιόπιστο χρήστη.



Εικόνα 32: Αντιμετώπιση επιθέσεων Cross Site Request Forgery (CSRF).

Έτσι, για την αντιμετώπιση του παραπάνω προβλήματος χρησιμοποιούνται CSRF tokens τα οποία παράγονται από τον ενσωματωμένο μηχανισμό του Spring Security. Αναλυτικότερα, προστίθεται σε κάθε φόρμα που υποβάλει ο χρήστης έτσι ώστε να καταλαβαίνει το backend ότι πρόκειται για αληθινό αίτημα και όχι κάποιο πλαστό από κάποιο κακόβουλο χρήστη. Στη συνέχεια, το CSRF token εξάγεται από τη φόρμα και προστίθεται στα headers του αιτήματος με τη βοήθεια του Axios Interceptor που αναλύθηκε στην υποενότητα 5.2.1.

5.3.4 Μηχανισμός Cross-Origin Resource Sharing Policy (CORS)



Εικόνα 33: Μηχανισμός Cross-Origin Resource Sharing Policy (CORS).

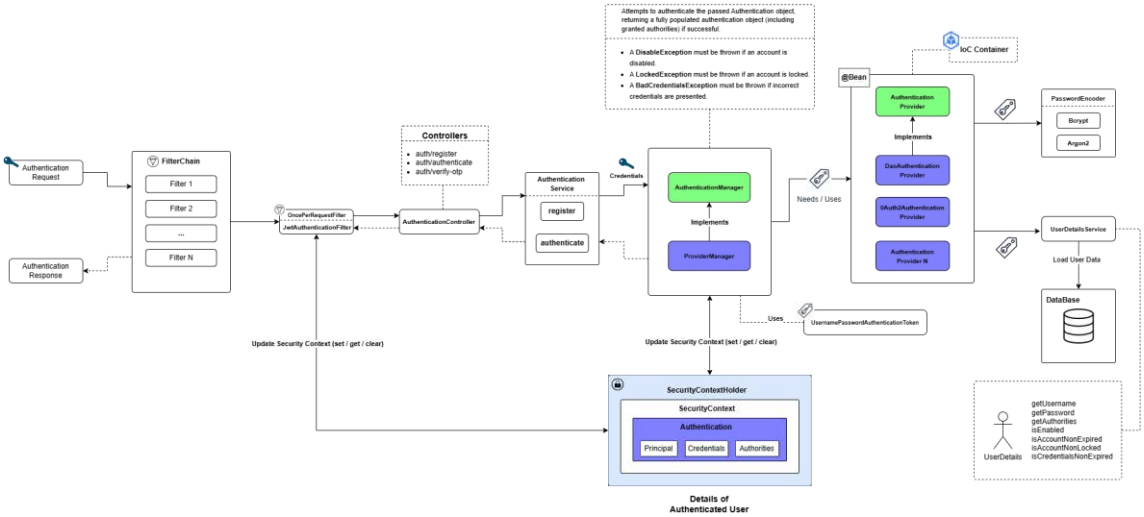
Το Cross-Origin Sharing (CORS) είναι ένας μηχανισμός ασφαλείας που βασίζεται στα HTTP-Headers και δίνει τη δυνατότητα σε ένα backend API να επιτρέπει ή να αποτρέπει ποιες προελεύσεις από domain, scheme ή port θα φορτώνουν πόρους από αυτό. Επιπλέον, ενσωματώνει μηχανισμό όπου οι browsers στέλνουν ένα εικονικό (preflight) αίτημα στον διακομιστή έτσι ώστε να διαπιστώσουν εάν έχουν πρόσβαση σε αυτό τον controller με ένα πραγματικό αίτημα. Επίσης, σε αυτό το εικονικό αίτημα προστίθενται η μέθοδος HTTP και τα headers που πρόκειται να προστεθούν στον πραγματικό αίτημα.

Στο πλαίσιο της εργασίας, τα CORS Policy προσαρμόστηκαν έτσι ώστε το backend API να λαμβάνει αιτήματα από το frontend μέσω των URL www.moonkeyeu.com και moonkeyeu.com. Επιπλέον, προστέθηκε παραμετροποίηση για τον περιορισμό των επιτρεπόμενων HTTP μεθόδων ανά endpoint, με στόχο την περαιτέρω ενίσχυση της ασφάλειας.’

5.4 Υλοποίηση Μηχανισμού Αυθεντικοποίησης στο Backend

Σε αυτή την υποενότητα παρουσιάζεται ο μηχανισμός αυθεντικοποίησης χρήστη από την πλευρά του backend.

5.4.1 Ανάλυση Αρχιτεκτονικής του Spring Security



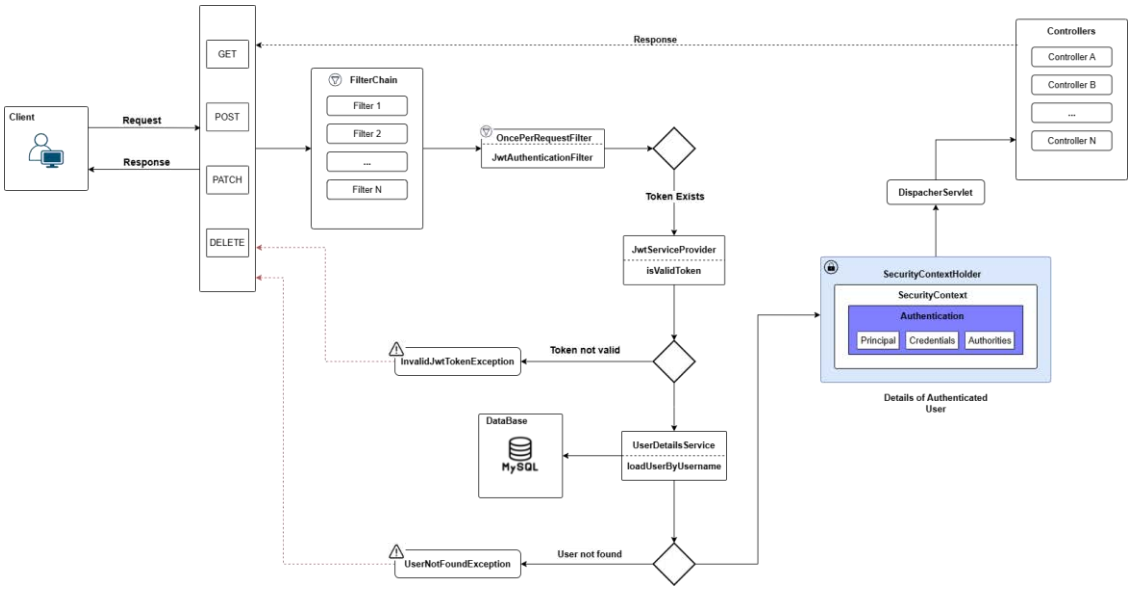
Εικόνα 34 Αρχιτεκτονική Spring Security.

Αφού αναλύσαμε την υλοποίηση του μηχανισμού αυθεντικοποίησης από την πλευρά του frontend, σε αυτή την ενότητα θα αναλυθεί η υλοποίηση του μηχανισμού από την πλευρά του backend. Στην Εικόνα 34 παρουσιάζεται η αρχιτεκτονική της βιβλιοθήκης Spring Security όπου χρησιμοποιήθηκε για την υλοποίηση μηχανισμού αυθεντικοποίησης. Η αρχιτεκτονική του αποτελείται από τα εξής:

- AuthenticationRequest: Αρχικά λαμβάνουμε ένα αίτημα με τα διαπιστευτήρια του χρήστη το username και το password.
- FilterChain: Όταν το αίτημα φτάσει στο backend εκτελεί μια αλυσίδα με φίλτρα όπου ένα από αυτά είναι και το JwtAutheticationFilter.
- JwtAutheticationFilter: Εάν το αίτημα απαιτεί πρόσβαση σε κάποιο controller που δε χρειάζεται αυθεντικοποίηση χρήστη, τότε δεν εκτελείται το φίλτρο και το αίτημα ανακατευθύνεται στο επόμενο φίλτρο. Αντίθετα, εάν το αίτημα απαιτεί πρόσβαση σε κάποιο controller που απαιτείται αυθεντικοποίηση χρήστη. Σε αυτή την περίπτωση ελέγχει εάν ο χρήστης έχει κάποιο ενεργό access JSON Web Token (JWT) έτσι ώστε να του επιτραπεί η πρόσβαση. Επίσης, τα στοιχεία του χρήστη αποθηκεύονται στο SecurityContextHolder.
- AuthenticationController: Στη συνέχεια, το αίτημα ανακατευθύνεται στον AuthenticationController ο οποίος έχει μια εξάρτηση (dependency) με την υπηρεσία AuthenticationService και περιέχει τις μεθόδους register και authenticate για σύνδεση και εγγραφή του χρήστη.

- AuthenticationService: Η υπηρεσία AuthenticationService αποτελείται από την εξάρτηση AuthenticationManager η οποία είναι μια διεπαφή (Interface). Επίσης, χρησιμοποιεί τη μέθοδο authenticate της κλάσης AuthenticationManager και χρησιμοποιείται για την επικύρωση των στοιχείων χρήστη. Εάν η επικύρωση αποτύχει τότε προκύπτουν οι εξαιρέσεις DisableException ή LockedException ή BadCredentialsException.
- UsernamePasswordAuthenticationToken: Δημιουργεί ένα αντικείμενο με τα διαπιστευτήρια ενός χρήστη και το δέχεται σαν παράμετρο η μέθοδος authenticate.
- ProviderManager: Η κλάση ProviderManager αποτελεί μια προεπιλεγμένη υλοποιεί της διεπαφής (Interface) AuthenticationManager. Επίσης, απαιτεί ένα Bean αντικείμενο τύπου AuthenticationProvider στο οποίο ορίζεται ο τρόπος με τον οποίο θα πραγματοποιείται η αυθεντικοποίηση του χρήστη.
- DaoAuthenticationProvider: Η κλάση DaoAuthenticationProvider υλοποιεί τη διεπαφή AuthenticationProvider και αποτελεί ένας τρόπος αυθεντικοποίηση χρήστη. Επίσης, απαιτεί να οριστούν τουλάχιστον δύο εξαρτήσεις για τη λειτουργία του, η πρώτη εξάρτηση είναι ο PasswordEnconder και χρησιμοποιείται για τον κατακερματισμό (hashing) των κωδικών των χρηστών. Η δεύτερη εξάρτηση είναι η υπηρεσία UserDetailsService η οποία περιέχει τη μέθοδο loadUserByUsername για την επιστροφή των δεδομένων χρήστη με βάση το username του από τη βάση δεδομένων.
- AuthenticationResponse: Επιστρέφεται το JSON Web Token αλλά όμως στο πλαίσιο της εργασίες επιστρέφεται το One Time Password Identifier.

Επομένως, η βιβλιοθήκη Spring Security περιέχει διάφορες υπηρεσίας οι οποίες βοηθούν στην υλοποίηση του μηχανισμού αυθεντικοποίησης χρήστη. Επιπλέον, οργανώνει το κώδικα σε κομμάτια έτσι ώστε η κάθε υπηρεσία να αναλαμβάνει το δικό της κομμάτι κατά τη διάρκεια της επικύρωσης των δεδομένων του χρήστη.

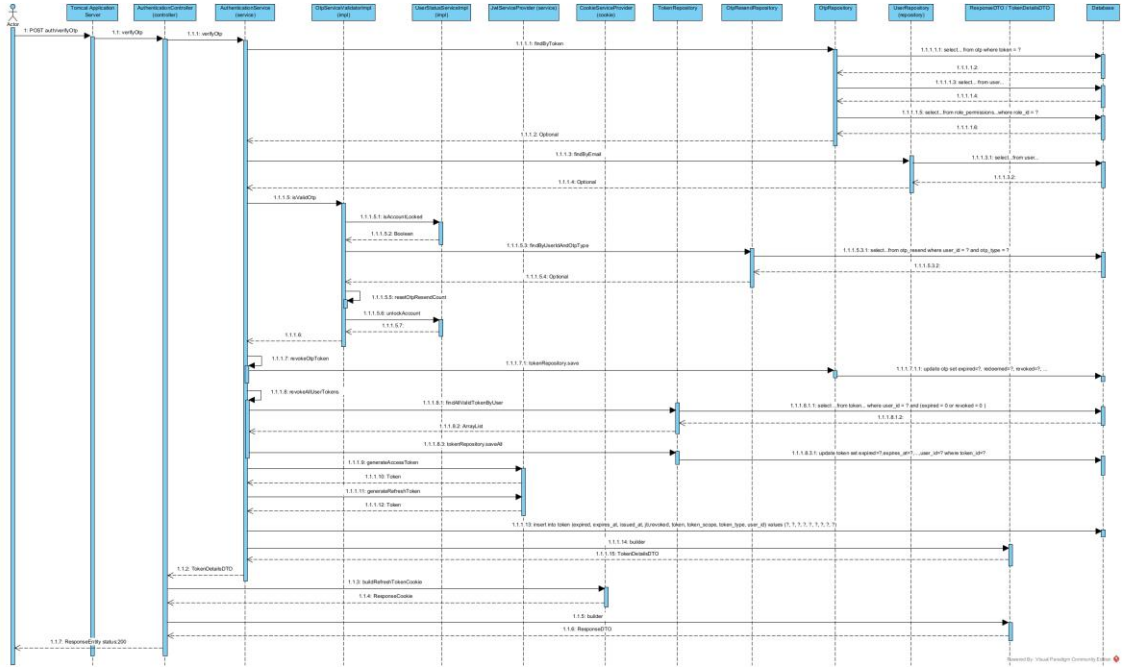


Εικόνα 35: Επικύρωση Access Token.

Στο παραπάνω διάγραμμα παρουσιάζεται η ροή επικύρωσης ενός access JSON Web Token (JWT) το οποίο το λαμβάνουμε στα headers του κάθε αιτήματος, δηλαδή αιτήματα που λαμβάνουμε από συνδεδεμένους χρήστες στην εφαρμογή. Αρχικά λαμβάνουμε ένα αίτημα από τον χρήστη με τα απαραίτητα διαπιστευτήρια στα headers. Έπειτα, το αίτημα εκτελεί το JwtAuthenticationFilter όπου ελέγχει εάν το JSON Web Token (JWT) υπάρχει στο Authentication του header. Εάν υπάρχει τότε εκτελείται η μέθοδος isValidToken της κλάσης JwtServiceProvider και ελέγχει εάν το JWT είναι σωστό. Σε περίπτωση που δεν είναι σωστό επιστρέφεται η εξαίρεση InvalidJWTTokenException.

Στη συνέχεια, εξάγεται το όνομα του χρήστη από το JSON Web Token (JWT) και το δέχεται σαν παράμετρο η μέθοδος loadUserByUsername όπου θα επιστρέψει τα δεδομένα του συγκεκριμένου χρήστη από την βάση δεδομένων. Σε περίπτωση που το όνομα του χρήστη δεν υπάρχει στη βάση δεδομένων τότε επιστρέφεται η εξαίρεση UserNotFoundException. Τέλος, τα δεδομένα του χρήστη αποθηκεύονται στο SecurityContextHolder και ανακατευθύνεται προς τον controller που απαιτούσε αυθεντικοποίηση χρήστη για την πρόσβαση σε αυτόν.

5.4.2 Διάγραμμα Ακολουθίας Μηχανισμού One Time Password



Εικόνα 36: Διάγραμμα ακολουθίας επικύρωση One Time Password (OTP).

Στην υποενότητα 5.3.1 αναλύσαμε την αρχιτεκτονική του Spring Security και παρουσιάστηκε μια πρώτη εικόνα της ροής των αιτημάτων. Σε αυτή την υποενότητα αναλύεται η υλοποίηση του μηχανισμού αυθεντικοποίηση στο πλαίσιο της εφαρμογής. Επιπλέον, για λόγους απλότητας θα υποθέσουμε ότι ο χρήστης έχει εισάγει τα διαπιστευτήρια του και του έχει αποσταλεί με επιτυχία στο email του το One Time Password (OTP). Ενώ παράλληλα, το backend του έχει επιστρέψει και το αντίστοιχο One Time Password Identifier στην απάντηση του.

Το τελικό στάδιο της διαδικασίας αυθεντικοποίησης, ξεκινάει αφού το backend λάβει ένα αίτημα (AuthenticationRequest) με τα διαπιστευτήρια του χρήστη με το One Time Password (OTP) και τον αντίστοιχο Identifier. Έπειτα ο AuthenticationController ανακατευθύνει το αίτημα (auth/verify-otp) στο αντίστοιχο endpoint έτσι ώστε να κληθεί η μέθοδος verifyOtp της εξάρτησης AuthenticationService και να ξεκινήσει η διαδικασία της επικύρωσης.

Η κλάση AuthenticationService αποτελείται από πολλαπλές εξαρτήσεις και καλεί τις μεθόδους αυτών των εξαρτήσεων έτσι ώστε να πραγματοποιήσει τους απαραίτητους ελέγχους. Πιο συγκεκριμένα, βρίσκει το One Time Password με βάση το Identifier και βρίσκει και το αντίστοιχο χρήστη. Στη συνέχεια, ελέγχει την ορθότητα του One Time Password (OTP) με τη μέθοδο isValidOtp και ανακαλεί όλα One Time Password αλλά

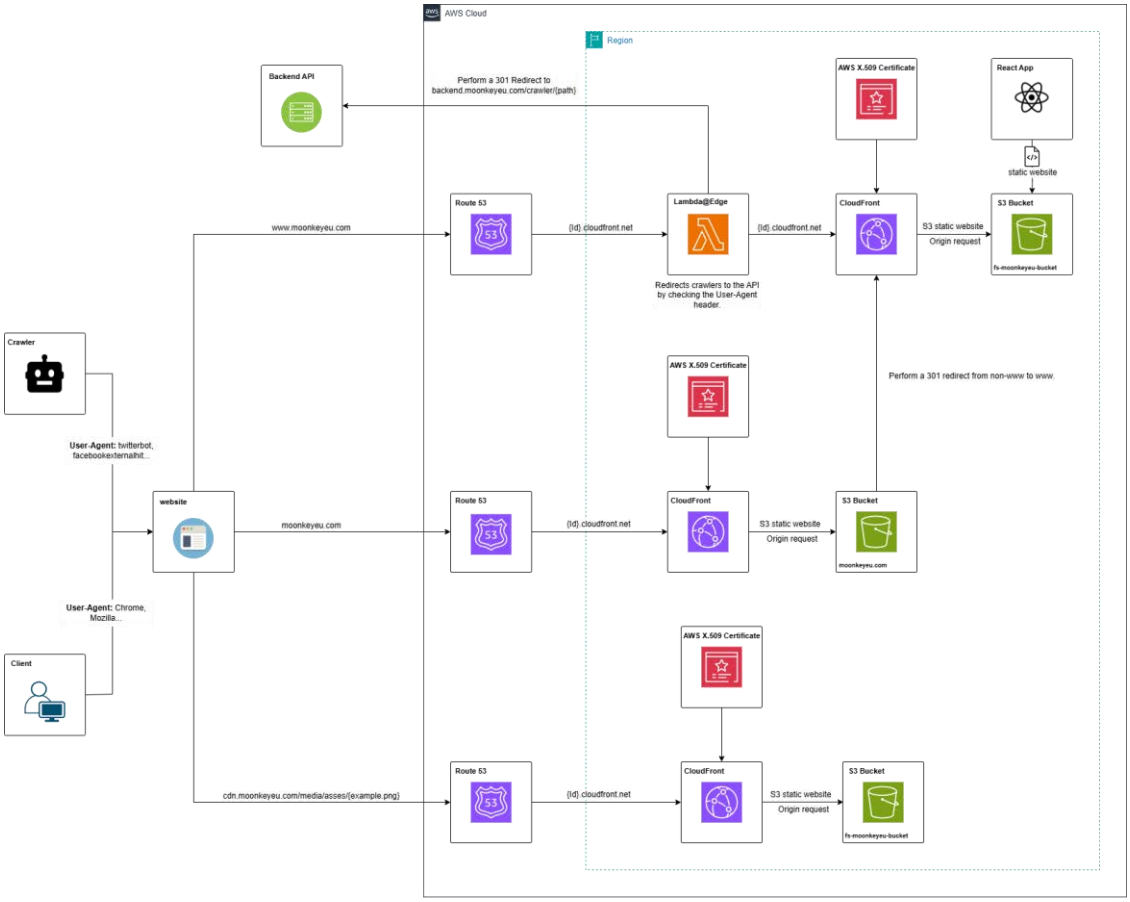
και τα JSON Web Tokens με τις μεθόδους revokeOtpToken και revokeAllUserTokens αντίστοιχα. Επιπλέον, δημιουργεί νέα access και refresh JSON Web Tokens και αποθηκεύει το refresh token στη βάση δεδομένων.

Τέλος, αφού ολοκληρωθεί η διαδικασία της επικύρωσης επιστρέφεται στον AuthenticationController ένα αντικείμενο TokenDetailsDTO το οποίο περιέχει το refresh και access tokens. Επίσης, δημιουργεί ένα cookie με το refresh token ενώ παράλληλα επιστρέφει στην απάντηση του το access token με το χρόνο διάρκειας.

6 Εγκατάσταση της Εφαρμογής στην Πλατφόρμα Amazon Web Services

Σε αυτή την ενότητα παρουσιάζεται η αρχιτεκτονική τόσο frontend όσο και του backend στο cloud. Αρχικά παρουσιάζεται το διάγραμμα της αρχιτεκτονικής του frontend και αναλύονται όλα τα εργαλεία του cloud που χρησιμοποιήθηκαν για την υλοποίηση του. Στη συνέχεια, παρουσιάζεται το διάγραμμα της αρχιτεκτονικής του backend και αναλύεται η διαδικασία της εγκατάστασης του στον Virtual Private Server (VPS).

6.1 Αρχιτεκτονική Frontend στο Cloud



Εικόνα 37: Αρχιτεκτονική frontend στο cloud.

Στην Εικόνα 37: Αρχιτεκτονική frontend στο cloud.παρουσιάζεται η αρχιτεκτονική του frontend στο cloud η οποία υλοποιήθηκε στην πλατφόρμα Amazon Web Services. Επίσης, ακολουθεί μια πρώτη ανάλυση της ροής του διαγράμματος και επιπρόσθετες πληροφορίες για τη λειτουργία και τον ρόλος της κάθε υπηρεσίας θα παρουσιαστούν στις επόμενες υποενότητες που ακολουθούν. Η ροή των αιτημάτων έχει ως εξής:

- Πελάτης (Client): Ο χρήστης κάνει ένα αίτημα HTTP(S) προς το domain της εφαρμογής (π.χ. www.moonkeyeu.com).
- Σύστημα ονομάτων τομέων (Route 53 DNS): Ανακατευθύνει τα αιτήματα του domain στην υπηρεσία CloudFront.
- Υπηρεσία CloudFront (CDN): Διανέμει το περιεχόμενο το οποίο αποθηκεύει στη cache και ανακτά νέο περιεχόμενο από τον αποθηκευτικό χώρο cloud S3 Bucket.
- Υπηρεσία Lambda@Edge: Εκτελείται πριν την υπηρεσία CloudFront δηλαδή πριν ελέγξει τη cache. Επίσης, η συνάρτηση έχει οριστεί να εκτελείται στο (viewer request) και ανακατευθύνει τα αιτήματα των social media crawlers στα αντίστοιχα endpoints στο backend.
- Υπηρεσία S3 Bucket: Αποθηκεύει τα στατικά αρχεία της React εφαρμογής όπως HTML, CSS, JavaScript και εικόνες. Επίσης, αποθηκεύει εικόνες που σχετίζονται με τα δεδομένα της βάσης δεδομένων.
- Υπηρεσία Certificate Manager: Η επικοινωνία μεταξύ των υπηρεσιών της Amazon προστατεύεται μέσω του πιστοποιητικού SSL το οποίο κάθε φόρα επικυρώνεται από την υπηρεσία AWS Certificate Manager.
- Ανακατεύθυνση αιτημάτων χωρίς το πρόθεμα www: Αιτήματα τα οποία δεν περιέχουν το πρόθεμα www (π.χ. moonkeyeu.com), ανακατευθύνομαι στο CloudFront Distribution μέσω του Route 53 και στη συνέχεια στο S3 Bucket με όνομα moonkeyeu.com. Τότε το αίτημα ανακατευθύνεται στο CloudFront Distribution, το οποίο επικοινωνεί με το S3 Bucket που είναι αποθηκευμένα τα στατικά αρχεία της εφαρμογής React και επιστρέφονται στον χρήστη.

6.1.1 Ανάλυση του Amazon Route 53 DNS

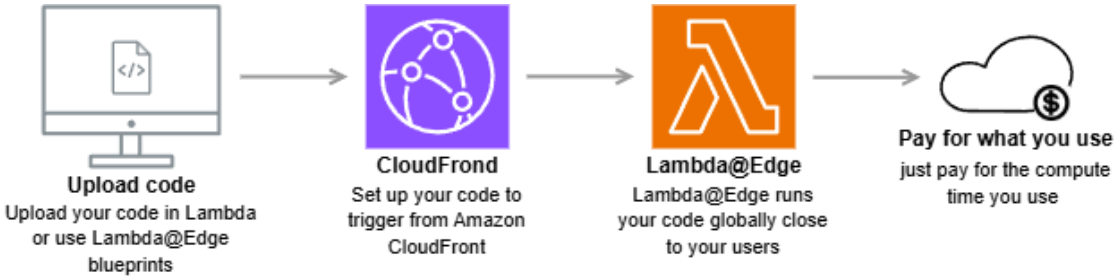
Εικόνα 38: Υπηρεσία Amazon Route 53.

Το Amazon Route 53 προσφέρει μια εξαιρετικά αποδοτική και αξιόπιστη υπηρεσία ονομάτων τομέα (DNS). Απλοποιεί τη διαδικασία ανακατεύθυνσης των διευθύνσεων IP των διακομιστών (servers) σε ονόματα που μπορούν να κατανοήσουν οι χρήστες. Για παράδειγμα, οι χρήστες μπορούν να χρησιμοποιούν τη διεύθυνση www.example.com αντί για την αριθμητική διεύθυνση 19.169.0.34, για να ανακατευθυνθούν σε μια υπηρεσία. Επιπλέον, το Route 53 παρέχει μηχανισμούς ελέγχου υγείας (health checks) για την παρακολούθηση της κατάστασης των υπηρεσιών και κατευθύνει την κυκλοφορία (Traffic) μόνο σε υγιείς endpoints. Παρέχει επίσης αρκετές μεθόδους δρομολόγησης όπως:

- Απλή δρομολόγηση (Simple routing): Η πιο βασική μορφή, όπου ένα όνομα τομέα κατευθύνει την κυκλοφορία σε ένα μόνο endpoint χωρίς λογική επιλογής.
- Δρομολόγηση βάσει καθυστέρησης (Latency Based Routing): Επιλέγει το endpoint με τη μικρότερη καθυστέρηση με βάση τη γεωγραφική τοποθεσία του χρήστη για ταχύτερη απόκριση.
- Δρομολόγηση βάσει γεωγραφικής τοποθεσίας (Geolocation Routing): Η κυκλοφορία κατευθύνεται ανάλογα με τη γεωγραφική τοποθεσία του χρήστη (π.χ. χώρα ή ήπειρο).
- Σταθμισμένη δρομολόγηση (Weighted Routing): Επιτρέπει την κατανομή της κυκλοφορίας (Traffic) μεταξύ πολλών endpoints σύμφωνα με προκαθορισμένα ποσοστά φόρτου (Load Balancing).
- Δρομολόγηση υπερχείλισης ή αστοχίας (Failover Routing): Επιτρέπει την ανακατεύθυνση της κυκλοφορίας σε εφεδρικό endpoint σε περίπτωση αστοχίας του κύριου. Συνδυάζεται με τους ελέγχους υγείας (health checks) για να εξασφαλιστεί η υψηλή διαθεσιμότητα της υπηρεσίας.

Όπως φαίνεται στο παραπάνω διάγραμμα, χρησιμοποιείται απλή δρομολόγηση (Simple Routing). Αφού οι χρήστες χρησιμοποιούν τη διεύθυνση διακομιστή www.moonkeyeu.com ή moonkeyeu.com και ανακατευθύνονται στην υπηρεσία CloudFront.

6.1.2 Ανάλυση των Συναρτήσεων Lambda@Edge



Εικόνα 39: Συνάρτηση Lambda@Edge.

Το AWS Lambda είναι μια υπηρεσία υπολογιστικής ισχύος χωρίς κάποιο διακομιστή (serverless computing service). Επιτρέπει την εκτέλεση κώδικα χωρίς την ανάγκη διαχείρισης υποδομών ή διακομιστών. Ο κώδικας οργανώνεται σε Lambda functions, οι οποίες υποστηρίζουν πολλές δημοφιλείς γλώσσες προγραμματισμού, όπως JavaScript, Java, Python, και άλλες.

Επιπλέον, το Lambda@Edge είναι μια επέκταση του AWS Lambda και σε αντίθεση με στους παραδοσιακές συναρτήσεις Lambda οι οποίες πρέπει να οριστούν χειροκίνητα από ποιες περιοχές θα εξυπηρετούνται. Οι συναρτήσεις Lambda@Edge έχουν το πλεονέκτημα ότι χρησιμοποιούν το παγκόσμιο δίκτυο διανομής περιεχομένου (Content Delivery Network) CloudFront και επιτρέπει τη διανομή των αποτελεσμάτων στους συνάρτησης σε edge locations. Δηλαδή σε γεωγραφικά διασκορπισμένα σημεία κοντά στους τελικούς χρήστες, επιτυγχάνοντας μεγαλύτερη απόδοση και μικρότερους χρόνους απόκρισης.

Υπάρχουν αρκετοί τρόποι που μπορεί κάποιος να χρησιμοποιήσει τα Lambda@Edge κάποιες ενδεικτικές είναι οι εξής:

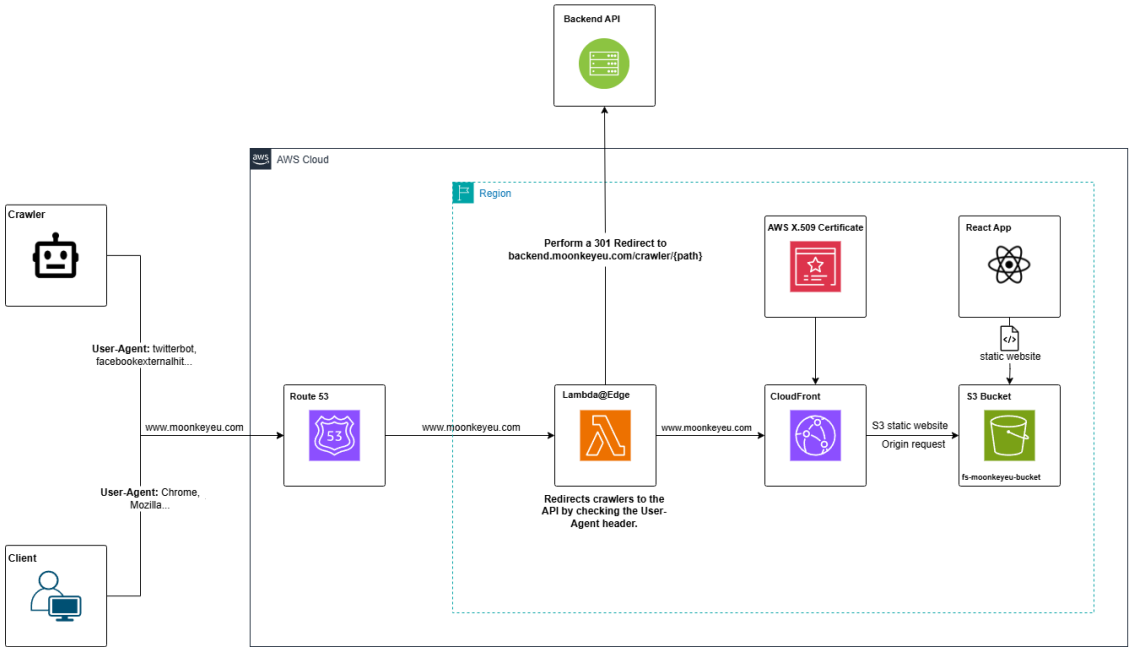
- Δυναμικές Web εφαρμογές (Dynamic Web Application): Συνδυάζοντας τις συναρτήσεις Lambda@Edge με άλλες υπηρεσίες της AWS, οι προγραμματιστές έχουν τη δυνατότητα να δημιουργήσουν ισχυρές και αποδοτικές web εφαρμογές μέσω του δικτύου (edge), οι οποίες κλιμακώνονται αυτόματα ανάλογα με τη ζήτηση τόσο προς τα πάνω όσο και προς τα κάτω. Όλα αυτά επιτυγχάνονται χωρίς την ανάγκη αρχικής

υποδομής όπως δημιουργία αντιγράφων ασφαλείας ή την εξασφάλιση πλεονασμού σε επίπεδο data center.

- Βελτιστοποίηση μηχανών αναζήτησης (SEO): Οι συναρτήσεις Lambda@Edge μπορούν να χρησιμοποιηθούν για τη βελτιστοποίηση μηχανών αναζήτησης (SEO) για web εφαρμογές. Για παράδειγμα, μπορεί να χρησιμοποιηθεί μια Lambda@Edge συνάρτηση η οποία θα παραδίδει μια προκαθορισμένη σελίδα HTML η οποία θα υπάρχει στο αποθηκευτικό χώρο cloud (S3 Bucket) και θα έχουν πρόσβαση σε αυτή μόνο crawler bots.
- Προσαρμογή εικόνων σε πραγματικό χρόνο (Real-Time Image Transformation): Με τις συναρτήσεις Lambda@Edge μπορεί να προσαρμοστεί η εμπειρία του χρήστη σε μια εφαρμογή προσαρμόζοντας της εικόνες της με βάση τη συσκευή του χρήστη. Επίσης, μπορούν να αποθηκευτούν αυτές οι εικόνες στη cache του παγκόσμιου δικτύου διανομής CloudFront για καλύτερη απόδοση.

Επίσης, η εκτέλεση των συναρτήσεων Lambda@Edge μπορεί να γίνει σε τέσσερα σημεία ροής:

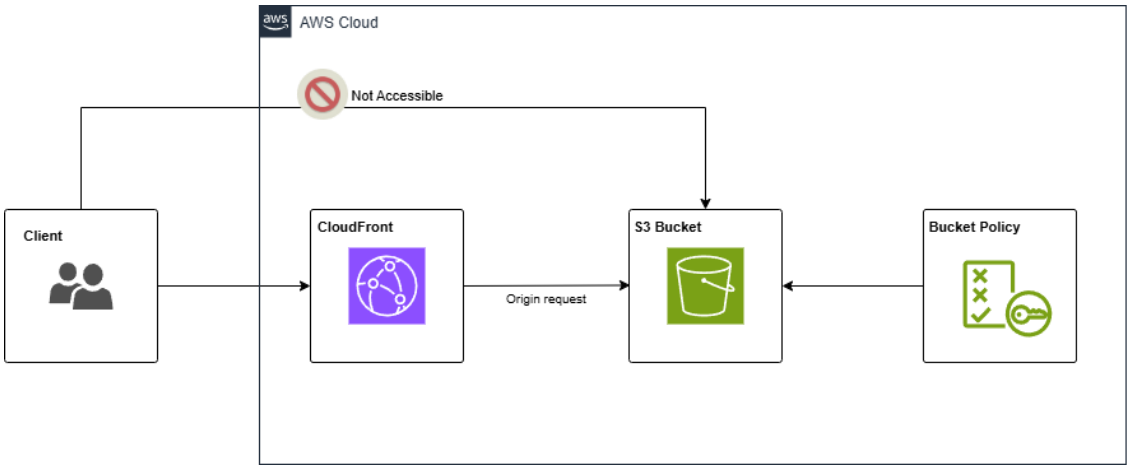
- Πριν το αίτημα φτάσει στο cache (Viewer Request) : Πρόκειται για το πρώτο σημείο εκτέλεσης μιας συνάρτησης, το οποίο ενεργοποιείται μόλις ο τελικός χρήστης (viewer) στείλει ένα αίτημα προς το CloudFront και πριν το CloudFront ελέγξει αν το περιεχόμενο υπάρχει ήδη στη cache ή πριν προωθήσει το αίτημα στον origin server.
- Αφού σταλεί απάντηση στον χρήστη (Viewer Response): Ενεργοποιείται αφού το CloudFront έχει λάβει την απάντηση (response), είτε από το cache είτε από τον origin, πριν σταλεί πίσω στον τελικό χρήστη.
- Πριν το αίτημα φτάσει στον origin (Origin Request): Αφορά τα αιτήματα που δεν εξυπηρετούνται από το cache του CloudFront, και επομένως πρόκειται να σταλούν στον origin server (π.χ. σε ένα controller ενός API).
- Μετά από απάντηση του origin (Origin Response): Εκτελείται μια συνάρτηση μόλις ο origin server απαντήσει. Δηλαδή, αυτό συμβαίνει πριν η απόκριση (response) επιστρέψει στο CloudFront και στη συνέχεια στον τελικό χρήστη.



Εικόνα 40: Ανακατεύθυνση Crawler bots με Lambda@edge.

Στη συνέχεια, όπως προ αναφέραμε στην ενότητα 2 το frontend υλοποιήθηκε ως ένα Single Page Application (SPA). Αυτό έχει ως αποτέλεσμα από τη φύση του όταν η σελίδα φορτώνει δε φορτώνονται από την αρχή και τα meta tags στο head, με αποτέλεσμα να μην είναι ορατά από τα crawler bots. Για το λόγο αυτό χρησιμοποιήθηκε μια συνάρτηση lambda@Edge για τη βελτιστοποίηση του μηχανισμού αναζήτησης (SEO) η οποίο εκτελείται πριν το αίτημα φτάσει στο cache του CloudFront (Viewer Request). Ελέγχει το header user-agent ώστε να διαπιστώσει εάν το αίτημα προέρχεται από κάποιο crawler bot ή από κάποιο χρήστη. Έτσι, με αυτό τον τρόπο ανακατευθύνει τα αιτήματα των crawlers στον CrawlerController του backend όπου θα λάβουν δυναμικά το head με τα meta tags από τον διακομιστή (SSR). Αντίθετα, αιτήματα που αφορούν τους χρήστες ακολουθούν την κανονική ροή και ανακατευθύνονται στο frontend.

6.1.3 Ανάλυση του Amazon CDN CloudFront



Εικόνα 41: Υπηρεσία CloudFront.

Η υπηρεσία Amazon CloudFront αποτελεί μια από τη σημαντικότερες υπηρεσία καθώς επιταχύνει τη διανομή του στατικού περιεχομένου, όπως HTML, CSS, JavaScript αλλά και διάφορα αρχεία πολυμέσων στους χρήστες. Το CloudFront αποτελείται από ένα παγκόσμιο δίκτυο διαμοιρασμού περιεχομένου (Content Delivery Network), το οποίο αποτελείται από κάποια κέντρα δεδομένων (data centers) τα οποία ονομάζονται edge locations. Έτσι, όταν ένας χρήστης πραγματοποιεί κάποιο αίτημα τότε ανακατευθύνετε στο κοντινότερο edge location με τη μικρότερη καθυστέρηση (lowest latency) και το περιεχόμενο διαμοιράζεται στον τελικό χρήστη με τη μέγιστη δυνατή ταχύτητα.

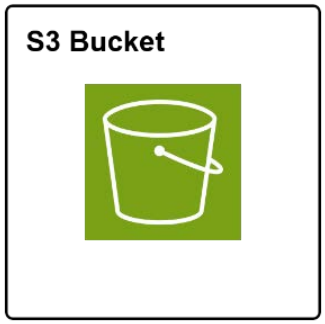
Επίσης, πριν γίνει ο διαμοιρασμός του περιεχομένου στον τελικό χρήστη, εκτελούνται τα εξής:

1. Εάν το περιεχόμενο υπάρχει ήδη σε κάποιο edge location με τη χαμηλότερη καθυστέρηση (Lowest Latency), τότε το CloudFront το επιστρέφει αμέσως στον τελικό χρήστη.
2. Εάν το περιεχόμενο δε βρίσκεται σε κάποιο edge location, τότε το CloudFront το ανακτά από κάποια άλλη υπηρεσία όπως το S3 Bucket ή από κάποιο API ή από κάποια άλλη πηγή.

Στην παρούσα περίπτωση, χρησιμοποιήθηκαν τρεις υπηρεσίες CloudFront η μία αφορά τον διαμοιρασμό των στατικών αρχείων της εφαρμογής React. Η δεύτερη αφορά τον διαμοιρασμό των εικόνων της εφαρμογής αλλά και εικόνων που σχετίζονται με τα δεδομένα στη βάση δεδομένων MySQL. Ενώ, η τρίτη υπηρεσία αφορά την ανακατεύθυνση αιτημάτων προς στη σελίδα που δεν έχουν το πρόθεμα www. Όλα αυτά τα αρχεία

βρίσκονται αποθηκευμένα στο αποθηκευτικό χώρο Cloud S3 Bucket όπου θα δούμε αναλυτικότερα στη συνέχεια.

6.1.4 Ανάλυση του Amazon Cloud Storage S3 Bucket



Εικόνα 42: Υπηρεσία Cloud Storage S3 Bucket.

Η υπηρεσία Amazon Simple Storage Service (S3 Bucket) αποτελεί μια επεκτάσιμη, αξιόπιστη και ασφαλή λύση αποθήκευσης αντικειμένων (object storage), η οποία προσφέρει υψηλή διαθεσιμότητα και απόδοση. Απευθύνεται σε οργανισμούς και επιχειρήσεις κάθε μεγέθους, επιτρέποντας την αποθήκευση και προστασία οποιουδήποτε όγκου δεδομένων για ένα ευρύ φάσμα περιπτώσεων χρήσης, όπως:

- Λίμνες δεδομένων (Data Lakes)
- Φιλοξενία ιστοσελίδων
- Εφαρμογές κινητών
- Δημιουργία αντιγράφων ασφαλείας και επαναφοράς (Disaster Recovery Plan)
- Αρχειοθέτηση
- Συσκευές Internet of Things (IoT)
- Ανάλυση μεγάλων δεδομένων (Big Data)

Στην παρούσα περίπτωση, χρησιμοποιήθηκε σε συνδυασμό με την υπηρεσία CloudFornt και όπως προαναφέραμε στην προηγούμενη ενότητα μπορεί να χρησιμοποιηθεί για παγκόσμιο διαμοιρασμό του στατικού περιεχομένου. Επίσης, χρησιμοποιήθηκαν τρία buckets, τα δύο αφορούν τα στατικά αρχεία της εφαρμογής React και την ανακατεύθυνση αιτημάτων που δεν έχουν το πρόθεμα www, ενώ το τρίτο αφορά εικόνες που σχετίζονται με τα δεδομένα της βάσης δεδομένων.

6.1.5 Έκδοση πιστοποίησης SSL Μέσω του Certificate Manager

Εικόνα 43: Υπηρεσία Certificate Manager.

Η υπηρεσία Amazon Certificate Manager (ACM) αναλαμβάνει τη διαχείριση και απλοποιεί τη διαδικασία της δημιουργίας αλλά και της ανανέωσης των πιστοποιητικών SSL/TSL X.509 τα οποία χρησιμοποιούνται για την προστασία και την ασφαλή επικοινωνία των υπηρεσιών cloud στο περιβάλλον της Amazon Web Services (AWS). Επίσης, πιστοποιητικά που παρέχονται από την υπηρεσία Amazon Certificate Manager μπορούν να προστατεύουν μεμονωμένα ονόματα τομέων (domain), πολλαπλά ονόματα τομέων αλλά και απεριόριστο αριθμό υποτομέων.

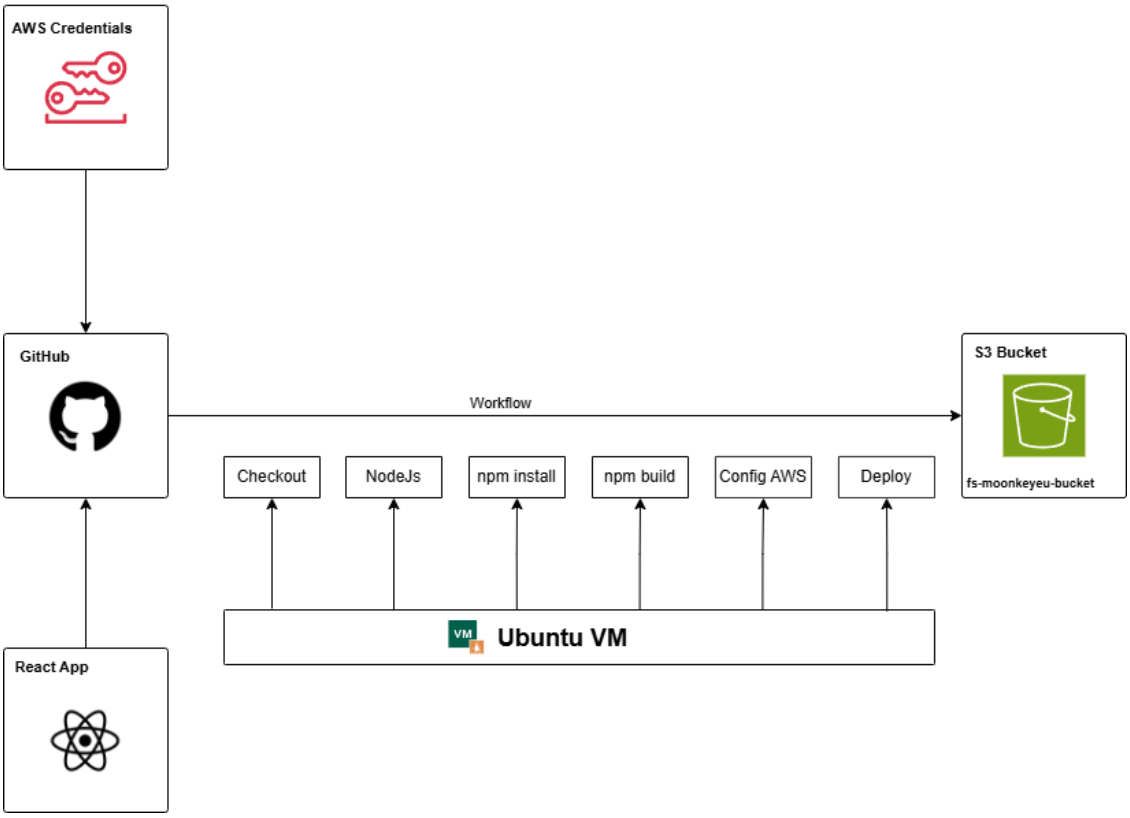
Στην παρούσα εργασία, χρησιμοποιείται για την πιστοποίηση του τομέα της σελίδας (domain name), έτσι ώστε να μπορεί να χρησιμοποιηθεί το όνομα του τομέα στις εσωτερικές υπηρεσίες της AWS.

6.1.6 Αυτοματοποίηση της Εγκατάστασης Μέσω των GitHub Actions

Εικόνα 44: Λογότυπο GitHub Actions [32].

Η υπηρεσία GitHub Actions αποτελεί ένα εργαλείο της πλατφόρμας GitHub και επιτρέπει την αυτοματοποιημένη εκτέλεση ροών εργασιών (Workflows). Μια ροή εργασίας είναι μια αυτοματοποιημένη διαδικασία που περιλαμβάνει μία ή περισσότερες εργασίες, οι

οποίες μπορούν να παραμετροποιηθούν, ώστε να εκτελούνται όταν πραγματοποιούνται συγκεκριμένες ενέργειες στο repository, όπως η προσθήκη νέου κώδικα (push request) ή η υποβολή αιτήματος συγχώνευσης (pull request). Η παραμετροποίηση των ροών εργασιών γίνεται μέσω αρχείων με μορφή YAML, τα οποία περιγράφουν τα βήματα και τις συνθήκες εκτέλεσης κάθε εργασίας.



Εικόνα 45: Αυτοματοποίηση εγκατάστασης στο cloud.

Στην παρούσα εργασία, χρησιμοποιήθηκε η υπηρεσία GitHub Actions για την αυτοματοποιημένη εκτέλεση της διαδικασίας της εγκατάστασης του frontend στην πλατφόρμα cloud της AWS.

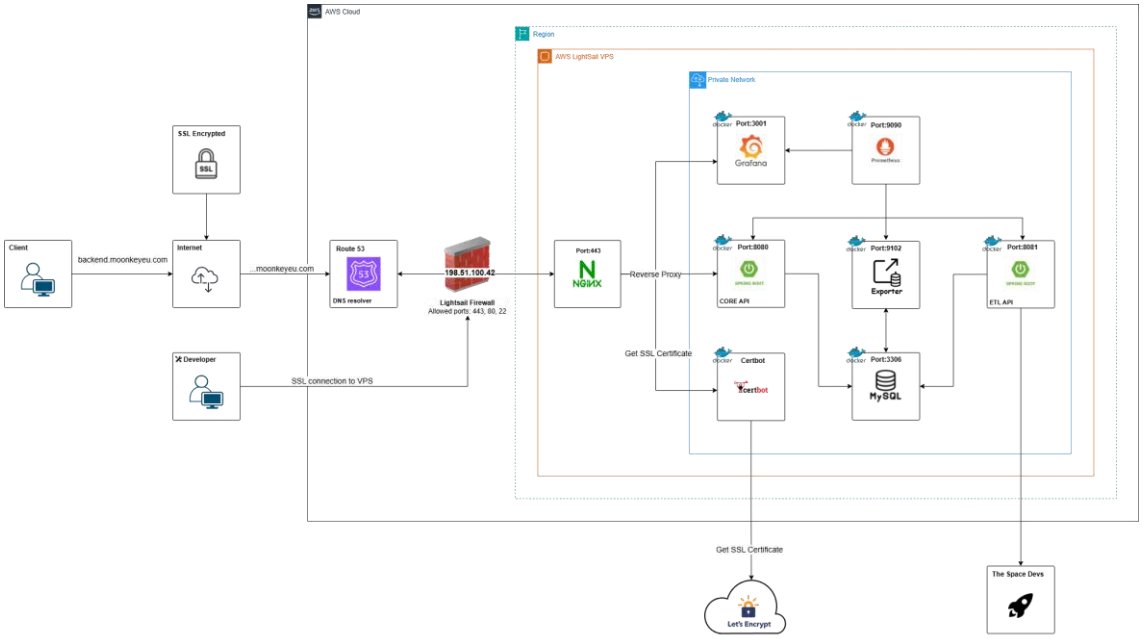
Αρχικά πρέπει να οριστεί ένα repository το όποιο θα περιέχει το κώδικα του frontend. Επίσης, πρέπει να οριστούν τα απαραίτητα διαπιστευτήρια στον περιβάλλον εκτέλεσης του repository. Στη συνέχεια, αφού ορίσουμε τα παραπάνω μπορεί να ξεκινήσει η διαδικασία ορίσματος του αρχείου YAML στο όποιο θα οριστεί η ροή της εργασίας. Αυτό πραγματοποιείται δημιουργώντας ένα αρχείο main.yaml στο φάκελο .github/workflow που ορίστηκε στο φάκελο του έργου που αφορά το frontend. Επιπλέον, ορίζουμε ότι η εργασία θα εκτελείται όταν προστίθεται νέος κώδικας και για να εκτελεστούν αυτές οι

εργασίες θα πρέπει να οριστεί ένας εικονικός υπολογιστής με ubuntu. Αφού οριστούν τα παραπάνω εκτελούνται τα ακόλουθα βήματα:

- Ορισμός repository Checkout: Ορίζουμε το repository στο οποίο θα γίνουν οι εργασίες.
- Εγκατάσταση NodeJs: Για να προετοιμάσουμε την εφαρμογή React για εγκατάσταση (deployment) πρέπει να εγκατασταθεί κάποια έκδοση του NodeJs στον εικονικό υπολογιστή.
- Εγκατάσταση εξαρτήσεων (npm install): Εγκαθιστούμε όλες τις απαραίτητες βιβλιοθήκες της εφαρμογής React.
- Δημιουργία τελικού πακέτου της εφαρμογής React: Εκτέλεση των εντολών που δημιουργούν το τελικό αρχείο της εφαρμογής.
- Προσθήκη απαραίτητων διαπιστευτηρίων: Ορίζουμε τα απαραίτητα διαπιστευτήρια για την πρόσβαση στον αποθηκευτικό χώρο S3 Bucket.
- Εγκατάσταση της εφαρμογής στο αποθηκευτικό χώρο S3 Bucket: Αφού οριστούν τα απαραίτητα διαπιστευτήρια τότε μπορεί να εκτελεστεί το βήμα που εκτελεί την εντολή για το ανέβασμα του έργου στον αποθηκευτικό χώρο S3 Bucket.

Σύμφωνα με τα παραπάνω, συμπεραίνουμε ότι το εργαλείο GitHub Actions μας βοηθάει να αυτοματοποιούμε τη διαδικασίας της εγκατάστασης μιας εφαρμογής σε μία cloud πλατφόρμα. Στη συγκεκριμένη περίπτωση γλυτώνουμε αρκετό χρόνο αφού όλες αυτές τις εντολές θα χρειαζόταν να εκτελεστούν χειροκίνητα εάν δε χρησιμοποιούσαμε GitHub Actions. Έτσι, με μόνο μια εντολή (git pull origin master) νέες αλλαγές στον κώδικα μπορούν να επηρεάσουν την τελική εφαρμογή στο cloud άμεσα.

6.2 Αρχιτεκτονική Backend στο Cloud



Εικόνα 46: Αρχιτεκτονική backend στο cloud.

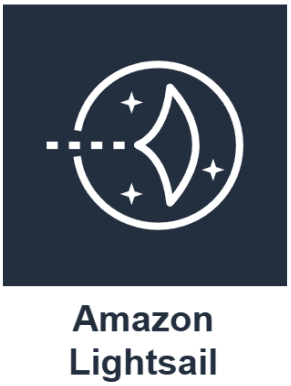
Στην Εικόνα 46 παρουσιάζεται η αρχιτεκτονική του backend στο cloud η οποία υλοποιήθηκε στην πλατφόρμα Amazon Web Services (AWS). Επίσης, ακολουθεί μια πρώτη ανάλυση της ροής του διαγράμματος, η ανάλυση της λειτουργίας και ο ρόλος της κάθε υπηρεσίας θα πραγματοποιηθεί στις επόμενες υποενότητες που ακολουθούν. Η ροή των αιτημάτων έχει ως εξής:

- Πελάτης (Client): Ο χρήστης κάνει ένα αίτημα HTTP(S) προς το domain της εφαρμογής (π.χ. backend.moonkeyeu.com). Επίσης, σε αυτή την περίπτωση πραγματοποιούνται αιτήματα όχι μόνο από απλούς χρήστες αλλά και από άλλες εφαρμογές, για παράδειγμα από την εφαρμογή React του frontend.
- Πρωτόκολλο SSL: Η επικοινωνία μεταξύ του πελάτη αλλά και του API πραγματοποιείται μέσω ασφαλούς πρωτοκόλλου επικοινωνίας SSL.
- Σύστημα ονομάτων τομέων (Route 53 DNS): Ανακατευθύνει τα αιτήματα του domain στη διεύθυνση (IP) του Virtual Private Server (VPS).
- Ενδιάμεσος διακομιστής Nginx Reverse Proxy: Ανακατευθύνει τα αιτήματα στις αντίστοιχες υπηρεσίες εντός του ιδιωτικού δικτύου Docker.
- Υπηρεσία CertBot: Βρίσκεται εσωτερικά σε ένα Docker container και αναλαμβάνει την έκδοση αλλά και την ανανέωση της πιστοποίησης SSL από την LetsEncrypt.

Αυτό πραγματοποιείται με την εκτέλεση ενός cronjob το οποίο εκτελεί το container κάθε δεκαπέντε ημέρες, έτσι ώστε να διαπιστώσει εάν η πιστοποίηση SSL έχει λήξει.

- Υπηρεσία MySQL: Βρίσκεται εσωτερικά σε ένα Docker container και αναλαμβάνει την αποθήκευση των δεδομένων.
- Υπηρεσία Spring Boot Core API: Βρίσκεται εσωτερικά σε ένα Docker container και αναλαμβάνει τις περισσότερες λειτουργίες του backend όπως την πιστοποίηση του χρήστη, διαμοιρασμό του περιεχόμενου.
- Υπηρεσία Spring Boot ETL API: Βρίσκεται εσωτερικά σε ένα Docker container. Αναλαμβάνει την ανάσυρση τον δεδομένων από το API The Space Devs, τον μετασχηματισμό αλλά και την αποθήκευση τον νέων δεδομένων στη βάση MySQL.
- Υπηρεσία συλλογής μετρήσεων Prometheus: Βρίσκεται εσωτερικά σε ένα Docker container και αναλαμβάνει τη συλλογή μετρήσεων από τις υπηρεσίες Core API, ETL API και MySQL. Οι μετρήσεις αφορούν τους πόρους που καταναλώνουν οι υπηρεσίες στον διακομιστή αλλά και την παρακολούθηση αιτημάτων προς αυτές.
- Υπηρεσία προβολής μετρήσεων Grafana: Βρίσκεται εσωτερικά σε ένα Docker container. Αναλαμβάνει την τροποποίηση και την προβολή τον μετρήσεων σε ένα φιλικό προς το χρήστη περιβάλλον.

6.2.1 Ανάλυση του Amazon LightSail VPS Server



Εικόνα 47: Amazon LightSail VPS Server.

Ένας εικονικός ιδιωτικός διακομιστής (Virtual Private Server) αποτελεί μια λύση φιλοξενίας εφαρμογών και βασίζεται στην τεχνολογία εικονικοποίησης (virtualization). Συγκεκριμένα, ένας φυσικός διακομιστής (deticated server) χωρίζεται σε πολλούς ανεξάρτητους εικονικούς διακομιστές (servers), ο καθένας εκ των οποίων λειτουργεί με δικούς

του υπολογιστικούς πόρους (π.χ. CPU, RAM, αποθηκευτικός χώρος) και έχουν ανεξάρτητο λειτουργικό σύστημα.

Επιπλέον, Οι διακομιστές VPS παρέχουν στους χρήστες πλήρη πρόσβαση (root access) επιτρέποντας την εγκατάσταση και διαμόρφωση του λογισμικού σύμφωνα με τις απαιτήσεις της εκάστοτε εφαρμογής ή υπηρεσίας. Επίσης, λόγω της απομόνωσης των πόρων προσφέρουν μεγαλύτερη σταθερότητα, ασφάλεια και απόδοση σε σχέση με τις λύσεις κοινόχρηστης φιλοξενίας (shared hosting).

Στο πλαίσιο αυτό, η υπηρεσία Amazon Lightsail προσφέρει μία απλοποιημένη και προσιτή λύση για τη δημιουργία και διαχείριση διακομιστών VPS στο cloud. Η πλατφόρμα Lightsail παρέχει προκαθορισμένα πακέτα με συγκεκριμένες προδιαγραφές υλικού όπως αριθμό επεξεργαστών, μνήμη και χώρο αποθήκευσης με σταθερή τιμολόγηση. Καθιστώντας την ιδανική λύση για όσους επιθυμούν να αναπτύξουν εφαρμογές χωρίς να εμπλακούν με την πολυπλοκότητα των πιο εξελιγμένων υπηρεσιών του AWS όπως το Amazon EC2.

Στην παρούσα εργασία, χρησιμοποιήθηκε η υπηρεσία Amazon LightSail για τη φιλοξενία ολόκληρης της υποδομής του backend, η οποία αποτελείται από τον Nginx Proxy Server και τις υπηρεσίες του API που βρίσκονται σε Docker containers..

6.2.2 Ανάλυση του Docker

Εικόνα 48: Λογότυπο Docker [33].

Το Docker είναι μια πλατφόρμα ανοιχτού κώδικα η οποία χρησιμοποιεί την τεχνολογία εικονικοποίησης (virtualization) μέσω του containerization βελτιστοποιώντας την ανάπτυξη λογισμικού. Περιτυλίγει μια εφαρμογή σε ένα container το οποία δεν απαιτεί πολλούς πόρους και περιλαμβάνει όλες της απαραίτητες βιβλιοθήκες που απαιτούνται για να εκτέλεση της. Έτσι, η εφαρμογή μπορεί να εκτελείται σε διαφορετικά περιβάλλοντα και

δεν προαπαιτεί την εγκατάσταση του απαραίτητου λογισμικού για την εκτέλεση της, βελτιώνοντας τη φορητότητα και την υποστήριξη της εφαρμογής.

Αναλυτικότερα, τα containers και οι εικονικές μηχανές (Virtual Machines) βασίζονται στην τεχνολογία εικονικοποίησης αλλά με διαφορετικό τρόπο. Οι εικονικές μηχανές περιλαμβάνουν ολόκληρο το λειτουργικό σύστημα, πράγμα που τις κάνει να απαιτούν περισσότερους πόρους και να αργούν κατά της διάρκεια εκκίνησης. Αντίθετα, τα containers μοιράζονται τον πυρήνα του λειτουργικού συστήματος του διακομιστή δηλαδή του ηλεκτρονικού υπολογιστή όπου τρέχουν και απομονώνουν μόνο την εφαρμογή και τις εξαρτήσεις της. Με αποτέλεσμα, τα containers να μην απαιτούν πολλούς πόρους πράγμα τα οποία τα καθιστά περισσότερο αποδοτικά και γρηγορότερα κατά την εκκίνηση τους. Στη συνέχεια, αναλύονται τα βασικότερα χαρακτηρίστηκα του Docker:

- Docker Images: Ένα Dockerfile είναι ένα αρχείο κειμένου που περιέχει οδηγίες για τη δημιουργία ενός Docker image. Τα Docker images είναι αυτόνομα και φορητά περιβάλλοντα που περιλαμβάνουν όλα τα απαραίτητα στοιχεία για την εκτέλεση μιας εφαρμογής όπως τον πηγαίο κώδικα, τις βιβλιοθήκες και τις απαιτούμενες ρυθμίσεις. Επίσης, αξίζει να σημειωθεί ότι τα images είναι αμετάβλητα (immutable), δηλαδή δεν μπορούν να τροποποιηθούν αφού δημιουργηθούν.
- Containers: Ένα container είναι ένα απομονωμένο περιβάλλον, πράγμα που σημαίνει ότι δεν έχει πρόσβαση ούτε γνωρίζει το λειτουργικό σύστημα ή τα αρχεία του υπολογιστή του διακομιστή που εκτελείται.
- Volumes: Τα volumes επιτρέπουν την αποθήκευση δεδομένων εκτός του container, έτσι διασφαλίζεται πως όταν ένα container θα αντικατασταθεί, ενημερωθεί ή για κάποιο λόγο διαγραφή δε θα διαγραφούν τα δεδομένα. Επιπλέον, τα volumes μπορούν να μοιραστούν μεταξύ πολλών containers, επιτρέποντας την κοινή χρήση δεδομένων μεταξύ διαφορετικών εφαρμογών.
- Docker Compose: Το Docker Compose είναι ένα εργαλείο που επιτρέπει τη δήλωση και τη διαχείριση πολλαπλών Docker containers χρησιμοποιώντας ένα απλό αρχείο κειμένου, γνωστό ως docker-compose.yml. Επιπλέον, επιτρέπει τη δημιουργία ενός σύνολο εφαρμογών (application stack), περιγράφοντας τις υπηρεσίες, τα δίκτυα και τα volumes που απαιτούνται ώστε τα containers να συνεργάζονται μεταξύ τους.

Στην παρούσα εργασία αξιοποιήθηκε το Docker με σκοπό την απλοποίηση της διαδικασίας ενσωμάτωσης της εφαρμογής στον διακομιστή VPS. Όπως φαίνεται και στην Εικόνα 46, οι υπηρεσίες Spring Boot Core API, Spring Boot ETL API, καθώς και οι

MySQL, Prometheus και Grafana, εκτελούνται μέσα σε Docker containers. Χάρη στη χρήση των containers, έχει προκαθοριστεί ο τρόπος παραμετροποίησης κάθε υπηρεσίας σε ξεχωριστά απομονωμένα περιβάλλοντα, επιτρέποντας την εκτέλεση τους σε διαφορετικά λειτουργικά συστήματα χωρίς την ανάγκη τροποποίησης του βασικού περιβάλλοντος του διακομιστή.

6.2.3 Ανάλυση του Nginx Proxy Server

Εικόνα 49: Λογότυπο Nginx [34].

Ο Nginx είναι ένας Web Server οποίος συνήθως χρησιμοποιείται σαν ένας διακομιστής μεσολάβησης (Reverse Proxy) ανάμεσα στους τελικούς χρήστες και στους διακομιστές εφαρμογών (Application Servers). Έτσι, τα αιτήματα των χρηστών υποβάλλονται αρχικά στον Nginx, ο οποίος βάσει των ρυθμίσεων του τα ανακατευθύνει προς τις κατάλληλες υπηρεσίες και στην συνέχεια επιστρέφει την απάντηση στους χρήστες.

Στην παρούσα εργασία, χρησιμοποιήθηκε ο Nginx Web Server ως μεσολάβησης για την ανακατεύθυνση των αιτημάτων προς τις κατάλληλες υπηρεσίες του backend οι οποίες βρίσκονται σε Docker containers. Επίσης, χρησιμοποιείται για τη διαχείριση και αποκρυπτογράφηση των SSL/TLS αιτημάτων.

7 Συμπεράσματα

Η ανάπτυξη της διαδικτυακής εφαρμογής με σκοπό την προβολή δεδομένων διαστημικών αποστολών ολοκληρώθηκε με επιτυχία. Μέσω της εφαρμογής, ο χρήστης μπορεί να ενημερώνεται για το πότε θα πραγματοποιηθεί κάποια εκτόξευση. Επίσης, έχει τη δυνατότητα να δει αναλυτικές πληροφορίες σχετικά με την αποστολή, όπως τους συμμετέχοντες αστροναύτες, τους εμπλεκόμενους οργανισμούς, το πρόγραμμα στο οποίο εντάσσεται, καθώς και άλλες σχετικές πληροφορίες.

Κατά την ανάπτυξη της εφαρμογής, παρουσιάστηκαν αρκετές δυσκολίες, κυρίως στο κομμάτι που αφορά την εξαγωγή, τον μετασχηματισμό και τη φόρτωση των δεδομένων, καθώς στην αυθεντικοποίηση των χρηστών. Ωστόσο, τα προβλήματα αυτά επιλύθηκαν επιτυχώς στην πορεία του έργου.

Επιπλέον, τα σημεία που απαίτησαν τον περισσότερο χρόνο για την υλοποίησή τους ήταν η σχεδίαση του γραφικού περιβάλλοντος του frontend και η υλοποίηση του backend μηχανισμού για την επεξεργασία των δεδομένων. Αναλυτικότερα, η υλοποίηση του backend μηχανισμού απαιτούσε τη μελέτη του SpaceDevs API, προκειμένου να γίνει κατανοητή η δομή των δεδομένων και να υλοποιηθούν σωστά οι σχέσεις στους πίνακες της βάσης δεδομένων. Στη συνέχεια, η κατασκευή του γραφικού περιβάλλοντος απαίτησε επιπλέον χρόνο ώστε να διασφαλιστεί η βέλτιστη εμπειρία χρήστη κατά την περιήγηση στην εφαρμογή.

Παρότι η εφαρμογή περιλαμβάνει πληθώρα δεδομένων σχετικών με διαστημικές εκτοξεύσεις, υπάρχουν ακόμη περιθώρια βελτίωσης και επέκτασης με νέες ενότητες, ώστε να εμπλουτιστεί περαιτέρω το περιεχόμενο.

Στο πλαίσιο μελλοντικής επέκτασης, θα μπορούσε να προστεθεί ενότητα με πληροφορίες για όλους τους διαστημικούς σταθμούς, καθώς και παρακολούθηση των καιρικών συνθηκών για τις πιο δημοφιλείς πλατφόρμες εκτοξεύσεων. Επιπλέον, θα μπορούσε να βελτιστοποιηθεί η απόδοση της διαδικασίας εξαγωγής, μετασχηματισμού και φόρτωσης των δεδομένων, ενώ προτείνεται και η ολοκλήρωση του μηχανισμού OAuth2, ώστε να

διευκολυνθεί η διαδικασία εγγραφής των χρηστών. Τέλος, για τη διασφάλιση της ποιότητας του λογισμικού, προτείνεται η ενσωμάτωση ελέγχων με JUnit τόσο στο backend όσο και στο frontend.

Βιβλιογραφία

[1] Pohl, Klaus, "Requirements Engineering: Fundamentals, Principles and Techniques", Springer-Verlag, 2010.

[2] Larman, Craig, "Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development ", 3rd Edition, Pearson, 2004.

[3] Deinum, Marten, "Spring Boot 3 Recipes: A Problem-Solution Approach for Java Microservices and Cloud-Native Applications ", Apress, 2024.

[4] Khalfallah, Héla Ben, "Semirings for Soft Constraint Solving and Programming", Apress, 2024.

[5] Reenskaug, Trygve M. H., "MVC, XEROX PARC 1978-79", available online (https://folk.universitetetioslo.no/trygver/themes/mvc/mvc-index.html (accessed, Sep. 2024).

[6] Ahmed, Mohammed Ilyas, "Cloud-Native DevOps: Building Scalable and Reliable Applications", Apress, 2024.

[7] Martin, Robert C., "Clean Code: A Handbook of Agile Software Craftsmanship", Prentice Hall, 2008.

[8] Martin, Robert C., "Clean Architecture: A Craftsman's Guide to Software Struc-ture and Design", Prentice Hall, 2017.

[9] https://github.com/TheSpaceDevs - TheSpaceDevs API Documentation. [2025]

[10] https://nextspaceflight.com/ - Next Spaceflight. [19/05/2025]

[11] https://www.spacelaunchschedule.com/ - Space Launch Schedule. [19/05/2025]

[12] https://www.nasa.gov/ - NASA. [19/05/2025]

[13] https://www.esa.int/ - European Space Agency (ESA). [19/05/2025]

[14] https://www.spacex.com/ - SpaceX. [19/05/2025]

[15] https://www.geeksforgeeks.org/what-is-single-page-application/ - Single Page Application Documentation. [25/12/2021]

[16] https://mariyabaig.com/react-a-comprehensive-guide/ - Component Based Architecture Documentation. [25/12/2021]

[17] https://1000logos.net/figma-logo/ - Figma Logo. [25/12/2021]

[18] https://tanstack.com/ - Tanstack Auery Documentation. [25/12/2021]

[19] https://commons.wikimedia.org/wiki/File:Axios_%28computer_library%29_logo.svg - Axios Logo. [25/12/2021]

[20] https://www.heatware.net/devops-cloud/deploy-spring-app-to-aws/ - Spring Boot Logo. [25/12/2021]

[21] https://www.geeksforgeeks.org/spring-boot-architecture/ - Spring Boot Architecture Documentation. [25/12/2021]

[22] https://codingnomads.com/spring-data-jpa - Spring Data JPA Diagram. [25/12/2021]

[23] https://docs.spring.io/spring-batch/reference/domain.html - Spring Batch Pro-cessing Documentation. [25/12/2021]

[24] https://medium.com/@knoldus/webclient-springs-way-to-connect-rest-services-5b6484226aed - Spring Web Client Logo. [25/12/2021]

[25] https://aws.amazon.com/rds/mysql/ - MySQL Logo. [25/12/2021]

[26] https://www.devskillbuilder.com/understanding-JSON-web-token-jwt-b7a9a5d6df37 - JSON Web Token Diagram. [25/12/2021]

[27] https://jwt.io/introduction - JSON Web Token Format. [25/12/2021]

[28] https://www.ssl.com/article/what-is-a-one-time-password-otp/ - One Time Password Diagram. [25/12/2021]

[29] https://www.geeksforgeeks.org/csrf-protection-in-flask/ - Cross-site Request Forgery Diagram. [25/12/2021]

[30] https://aws.amazon.com/what-is/cross-origin-resource-sharing/ - Cross-origin Re-Source Sharing Diagram. [25/12/2021]

[31] https://aws.amazon.com/lambda/edge/ - Amazon Web Services Lambda@edge Documentation [25/12/2021]

[32] https://1000logos.net/docker-logo/ - Docker logo. [25/12/2021]

[33] https://1000logos.net/docker-logo/ - Docker logo. [25/12/2021]

[34] https://1000logos.net/nginx-logo/ - Nginx logo. [25/12/2021]

[35] https://docs.docker.com/ - Docker Documentation. [25/12/2021]

[36] https://docs.spring.io/spring-boot/index.html - Spring Boot Documentation. [25/12/2021]

[37] https://nginx.org/en/docs/ - Nginx Documentation. [25/12/2021]

[38] https://www.geeksforgeeks.org/spring-boot-architecture/ - Spring Boot Architecture Documentation. [25/12/2021]

[39] https://docs.aws.amazon.com/ - Amazon Web Services Documentation. [25/12/2021]

[40] https://commons.wikimedia.org/wiki/File:Axios_%28computer_library%29_logo.svg - Axios Logo. [25/12/2021]

[41] https://api.octoperf.com/doc/integrations/ci/github-actions/ - GitHub Actions Logo. [25/12/2021]

[42] https://aws.amazon.com/architecture/icons/ - Εικονίδια Amazon Web Services. [25/12/2021]

[43] https://app.diagrams.net/ - Εργαλείο δημιουργίας διαγραμμάτων. [25/12/2021]

Παράρτημα Α

Στο παρακάτω παράρτημα παρουσιάζονται κάποιοι ενδεικτικοί κώδικες

Των component της React.

Ανάσυρση δεδομένων με Tanstack Query (Queries.jsx)

```
export const useParameterizedQuery = (
    {
        url,
        params = '',
        cacheKey,
        staleTime = 5 * 60 * 1000,
        placeholderData = keepPreviousData,
        refetchOnWindowFocus = true,
        retry = 2,
        queryOptions = {},
        options = { withCredentials: false, Csrf: false }
    }) => {
    const {showBoundary} = useErrorBoundary();
       return useQuery({
            queryKey: [cacheKey, params],
            queryFn: () => handleGet(url, options),
            placeholderData: placeholderData,
            refetchOnWindowFocus: refetchOnWindowFocus,
            refetchOnReconnect: true,
            staleTime: staleTime,
            retry: retry,
            ...queryOptions,
            throwOnError: (error) => showBoundary(error?.response),
        });
};
```

Αποστολή δεδομένων με Tanstack Query (Mutations.jsx)

```
export const useCreateMutation = (
    {
        successMessage = undefined,
        errorMessage = "Something went wrong!",
        queryKeysToInvalidate = [],
        mutationOptions = {}
    } = {}) => {
    const queryClient = useQueryClient();

    return useMutation({
        mutationFn: ({ url, data, options }) => handlePost(url, data,
options),
        onSuccess: () => {
            !!successMessage && toast.success(successMessage);
        },
        onError: (error) => {
```

```
            showErrorToast(error, error ? error.response?.data?.error
: errorMessage);
        },
        onSettled: async (_, error) => {
            if (!error && queryKeysToInvalidate.length > 0) {
                await Promise.all(
                    queryKeysToInvalidate.map((key) =>
                        queryClient.invalidateQueries({ queryKey:
[key] })
                    )
                );
            }
        },
        ...mutationOptions,
    });
};
```

Αναχαιτιστής αιτημάτων Axios Interceptors (AuthProvider.jsx)

```
   function subscribeTokenRefresh(callback) {
       failedQueue.push(callback);

   }

   function onRefreshed(newToken) {
       failedQueue.forEach(callback => callback(newToken));
       failedQueue = [];
   }

  useLayoutEffect(() => {
        const authInterceptor = api.interceptors.request.use((config)
=> {
            if (config.Bearer && token) {
                config.headers.Authorization = `Bearer ${token}`;
            }
            if (config.Csrf && csrfQuery?.data ) {
                config.headers["X-XSRF-TOKEN"] = csrfQuery?.data?.to-
ken;
            }
            return config;
        });

        return () => api.interceptors.request.eject(authInterceptor);
    }, [token, csrfQuery?.data]);

    useLayoutEffect(() => {
        const refreshInterceptor = api.interceptors.response.use(
            (response) => response, async (error) => {
                const originalRequest = error.config;

                if (
                    !originalRequest._retry
                    && (error?.response?.status === 401 && (error?.re-
sponse?.data?.error === "Unauthorized" || error?.response?.data?.busi-
nessErrorCode === 303))
                ) {
                    if (!isRefreshing.current) {
                        isRefreshing.current = true;
                        try {
```

```
                            const response = await refreshMutation.mu-
tateAsync({
                                url: refreshUrl,
                                data: null,
                                options: { withCredentials: true }
                            }

                            );
                            setToken(response.access_token);
                            onRefreshed(token);
                            api.headers.defaults.common['Authoriza-
tion'] = `Bearer ${token}`;
                            return Promise.all(failedQueue.map(cb =>
cb(token))).then(() => {
                                return api(originalRequest);
                            });
                        } catch (error){
                            return Promise.reject(error);
                        } finally {
                            isRefreshing.current = false;
                            failedQueue = [];
                        }
                    }
                    return new Promise((resolve, reject) => {
                        subscribeTokenRefresh((newToken) => {
                            originalRequest.headers.Authorization =
`Bearer ${newToken}`;
                            originalRequest._retry = true;
                            resolve(api(originalRequest));
                        });
                    });
                }
                return Promise.reject(error);
            }
        );
        return () => api.interceptors.response.eject(refreshIntercep-
tor);
    }, [token]);
```

Αιτήματα με Axios (api.jsx)

```
const api = axios.create({
    headers: {
        "Content-Type": "application/JSON",
    },
    withCredentials: true,
    Csrf: true,
    timeout: 60000
});

const handlePost = async (url, data, options) => {
    const response = await api.post(url, data, {...options});
    return response.data;
}
const handleGet = async (url, options ) => {
    const response = await api.get(url, {...options});
    return response.data;
}
const handlePut = async (url, data, options) => {
    const response = await api.put(url, data, {...options});
    return response.data;
}
```

```
const handleDelete = async (url, options) => {
    const response = await api.delete(url, {...options});
    return response.data;
}

export { api, handlePost, handleGet, handlePut, handleDelete };
```

Παράρτημα Β

Στο παρακάτω παράρτημα παρουσιάζονται κάποιοι ενδεικτικοί κώδικες

των υπηρεσιών του Spring Boot.

Ορισμός Bean Spring Security

```
@Configuration
@RequiredArgsConstructor
public class SecurityBeansConfig {
    private final UserDetailsService userDetailsService;

    @Bean
    public AuthenticationProvider authenticationProvider(){
        DaoAuthenticationProvider authProvider = new DaoAuthentica-
tionProvider();
        authProvider.setUserDetailsService(userDetailsService);
        authProvider.setPasswordEncoder(passwordEncoder());
        return authProvider;
    }
    @Bean
    public AuthenticationManager authenticationManager(Authentication-
Configuration config) throws Exception {
        return config.getAuthenticationManager();
    }
    @Bean
    public PasswordEncoder passwordEncoder(){
        return new BCryptPasswordEncoder();
    }
}
```

Φίλτρο JwtAuthenticationFilter.java

```
@Component
@RequiredArgsConstructor
@Slf4j
public class JwtAuthenticationFilter extends OncePerRequestFilter {
    private final JwtRequestExtractor jwtRequestExtractor;
    private final JwtServiceParserImpl jwtServiceParserImpl;
    private final JwtServiceProvider jwtServiceProvider;
    private final UserDetailsServiceImpl userDetailsService;
    private final TokenRepository tokenRepository;

    @Override
    protected void doFilterInternal(
            @NonNull HttpServletRequest request,
            @NonNull HttpServletResponse response,
            @NonNull FilterChain filterChain)
            throws ServletException, IOException {

        if (request.getRequestURI().matches("^/api/v1/(auth|public|im-
ages|csrf|community)/.*")) {
            filterChain.doFilter(request, response);
            return;
```

```
        }

        Optional<String> extractedJwt = jwtRequestExtractor.extractTo-
ken(request);
        if (extractedJwt.isPresent()) {
            String jwtToken = extractedJwt.get();
            TokenDTO tokenDTO = jwtServiceParserImpl.parseToken(jwtTo-
ken);
            if (tokenDTO.getUserName() != null && SecurityCon-
textHolder.getContext().getAuthentication() == null) {
                User userDetails = (User) userDetailsSer-
vice.loadUserByUsername(tokenDTO.getUserName());
                boolean hasValidRefreshToken =
                        tokenRepository.findAllValidToken-
ByUser(userDetails.getUserId())
                                .stream()
                                .filter(t1 -> !t1.isRevoked() &&
!t1.isExpired())
                                .anyMatch(t1 -> jwtServiceProvider.is-
ValidToken(t1.token, userDetails));
                if (jwtServiceProvider.isAccessToken(jwtToken, To-
kenScope.valueOf(tokenDTO.getTokenScope().name().toUpperCase())) &&
hasValidRefreshToken) {
                    setSecurityContext(request, userDetails);
                } else {
                   throw new InvalidJwtTokenException("Invalid token
scope, or expire token.");
                }
            }
        }

        filterChain.doFilter(request, response);
    }
```

Repository LaunchSpecification.java

```
package com.moonkeyeu.core.api.launch.repository.specifications;

public class LaunchSpecification {
    public static Specification<Launch> rootSpecification() {
        return (Root<Launch> root, CriteriaQuery<?> query, Crite-
riaBuilder builder) -> {
            if (query.getResultType() != Long.class &&
query.getResultType() != long.class) {
                root.fetch("launchStatus", JoinType.LEFT);
                root.fetch("agencies", JoinType.LEFT);
                root.fetch("launchPad", JoinType.LEFT)
                        .fetch("location", JoinType.LEFT);
                root.fetch("rocket", JoinType.LEFT)
                        .fetch("rocketConfiguration", JoinType.LEFT);
                root.fetch("missionPatches", JoinType.LEFT);
                root.fetch("videoUrls", JoinType.LEFT);
                root.fetch("infoUrls", JoinType.LEFT);
            }

            return builder.conjunction();
        };
    }
```

```
    public static Specification<Launch> hasLauncherSerialNumber(String
launcherId) {
        return (Root<Launch> root, CriteriaQuery<?> query, Crite-
riaBuilder builder) -> {
            Join<Launch, Rocket> rocketJoin = root.join("rocket",
JoinType.LEFT);
            Join<Rocket, LauncherStage> launcherStagesJoin = rocket-
Join.join("launcherStages", JoinType.LEFT);
            Join<LauncherStage, Launcher> launcherJoin = launcherStag-
esJoin.join("launcher", JoinType.LEFT);
            return builder.equal(launcherJoin.get("launcherId"),
launcherId);
        };
    }


    public static Specification<Launch> isUpcomingLaunch(Boolean up-
coming){
        return (Root<Launch> root, CriteriaQuery<?> query, Crite-
riaBuilder builder) -> {
            if(upcoming) {
                return builder.greaterThanOrEqualTo(root.get("net"),
Instant.now());
            }
            return builder.lessThanOrEqualTo(root.get("net"), In-
stant.now());
        };
    }
    public static Specification<Launch> hasRocketConfiguration(String
rocketConfigId){
        return (Root<Launch> root, CriteriaQuery<?> query, Crite-
riaBuilder builder) -> {
            Join<Object, Object> rocket = root.join("rocket",
JoinType.LEFT);
            Join<Object, Object> rocketConfiguration =
rocket.join("rocketConfiguration", JoinType.LEFT);
            return builder.equal(rocketConfiguration.get("rocketCon-
fId"), rocketConfigId);
        };
    }
    public static Specification<Launch> hasSpacecraftConfigura-
tion(String spacecraftConfigId){
        return (Root<Launch> root, CriteriaQuery<?> query, Crite-
riaBuilder builder) -> {
            Join<Object, Object> rocket = root.join("rocket",
JoinType.LEFT);
            Join<Object, Object> spacecraftStage = rocket.join("space-
craftStages", JoinType.LEFT);
            Join<Object, Object> spacecraft = space-
craftStage.join("spacecraft", JoinType.LEFT);
            Join<Object, Object> spacecraftConfiguration = space-
craft.join("spacecraftConfiguration", JoinType.LEFT);
            return builder.equal(spacecraftConfiguration.get("space-
craftConfId"), spacecraftConfigId);
        };
    }
    public static Specification<Launch> hasAgency(String agencyId) {
        return (Root<Launch> root, CriteriaQuery<?> query, Crite-
riaBuilder builder) -> {
            Join<Object, Object> agency = root.join("agencies",
JoinType.LEFT);
```

```
            return builder.equal(agency.get("agencyId"), agencyId);
        };
    }
    public static Specification<Launch> hasAstronaut(String astro-
nautId){
        return (Root<Launch> root, CriteriaQuery<?> query, Crite-
riaBuilder builder) -> {
            Join<Object, Object> rocket = root.join("rocket",
JoinType.LEFT);
            Join<Object, Object> spacecraftStage = rocket.join("space-
craftStages", JoinType.LEFT);
            Join<Object, Object> crewMembers = space-
craftStage.join("crewMembers", JoinType.LEFT);
            Join<Object, Object> astronaut = crewMembers.join("astro-
naut", JoinType.LEFT);
            return builder.equal(astronaut.get("astronautId"), astro-
nautId);
        };
    }
   public static Specification<Launch> hasLocation(String locationId){
        return (Root<Launch> root, CriteriaQuery<?> query, Crite-
riaBuilder builder) -> {
            Join<Object, Object> launchPad = root.join("launchPad",
JoinType.LEFT);
            Join<Object, Object> location = launchPad.join("location",
JoinType.LEFT);
            return builder.equal(location.get("locationId"), loca-
tionId);
        };
   }

   public static Specification<Launch> hasSearchKey(String key) {
       String searchKeyPattern = "%" + key.toLowerCase() + "%";
       return (root, query, cb) -> {
           Join<Launch, Agencies> agencyJoin = root.join("agencies",
JoinType.LEFT);
           Join<Launch, Rocket> rocketJoin = root.join("rocket",
JoinType.LEFT);
           Join<Launch, RocketConfiguration> rocketConfJoin = rocket-
Join.join("rocketConfiguration", JoinType.LEFT);
           Join<Launch, SpacecraftStage> spacecraftStageJoin = rocket-
Join.join("spacecraftStages", JoinType.LEFT);
           Join<Launch, Spacecraft>  spacecraftJoin = space-
craftStageJoin.join("spacecraft", JoinType.LEFT);
           Join<Launch, SpacecraftConfiguration> spacecraftConfJoin =
spacecraftJoin.join("spacecraftConfiguration", JoinType.LEFT);

           Predicate searchByName =
cb.like(cb.lower(root.get("launchName")), searchKeyPattern);
           Predicate searchBySlug =
cb.like(cb.lower(root.get("slug")), searchKeyPattern);

           Predicate searchByAgencyName = cb.like(cb.lower(agen-
cyJoin.get("agencyName")), searchKeyPattern);
           Predicate searchByRocketName = cb.like(cb.lower(rock-
etConfJoin.get("rocketName")), searchKeyPattern);
           Predicate searchByRocketVariant = cb.like(cb.lower(rock-
etConfJoin.get("variant")), searchKeyPattern);
           Predicate searchBySpacecraftConfName =
cb.like(cb.lower(spacecraftConfJoin.get("spacecraftConfName")),
searchKeyPattern);
```

```
            return cb.or(
                    searchByName,
                    searchBySlug,
                    searchByAgencyName,
                    searchByRocketName,
                    searchByRocketVariant,
                    searchBySpacecraftConfName
            );
        };
    }
}
```

Υπηρεσία LaunchServiceImpl.java

```
@Service
public class LaunchServiceImpl implements LaunchService {
    private final LaunchRepository launchRepository;
    private final DtoConverter dtoConverter;

    @Autowired
    public LaunchServiceImpl(DtoConverter dtoConverter, LaunchReposi-
tory launchRepository) {
        this.dtoConverter = dtoConverter;
        this.launchRepository = launchRepository;
    }

    @Cacheable(value = "launch-cache",  key = "'launch-pagination' +
#requestParams + #pageSortingDTO", sync = true)
    @Override
    public Page<DTOEntity> searchLaunch(Map<String, String> request-
Params, PageSortingDTO pageSortingDTO) throws NumberFormatException {
        Specification<Launch> spec = Specification.where(null);
        spec = spec.and(LaunchSpecification.rootSpecification());

        if (requestParams != null && !requestParams.isEmpty()) {

            if (requestParams.containsKey("launcher")) {
                String serialNumber = requestParams.get("launcher");
                spec = spec.and(LaunchSpecification.hasLauncherSerial-
Number(serialNumber));
            }

            if (requestParams.containsKey("spacecraftConfig")) {
                String spacecraft = requestParams.get("spacecraftCon-
fig");
                spec = spec.and(LaunchSpecification.hasSpacecraftCon-
figuration(spacecraft));
            }

            if (requestParams.containsKey("rocketConfig")) {
               String rocketConfiguration = requestParams.get("rock-
etConfig");
                spec = spec.and(LaunchSpecification.hasRocketConfigura-
tion(rocketConfiguration));
            }

            if (requestParams.containsKey("upcoming")) {
                Boolean upcoming = Boolean.valueOf(request-
Params.get("upcoming"));
```

```
                spec = spec.and(LaunchSpecification.isUpcom-
ingLaunch(upcoming));
                pageSortingDTO.setSort(upcoming ? "asc" : "desc");
            }

            if (requestParams.containsKey("astronaut")) {
                String astronaut = requestParams.get("astronaut");
                spec = spec.and(LaunchSpecification.hasAstronaut(as-
tronaut));
            }

            if (requestParams.containsKey("agency")) {
                String agency = requestParams.get("agency");
                spec = spec.and(LaunchSpecification.hasA-
gency(agency));
            }

            if (requestParams.containsKey("location")) {
                String astronaut = requestParams.get("location");
                spec = spec.and(LaunchSpecification.hasLocation(astro-
naut));
            }

            if (requestParams.containsKey("search")) {
                String searchKey = requestParams.get("search");
                spec = spec.and(LaunchSpecification.hasSearch-
Key(searchKey));
            }

        }
        Sort sortObject = "desc".equalsIgnoreCase(pageSorting-
DTO.getSort())
                    ? Sort.by(pageSortingDTO.getField()).descending()
                    : Sort.by(pageSortingDTO.getField()).ascending();

        int page = (pageSortingDTO.getPage() > 0) ? pageSorting-
DTO.getPage() - 1 : 0;
        Pageable pageable = PageRequest.of(page, pageSorting-
DTO.getLimit(), sortObject);
        Page<Launch> launches = launchRepository.findAll(spec, pagea-
ble);
        return launches.map(launch -> dtoConverter.convert-
ToDto(launch, LaunchNormalDTO.class));
    }
    @Cacheable(value = "launch-cache",  key = "'launch-' + #launchId",
sync = true)
    public Optional<DTOEntity> getLaunchById(String launchId) {
        Optional<Launch> launch = launchRepository.findLaunchWith-
LaunchId(launchId);
        return launch.map(l -> dtoConverter.convertToDto(l,
LaunchDTO.class));
    }

}
```