ΔΙΕΘΝΕΣ
ΠΑΝΕΠΙΣΤΗΜΙΟ
ΤΗΣ ΕΛΛΑΔΟΣ

ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΚΑΙ ΗΛΕΚΤΡΟΝΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

«Υλοποίηση πλατφόρμας Blogging για περιηγητές και κινητές συσκευές με λειτουργικό Android.»

Του φοιτητή
Δημήτριου Τζήκα
Αρ. Μητρώου: 175042

Επιβλέπων
Κυριάκος Τσιακμάκης
Επίκουρος Καθηγητής

Ιούνιος 2026

Τίτλος Π.Ε. Υλοποίηση πλατφόρμας Blogging για περιηγητές και κινητές συσκευές με λειτουργικό Android.

Κωδικός Π.Ε. 25169

Ονοματεπώνυμο φοιτητή Δημήτριος Τζήκας

Ονοματεπώνυμο εισηγητή Κυριάκος Τσιακμάκης

Ημερομηνία ανάληψης Π.Ε. 11-03-2025

Ημερομηνία περάτωσης Π.Ε. 1-06-2025

Βεβαιώνω ότι είμαι ο συγγραφέας αυτής της εργασίας και ότι κάθε βοήθεια την οποία είχα για την προετοιμασία της είναι πλήρως αναγνωρισμένη και αναφέρεται στην εργασία. Επίσης, έχω καταγράψει τις όποιες πηγές από τις οποίες έκανα χρήση δεδομένων, ιδεών, εικόνων και κειμένου, είτε αυτές αναφέρονται ακριβώς είτε παραφρασμένες. Επιπλέον, βεβαιώνω ότι αυτή η εργασία προετοιμάστηκε από εμένα προσωπικά, ειδικά ως διπλωματική εργασία, στο Τμήμα Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων του ΔΙ.ΠΑ.Ε.

Η παρούσα εργασία αποτελεί πνευματική ιδιοκτησία του φοιτητή Δημήτριος Τζήκας που την εκπόνησε. Στο πλαίσιο της πολιτικής ανοικτής πρόσβασης, ο συγγραφέας/δημιουργός εκχωρεί στο Διεθνές Πανεπιστήμιο της Ελλάδος άδεια χρήσης του δικαιώματος αναπαραγωγής, δανεισμού, παρουσίασης στο κοινό και ψηφιακής διάχυσης της εργασίας διεθνώς, σε ηλεκτρονική μορφή και σε οποιοδήποτε μέσο, για διδακτικούς και ερευνητικούς σκοπούς, άνευ ανταλλάγματος. Η ανοικτή πρόσβαση στο πλήρες κείμενο της εργασίας, δεν σημαίνει καθ' οιονδήποτε τρόπο παραχώρηση δικαιωμάτων διανοητικής ιδιοκτησίας του συγγραφέα/δημιουργού, ούτε επιτρέπει την αναπαραγωγή, αναδημοσίευση, αντιγραφή, πώληση, εμπορική χρήση, διανομή, έκδοση, μεταφόρτωση (downloading), ανάρτηση (uploading), μετάφραση, τροποποίηση με οποιονδήποτε τρόπο, τμηματικά ή περιληπτικά της εργασίας, χωρίς τη ρητή προηγούμενη έγγραφη συναίνεση του συγγραφέα/δημιουργού.

Η έγκριση της διπλωματικής εργασίας από το Τμήμα Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων του Διεθνούς Πανεπιστημίου της Ελλάδος, δεν υποδηλώνει απαραιτήτως και αποδοχή των απόψεων του συγγραφέα, εκ μέρους του Τμήματος.

«Στους αγαπημένους μας»

Πρόλογος

Η επιλογή του συγκεκριμένου θέματος πηγάζει από το ενδιαφέρον μου να ασχοληθώ με τη δημιουργία full stack εφαρμογών αλλά και τη δημιουργία scalable εφαρμογών. Αν και δεν μπόρεσε να δημιουργηθεί, λόγο του χρονικού πλαισίου, ένα σύστημα το οποίο θα κάνει χρήση τεχνολογιών όπως kubernetes για το καλύτερο scalability και για την καλύτερη αποδοτικότητα της εφαρμογής σε υπηρεσίες όπως AWS και Azure. Παρόλα αυτά, η εφαρμογή χρησιμοποιεί ένα μεγάλο εύρος τεχνολογιών και καταφέρνει να είναι μια σύγχρονη και ασφαλή εφαρμογή. Η υλοποίηση με τη χρήση αυτών των τεχνολογιών εμφάνισε αρκετές προκλήσεις όπου είχαν ως αποτέλεσμα τον επανασχεδιασμό τμημάτων της εφαρμογής αλλά και τον προβληματισμό για το πόσο είναι εφικτό να κάνει κάποιος μια ασφαλή εφαρμογή που δε θα θυσιάζει την εμπειρία του χρήστη.

Περίληψη

Στην παρούσα πτυχιακή εργασία πραγματοποιήθηκε η σχεδίασης και η ανάπτυξη μιας ασφαλούς και σύγχρονης full stack εφαρμογής κοινωνικής δικτύωσης η οποία μιμείται κάποια χαρακτηριστικά των Blogging ιστοσελίδων. Ένα σημαντικό χαρακτηριστικό της εφαρμογής είναι την ασφάλεια από τη σχεδίασης (security by design) για την αντιμετώπιση συχνών ευπαθειών όπως XSS, CSRF και Phishing.

Για τη σχεδίαση της εφαρμογής έγινε η χρήση της αρχιτεκτονικής BFF (Backend for Frontend). Για γραφικά τμήματα της ιστοσελίδας χρησιμοποιήθηκαν τεχνολογίες όπως Vue 3, Vite, PrimeVue4 και Tailwind CSS.

Με σκοπό την ανάπτυξη της εφαρμογή των κινητών συσκευών android χρησιμοποιήθηκε η Kotlin Multiplatform με compose multiplatfrom. Αυτό επέτρεψε τον διαμοιρασμό κώδικα μεταξύ διαφορετικών πλατφορμών. Επιπλέον, έγινε η χρήση offline caching με την ενσωμάτωση της τοπικής Room KMP βάση δεδομένων, με τη στρατηγική fail closed για τη δυναμική αναγέννηση κλειδιών.

Ένα σημαντικό σημείο που δόθηκε βαρύτητα είναι η υβριδική αρχιτεκτονική αυθεντικοποίησης η οποία περιλαμβάνει την τοπική αυθεντικοποίηση με υποστήριξη για την πολυπαραγοντική αυθεντικοποίηση χρησιμοποιώντας φυσικό κλειδί. Επιπλέον, υπάρχει και η δυνατότητα για σύνδεση μέσω Google OAuth 2.0 με επαλήθευση από τη μεριά του διακομιστή.

«Implementation of a blogging platform for web browsers and Android mobile devices»

«Dimitrios Tzikas»

Abstract

In this thesis, the design and development of a secure and modern full stack was carried out which imitates some features of Blogging websites. An important feature of it is security by design to address common vulnerabilities such as XSS, CSRF and Phishing.

For its design, the BFF (Backend for Frontend) architecture was used. For graphical parts of the website, technologies such as Vue 3, Vite, PrimeVue4 and Tailwind CSS were used.

In order to develop the application for Android mobile devices, Kotlin Multiplatform was used with compose multiplatform. This allowed code sharing between different platforms. In addition, offline caching was used with the integration of the local KMP database, with the fail-closed strategy for dynamic key regeneration.

An important point that was given importance is the hybrid authentication architecture which includes local authentication with support for multifactor authentication using a physical key. Additionally, there is the option to connect via Google OAuth 2.0 with server-side retry.

Ευχαριστίες

Με την ολοκλήρωση της παρούσας πτυχιακής εργασίας, θα ήθελα να ευχαριστήσω όλα τα άτομα που με βοήθησαν να την τελειώσω είτε συνέλαβαν άμεσα ή έμμεσα στην εκπόνηση της.

Επίσης, θα ήθελα να πω ένα μεγάλο ευχαριστώ για την κατανόηση και την υποστήριξη που έδειξε η οικογένεια μου και οι φίλοι.

Περιεχόμενα

Κατάλογος Σχημάτων

Κατάλογος Φωτογραφιών

Κατάλογος Πινάκων

Συντομογραφίες

MFA	Multi-Factor Authentication
2FA	Two-Factor Authentication
OTP	One-Time Password
XSS	Cross-Site Scripting
CSRF	Cross-Site Request Forgery
TLS	Transport Layer Security
WebAuthn	Web Authentication
FIDO2	Fast Identity Online 2
OAuth	Open Authorization
CWE	Common Weakness Enumeration
OWASP	Open Web Application Security Project
HMAC	Hash-based Message Authentication Code
SHA	Secure Hash Algorithm
MD5	Message Digest 5
ECDSA	Elliptic Curve Digital Signature Algorithm
BFF	Backend-for-Frontend
SPA	Single Page Application
API	Application Programming Interface
JVM	Java Virtual Machine
DOM	Document Object Model
HMR	Hot Module Replacement
ESM	ECMAScript Modules
ES6	ECMAScript 6
DSL	Domain Specific Language
XML	Extensible Markup Language
CSS	Cascading Style Sheets
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
I/O	Input/Output

CPS	Continuation-Passing Style
URL	Uniform Resource Locator
SQL	Structured Query Language
ORM	Object Relational Mapping
DAO	Data Access Object
JSON	JavaScript Object Notation
JSONB	JavaScript Object Notation Binary
CSV	Comma-Separated Values
CBOR	Concise Binary Object Representation
JWK	JSON Web Key
PEM	Privacy-Enhanced Mail
UUID	Universally Unique Identifier
GiST	Generalized Search Tree
BTree	Binary Tree
VARCHAR	Character Varying
BIGINT	Big Integer
RFC	Request for Comments
IETF	Internet Engineering Task Force
ISO	International Organization for Standardization
ASCII	American Standard Code for Information Interchange
SIM	Subscriber Identity Module
USB	Universal Serial Bus

Κεφάλαιο 1ο: Εισαγωγή

Η εξελίξεις στον τομέα της πληροφορικής έχει μεταβάλει με ριζικό τρόπο τον τρόπο με τον οποίο λειτουργεί η σημερινή κοινωνία. Μέσα σε αυτό είναι και ο παγκόσμιος ιστός, ο οποίος αποτελεί ένα κομμάτι του σημερινού ανθρώπου. Πλέον, για όποιο ζήτημα και να αφορά κάποιον θα υπάρχει μια ιστοσελίδα που να σχετίζεται με αυτό, από κρατικές ιστοσελίδες για διεκπεραίωση κρατικών υποχρεώσεων αλλά και πληροφοριών, έως και για ιστοσελίδες για άμεση παράδοση παιχνιδιών για κατοικίδια. Κάτι το οποίο όμως παρατηρήθηκε είναι πως ο σύγχρονος άνθρωπος έχει πλέον την ικανότητα να ταξιδεύει περισσότερο από ποτέ. Η σημερινές γενιές ταξιδεύουν πιο πολύ από όλες τις προηγούμενες, για αυτό είχαμε την ιδέα για το θέμα αυτής της πτυχιακής.

Το TravelTales δημιουργήθηκε για να προσφέρει στους χρήστες του τη δυνατότητα να γράψουν για τα ταξίδια τους και τις εμπειρίες τους με τη μορφή ιστοριών. Επιπλέον, ο χρήστης μπορεί να δημιουργεί οδηγούς όπου βασίζονται στην προσωπική του εμπειρία για να βοηθήσει άλλους ταξιδιώτες. Το TravelTales προσφέρει ένα φιλικό, όμορφο και γρήγορο γραφικό περιβάλλον, ενθαρρύνοντας τη χρήση της εφαρμογής και την αλληλεπίδραση με άλλους χρήστες, μέσω σχολίων, αξιολογήσεις και ανταλλαγή απόψεων για προορισμούς σε όλο τον κόσμο.

Κάθε εφαρμογή εκτός από τις προκλήσεις που έχει για την ανάπτυξη ενός περιβάλλοντος που έχει καλή εμπειρία χρήστη αλλά και την ολοκλήρωση του σκοπού της, υπάρχουν και προκλήσεις στον τομέα της ασφάλειας του χρήστη και των πληροφοριών του. Για αυτόν τον λόγο κάθε σύγχρονη εφαρμογή έχει πάντα ως βασικούς πυλώνες τη διασφάλιση της ακεραιότητας, των δεδομένων και την αποτροπή κακόβουλων ενεργειών. Λόγο αυτού, το TraveTales σχεδιάστηκε και υλοποιήθηκε με σκοπό να παρέχει ασφάλεια από τον σχεδιασμό (security by design). Για τον σκοπό αυτό χρησιμοποιήθηκαν προηγμένες αρχιτεκτονικές προσεγγίσεις και κρυπτογραφικά πρότυπα.

1.1 Το πρόβλημα και το κίνητρο

Η πλειονότητα των περισσότερων μικρών διαδικτυακών εφαρμογών βασίζονται σε απλούς μηχανισμούς για την ταυτοποίηση του χρήστη, όπως όνομα και κωδικό πρόσβασης. Η προσέγγιση αυτή παρουσιάσει πολλαπλά προβλήματα. Ο μέσος χρήστης τείνει να χρησιμοποιεί αδύναμους κωδικούς αλλά και να τους επαναλαμβάνει σε πολλαπλές υπηρεσίες [1]. Λόγο αυτής της φύσεων των χρηστών, η χρήση μόνο ενός κωδικού είναι ευάλωτη σε επιθέσεις brute force και phishing. Η πιο συνηθισμένη υλοποίηση για να γίνει πιο ασφαλές το σύστημα είναι η χρήση πολυπαραγοντικής αυθεντικοποίησης (Multi factor authentication - MFA), όπου χρησιμοποιούν κωδικούς μίας χρήσης (one time password - OTP) οι οποίοι στέλνονται στον τελικό χρήστη μέσω email ή sms. Αλλά ακόμη και αυτή η υλοποίηση είναι ευάλωτη σε επιθέσεις SIM swapping και main in the middle. Μια επιπλέον ευπάθεια που υπάρχει σε ιστοσελίδες που χρησιμοποιούν SPAs (single page application) και JWT tokens, είναι ότι αποθηκεύουν τα tokens στην τοπική μνήμη του browser με αποτέλεσμα να είναι επιρρεπή σε επιθέσεις XSS (cross site scripting).

Το κίνητρο για την εκπόνηση της παρούσας πτυχιακής εργασίας είναι η δημιουργία μιας σύγχρονης ιστοσελίδας και εφαρμογής, όπου παρέχει καλή εμπειρία χρήστη αλλά και λύνει όσα περισσότερα θέμα ασφάλειας μπορεί. Για αυτόν τον λόγο η πτυχιακή εστιάζει στην αρχιτεκτονική προτύπου Backend for Frontend (BFF). Μέσω αυτής της αρχιτεκτονικής, η διαχείριση και αποθήκευση πληροφοριών και tokens που αποτελούν ευαίσθητες πληροφορίες αποθηκεύονται και μεταφέρονται με τρόπο έτσι ώστε να μειώνεται η επιφάνεια επίθεσης αλλά και η εξάλειψη γνωστών επιθέσεων.

Επιπλέον, η εργασία στοχεύει στη χρήση του WebAuthn το οποίο χρησιμοποιείται για βιομετρική αυθεντικοποίηση και MFA (multi factor authentication). Μέσω της οποία γίνεται χρήση ασύμμετρης κρυπτογραφίας σε φυσικό επίπεδο, το οποίο εξαλείφει πιθανές επιθέσεις όπως phishing.

1.2 Μεθοδολογία ανάπτυξης

Για την ανάπτυξη της εφαρμογής χρησιμοποιήθηκαν μεθοδολογίες που βασίζονται στη σε σύγχρονα πρότυπα και βέλτιστες πρακτικές ασφάλειας. Ένα σημαντικό χαρακτηριστικό είναι η ασφάλεια από τον σχεδιασμό, δεν προστέθηκε ως εκ των υστέρων, αλλά αποτελεί βασικό χαρακτηριστικό της σχεδίασης της εφαρμογής. Πολλά από τα πρότυπα που χρησιμοποιήθηκαν είναι οδηγίες από τον OWASP (Open Web Application Security Project) για την αντιμετώπιση των σημαντικότερων απειλών. Τέλος, ο σχεδιασμό του συστήματος έγινε με modular τμήματα τα οποία βοηθούν στη διασφάλιση της αναγνωσιμότητας, τη συντηρησιμότητας και την εύκολη επέκταση του κώδικα.

Κεφάλαιο 2ο: Θεωρητικό Υπόβαθρο και Τεχνολογίες

Στο παρών κεφάλαιο αναλύονται οι τεχνολογίες που επιλέχθηκαν για τη σχεδίαση και την υλοποίηση της full stack εφαρμογής TravelTales. Κατά τη σχεδίαση και επιλογή εργαλείων, γνώμονας ήταν η ασφάλεια, η υψηλή απόδοση, η δυνατότητα διαχείρισης γεωχωρικών δεδομένων μεγάλης κλίμακας και η ευελιξία κατά την ανάπτυξη της εφαρμογής.

2.1 Ασύγχρονος προγραμματισμός και τα πλεονεκτήματα της Kotlin και του Ktor.

2.1.1 Η γλώσσα Kotlin και στην ανάπτυξη συστημάτων backend\

Η Kotlin είναι μια σύγχρονη γλώσσα προγραμματισμού γενικής χρήσης, η οποία αναπτύχθηκε από την εταιρία JetBrains, η οποία τη σχεδίασε να είναι πλήρως διαλειτουργική με τη γλώσσα Java και το Java Vritual Machine (JVM).

Κάποιες από τις σημαντικότερες βελτιώσεις σε σύγκριση με τη Java, είναι η ασφάλεια που προσφέρει έναντι τιμών null. Αυτό το επιτυγχάνει το σύστημα τύπων την Kotlin, ορίζοντας ρητά τις μεταβλητές που μπορούν να πάρουν την τιμή null από εκείνες που δεν μπορούν. Με αυτόν τον τρόπο, τυχόν σφάλματα τύπου "NullPointerException", μπορούν να εντοπιστούν κατά το στάδιο της μεταγλώττισης του κώδικα. Ένα ακόμη μεγάλο πλεονέκτημα που θα παρατηρήσει ο προγραμματιστής είναι πως στην Kotlin, δεν υπάρχει η πολυλογία της Java. Αυτό το πετυχαίνει με διάφορους τρόπους, όπως αφαιρώντας λέξεις σε σημεία που εννοούνται, χρήση data classes, smart casts και extension functions. Μια από τις μεγαλύτερες βελτιώσεις είναι τα coroutines, τα οποία βελτιώνουν δραματικά τις επιδόσεις. Τα coroutines, επιτρέπουν την εκτέλεση λειτουργιών εισόδου/εξόδου παράλληλα (για λειτουργίες εισόδου/εξόδου που δεν εξαρτύονται η μια από την άλλη). Αυτό εξασφαλίζει την καλύτερη εκμετάλλευση των νημάτων του επεξεργαστή, χωρίς να υπάρχει κάποια ιδιαίτερη διαχείριση από τον προγραμματιστή.

2.1.2 Ktor Framework

Το Ktor είναι ένα ασύγχρονο, ελαφρύ framework το οποίο αναπτύχθηκε από την εταιρία JetBrains, μετά από μια μελέτη τους για να δουν πως χρησιμοποιούταν η γλώσσα Kotlin. Τα αποτελέσματα έδειξαν πως το 47% [2] των προγραμματιστών τη χρησιμοποιούσαν για web backend. Για αυτόν τον λόγο σχεδιάστηκε για να είναι αποδοτικό σε δικτυακές εφαρμογές και microservices. Σε αντίθεση με "παραδοσιακά" frameworks του οικοσυστήματος του JVM (όπως Spring Boot), το Ktor ακολουθεί τη φιλοσοφία του micro-kernel architecture. Σύμφωνα με αυτήν την αρχιτεκτονική ένας πυρήνας θα πρέπει να υλοποιεί μόνο τα απολύτως απαραίτητα για να λειτουργήσει και τυχόν έξτρα λειτουργίες θα πρέπει να προστίθενται στο user space. Ο κυριότερος τρόπος που έχει υλοποιηθεί αυτό στο Ktor, είναι με τα plugins, στα οποία θα αναφερθούμε πιο κάτω.

Αν συγκρίνουμε το Ktor με τη Spring Boot θα παρατηρήσουμε τα παρακάτω. Το Ktor, χρησιμοποιεί αισθητά λιγότερη μνήμη και έχει γρηγορότερη εκκίνηση, το οποίο είναι σημαντικό για μια auto-scalable microservices. Έχει ασύγχρονη φύση (Non-blocking I/O), καθώς πατάει πάνω στο Netty ή σε παρόμοια engines. Το οποίο το καθιστά ικανό να διαχειριστή χιλιάδες ταυτόχρονες συνδέσεις. Τέλος, η ρυθμίσεις γίνονται μέσα από type-safe Domain Specific Language (DSL) και όχι σε XML. Το οποίο το καθιστά πιο εύκολο στη ρύθμιση του και σε ενημερώσεις.

2.2 Μοντέρνα οικοσυστήματα frontend

2.2.1 Το Vue.js framework και το composition API

Για την υλοποίηση του ιστότοπου επιλέχθηκε το framework Vue.js, έκδοση 3. Το Vue.js είναι ένα προοδευτικό framework, το οποίο χρησιμοποιείται για την ανάπτυξη Single Page Applications (SPAs). Συγκεκριμένα επιλέχθηκε η έκδοση 3 έναντι της 2, λόγο του composition API και τις καλύτερες επιδόσεις, καθώς χρησιμοποιεί λιγότερους πόρους. Ειδικότερα, μπορείς πιο εύκολα να επαναχρησιμοποιήσεις τη λογική των συστατικών, αυτό συμβαίνει γιατί επιτρέπεται η ομαδοποίηση της λογικής των συστατικών ανά feature και όχι ανά επιλογή (option), οι οποίες ήταν data, methods, computed. Κάνοντας πιο εύκολη τη δημιουργία custom composables. Υπάρχει καλύτερη υποστήριξη για type script (αν και δε χρησιμοποιήθηκε στο TravelTales) και καλύτερο static analysis. Και μια από τις μεγαλύτερες, αν όχι η μεγαλύτερη βελτίωση, είναι το Virutal DOM. Σε σύγκριση με το παλιό Virtual DOM, χρησιμοποιεί έναν ταχύτερο αλγόριθμο για reconciliation της εικονικής αναπαράστασης του εγγράφου HTML, ελαχιστοποιώντας τις δαπανηρές λειτουργίες που εκτελούνται απευθείας στο DOM του browser.

Συγκρίνοντας το Vue 3 με React και Angular, ο προγραμματιστής θα παρατηρήσει πως είναι αισθητά πιο εύκολη η εκμάθηση και η χρήση της Vue 3. Ένας μεγάλος λόγος είναι η αυτόματη ανίχνευση αλλαγών μέσω ES6 Proxies, έτσι ο προγραμματιστής δε χρειάζεται να αναπτύξει χειροκίνητες βελτιστοποιήσεις, όπως useMemo και useCallback που χρησιμοποιούνται από React. Επιπλέον, η χαλαρή και λιτή αρχιτεκτονική της Vue την κάνει εύκολη στην ανάπτυξη και συντήρηση εφαρμογών σε σύγκριση με τη React και Angular.

2.2.2 Το εργαλείο κατασκευής Vite

Το Vite είναι ένα σύγχρονο εργαλείο κατασκευής το οποίο στοχεύει στη βελτιώσει της εμπειρίας ανάπτυξης ιστοσελίδων. Συγκεκριμένα, είναι ένας τοπικός εξυπηρετητής ο οποίος δημιουργήθηκε από την εταιρία VoidZero Inc. Τα πλεονεκτήματα του Vite είναι πως κατά τη διαδικασία ανάπτυξης της ιστοσελίδας, δεν κάνει bundle τον κώδικα. Αντιθέτως, σερβίρει τον πηγαίο κώδικα απευθείας, χρησιμοποιώντας το Native ES Modules (ESM) του browser, χαρίζοντας ακαριαία εκκίνηση του διακομιστή. Για τα εξαρτήματα που δεν είναι υπανάπτυξη, το vite, χρησιμοποιεί το Esbuild, το οποίο έχει γραφτεί σε go, το οποίο είναι ταχύτερο από παραδοσιακούς bundlers όπως είναι το Webpack. Τέλος, μια από τις μεγαλύτερες βελτιώσεις στην εμπειρία του προγραμματιστή, έρχεται από το Hot Module Replacement (HMR), το οποίο ενημερώνει ακαριαία την ιστοσελίδα όταν ο προγραμματιστής κάνει αλλαγές και τις αποθηκεύσει, χωρίς να χάσει την κατάσταση της εφαρμογής, με εξαιρέσεις όπου ο προγραμματιστής θα αναγκαστεί να κάνει ανανέωση την κατάσταση της εφαρμογής.

2.2.3 PrimeVue 4 και Tailwind CSS

Για την ανάπτυξη των γραφικών της ιστοσελίδας χρησιμοποιήθηκαν δύο κορυφαίες τεχνολογίες CSS. Το Tailwind CSS, το οποίο παρέχει χαμηλού επιπέδου utility classes έναντι ad-hoc CSS κανόνων. Με αυτόν τον τρόπο εξασφαλίζει συνεκτικότητα στο design και ελαχιστοποιεί το μέγεθος που έχει το τελικό παραγόμενο αρχείο CSS, αφαιρώντας κώδικα που δε χρησιμοποιείται. Τη βιβλιοθήκη PrimeVue 4, που παρέχει γραφικά αντικείμενα, τα οποία μπορούν να τροποποιηθούν αλλά και να επεκταθούν. Επιπλέον, στην έκδοση 4 ενσωματώθηκε πλήρως το Tailwind CSS, επιτρέποντας την τροποποίηση των γραφικών αντικειμένων με τη χρήση των utility classes του Tailwind.

2.3 Διαχείριση δεδομένων και σχεσιακές βάσεις δεδομένων

2.3.1 Postgresql και Γεωχωρικές δυνατότητες

Η αντικειμενοστραφής σχεσιακή βάση δεδομένων PostgreSQL, είναι από τις πιο γνωστές βάσεις δεδομένων, λόγο της αξιοπιστίας της, την ακεραιότητα των δεδομένων που προσφέρει και για την υποστήριξη προτύπων SQL.

Στο πλαίσιο της εφαρμογής TravelTales, η PostgreSQL χρησιμοποιήθηκε λόγο της υποστήριξης των JSON/JSONB, όπου επιτρέπουν την αποθήκευση ημι-δομημένων δεδομένων και την ταχεία προσπέλαση αυτών. Ένας δεύτερος λόγος, είναι η ανάγκη για γεωχωρικά δεδομένα. Αν και η PostgreSQL δεν υποστηρίζει απευθείας γεωχωρικά δεδομένα, μπορεί κάποιος να προσθέσει αυτή τη λειτουργία προσθέτοντας το PostGIS. Με το PostGIS μπορεί κάποιος να κάνει ερωτήματα που σχετίζονται με γεωγραφικές αποστάσεις και γεωμετριών στην υδρόγειο σφαίρα. Ένας επιπλέον σημαντικός λόγος, είναι τα ευρετήρια της PostgreSQL. Υποστηρίζει μια μεγάλη γκάμα από ευρετήρια όπως, BTree, Hash και GiST, τα οποία είναι απαραίτητα για ταχύτατες αναζητήσεις, ειδικά όταν κάνουμε αναζήτηση σε γεωγραφικές συντεταγμένες. Αξίζει να σημειωθεί αν και είναι χρήσιμα τα ευρετήρια σε δεδομένα που ενημερώνονται συχνά μπορεί ένα ευρετήριο να μεγαλώσει γρήγορα ξεπερνώντας των όγκο των ίδιων δεδομένων. Σε τέτοιες περιπτώσεις ο διαχειριστής της βάσης θα πρέπει να καθαρίζει (vacuum) τη βάση σε τακτά χρονικά διαστήματα.

2.3.2 Exposed Object Relational Mapping

Το Exposed ORM, είναι ένα ελαφρύ framework το οποίο είναι γραμμένο σε Kotlin, δημιουργήθηκε και συντηρείται από την ίδια τη JetBrains και προσφέρει πρόσβαση στα δεδομένα με δύο διαφορετικούς τρόπους. Με Exposed Domain Specific Language, όπου είναι type safe και μπορεί να ελέγχονται κατά τη διαδικασία της μεταγλώττισης. Και με Exposed Data Access Object, η οποία είναι μια αντικειμενοστραφής προσέγγιση για τη διαχείριση των εγγραφών ως Kotlin αντικείμενα, η οποία μοιάζει αρκετά με το Hibernate. Σε αντίθεση με το Hibernate, το Exposed DAO, μπορεί να αποτρέψει σφάλματα τύπων κατά τη διαδικασία της μεταγλώττισης, δε χρησιμοποιεί βαριά XML αρχεία και πολύπλοκα annotations. Το καλύτερο όμως, είναι η υποστήριξη των coroutines της Kotlin, όπου επιτρέπει ασύγχρονες συναλλαγές χωρίς να μπλοκάρει το κεντρικό νήμα της εφαρμογής.

2.4 Ταυτοποίηση

Η εφαρμογή TravelTales προσφέρει δύο τρόπους ταυτοποίησης, ένας είναι με όνομα χρήστη και κωδικό (επιπλέον ο χρήστης μπορεί να προσθέσει ταυτοποίηση δύο παραγόντων βάζοντας ένα hardware key), και ο δεύτερος με τη χρήση Google authentication.

2.4.1 JSON Web Tokens (JWT)

Το JSON Web Tokens χρησιμοποιείται για τη μεταφορά πληροφοριών ως αντικείμενο JSON, όπως ορίζεται στο ανοιχτό πρότυπο RFC 7519. Στην εφαρμογή TravelTales χρησιμοποιείται για τη stateless ταυτοποίηση των χρηστών.

Ένα JWT αποτελείται από 3 τμήματα τα οποία χωρίζονται από μία τελεία. Το πρώτο είναι η κεφαλίδα, η οποία καθορίζει βασικά χαρακτηριστικά του JWT, όπως τον τύπο του και τον αλγόριθμο που χρησιμοποιεί για την κρυπτογράφηση. Το δεύτερο τμήμα είναι το ωφέλιμο φορτίο, το οποίο το ορίζει ο προγραμματιστής και μπορεί να περιέχει πληροφορίες όπως, όνομα χρήστη, επίπεδο πρόσβαση και ημερομηνία λήξης σε unix time. Το τελευταίο τμήμα είναι η ψηφιακή υπογραφή, η οποία δημιουργείται

κρυπτογραφώντας την κωδικοποιημένη κεφαλίδα με το ωφέλιμο φορτίο σε κωδικοποιημένη μορφή (base64). Η κρυπτογράφηση γίνεται με το μυστικό κλειδί που χρησιμοποιεί το backend. Με αυτόν τον τρόπο μπορεί να γνωρίζεις ο παραλήπτης πως το περιεχόμενο του JWT δεν έχει υποστεί τροποποίηση από τρίτους.

Με αυτόν τον τρόπο η εφαρμογή είναι stateless και δεν υπάρχει η ανάγκη για διαχείριση sessions από τον διακομιστή. Αν και είναι αρκετά θετικό για όταν υπάρχουν πολλοί πελάτες, δημιουργεί την ανικανότητα να λήξη το JWT πριν παρέλθει ο χρόνος του. Οπότε, για εφαρμογές όπου θέλουμε να υπάρχει η δυνατότητα να λήξη δυναμικά η ταυτοποίηση ενός χρήστη, δεν είναι καλή επιλογή.

2.4.2 Google OAuth 2.0

Το Google OAuth 2.0 είναι η υλοποίηση της Google του πρωτοκόλλου 0Auth 2.0, το οποίο χρησιμοποιείται για να παρέχει τη δυνατότητα να συνδεθεί ένας χρήστης σε μια εφαρμογή χωρίς να δώσει όνομα χρήστη και κωδικό πρόσβασης. Με αυτόν τον τρόπο ένας χρήστης μπορεί να συνδέεται σε πολλαπλούς λογαριασμούς χρησιμοποιώντας μόνο τα διαπιστευτήρια της Google.

Στην εφαρμογή TravelTales, όταν ο χρήστης πατάει το κουμπί "σύνδεση με google account", τότε η ιστοσελίδα κάνει redirect στο backend, μετά το backend με τη σειρά του κάνει redirect στην google. Όταν ο χρήστης δώσει τα στοιχεία του και συνδεθεί, η google κάνει callback το backend και μετά το backend κάνει callback το frontend. Αυτή η διαδικασία αν ήταν να την περιγράψουμε με κάποιο pattern, αυτό θα ήταν το Backend For Frontend (BFF), όπως ορίζεται από τον οργανισμό Internet Engineering Task Force (IETF) στο "draft ietf oauth browser based apps" στην ενότητα 6 [3]. Κατά το οποίο υπάρχει ένα backend για κάθε frontend, το οποίο κάνει όλα τα απαραίτητα calls που χρειάζεται. Αν και χρησιμοποιείται κατά κύριο λόγο σε συστήματα που χρησιμοποιούν microservices έτσι ώστε να βελτιστοποιήσει την απόκριση του frontend καθώς και να μειώσει την πολυπλοκότητα του.

Κεφάλαιο 3ο: Ανάλυση απαιτήσεων και σχεδιασμός συστήματος

Σε αυτή την ενότητα αναλύεται ο τρόπος που σχεδιάστηκε το TravelTales και κάποιες από τις απαιτήσεις της.

3.1 Λειτουργικές απαιτήσεις

Με τον όρο λειτουργικές απαιτήσεις, αναφερόμαστε στις συγκεκριμένες υπηρεσίες, συμπεριφορές και λειτουργίες που παρέχει ένα σύστημα (στο πλαίσιο της πτυχιακής η εφαρμογή TravelTales) στους τελικούς χρήστες του. Όσο αναφορά την εφαρμογή TravelTales, η ανάλυση των απαιτήσεων έγινε με γνώμονα δύο πράγματα. Πρώτον, με τον ρόλο (unverified, verified, moderator, admin) που μπορεί να έχει κάθε χρήστης και το πως επηρεάζει τον τρόπο που αλληλεπιδρά με την εφαρμογή. Δεύτερον, με τη χρήση λεπτομερών σεναρίων χρήσης (use cases), όπου περιγράφουν με λεπτομέρεια τη συμπεριφορά της εφαρμογή και τη ροή που πρέπει να ακολουθήσει ο χρήστης για να ολοκληρώσει το κάθε σενάριο.

3.1.1 Ανάλυση ρόλων χρηστών

Για τη σωστή λειτουργία της εφαρμογής και την καλύτερη εμπειρία του χρήστη, το σύστημα διαθέτει πέντε διαφορετικούς ρόλους unsigned, unverified, verified, moderator και admin.

Ο unsigned ρόλος, δεν είναι ένας διακριτός ρόλος στο σύστημα, αλλά με αυτόν αναφερόμαστε στον χρήστη ο οποίος δεν έχει λογαριασμό. Ένας χρήστης ο οποίος δεν έχει λογαριασμό, έχει την πιο βασική και περιορισμένη χρήση από όλους τους υπόλοιπους χρήστες, καθώς μπορεί μόνο να διαβάσει το περιεχόμενο που προσφέρει η εφαρμογή χωρίς να μπορεί να αλληλεπιδράσει με άλλους χρήστες ή να δημιουργήσει νέο περιεχόμενο (εξαίρεση είναι η δημιουργία λογαριασμού).

Ο unverified ρόλος είναι ο προεπιλεγμένος ρόλος για τη δημιουργία λογαριασμού (άσχετα με τον αν είναι τοπικός ή Google 0Auth). Αυτός ο ρόλος προσθέτει τη δυνατότητα ο χρήστης να μπορεί να αλληλεπιδράσει με άρθρα και άλλους χρήστες, να αλλάζει τις ρυθμίσεις στον λογαριασμό του, καθώς παρέχει και τη δυνατότητα ο χρήστης να δημιουργήσει λίστες με τα αγαπημένα του άρθρα ή άρθρα που θα διαβάσει αργότερα.

Ο verified ρόλος, που θα μπορούσε να ονομάζεται και συγγραφέας. Είναι ο βασικός ρόλος της εφαρμογής και όλες οι βασικές λειτουργίες της έχουν σχεδιαστή με γνώμονα αυτόν τον ρόλο. Δίνει στο χρήστη τη δυνατότητα να δημιουργήσει περιεχόμενο (Stories ή Guides) και να ανεβάζει πολυμέσα.

Ο moderator ρόλος, ο οποίος είναι διαχειριστικός ρόλος καθώς δίνει τη δυνατότητα ένας χρήστης να μπορεί να αλλάξει τον ρόλο άλλων χρηστών, για ρόλους που είναι χαμηλότερα στην ιεραρχία των ρόλων.

Τέλος, ο admin ρόλος, ο οποίος έχει προνομιακή πρόσβαση στο σύστημα. Είναι υπεύθυνος για την εύρυθμη λειτουργία της εφαρμογής, τη διαχείριση των κατηγοριών και τον έλεγχο καταχρήσεων. Ο admin μπορεί να διαγράψει περιεχόμενο ή χρήστες που παραβιάζουν τους κανόνες της εφαρμογής.

3.1.2 Σενάρια χρήσης (use case)

Για την καλύτερη κατανόηση της ροής των εργασιών που μπορεί να κάνει ένας χρήστης στην εφαρμογή, παρατίθενται τέσσερα βασικά σενάρια χρήσης που περιγράφουν κάποιες από τις σημαντικές λειτουργίες της εφαρμογής.

Για το πρώτο σενάριο θα αναφερθούμε στην εγγραφή και ταυτοποίηση του χρήστη (Sign up & Login). Αυτό το σενάριο αφορά έναν επισκέπτη της εφαρμογής ο οποίος θέλει να δημιουργήσει έναν λογαριασμό και κατόπιν να συνδεθεί στην εφαρμογή. Ο επισκέπτης πρέπει να έχει πρόσβαση στο διαδίκτυο και να βρίσκεται ή να πλοηγηθεί στη σελίδα σύνδεσης/εγγραφής. Κατόπιν αν επιλέξει τη δημιουργία λογαριασμού, πρέπει να συμπληρώσει το όνομα χρήστη, ένα email και έναν κωδικό πρόσβασης. Όταν πατήσει το κουμπί για τη δημιουργία λογαριασμού, στέλνονται τα στοιχεία στο backend το οποίο με τη σειρά του ελέγχει αν το username και το email χρησιμοποιούνται από κάποιον άλλων χρήστη. Στην περίπτωση που υπάρχουν, στέλνει στο frontend πως τα στοιχεία χρησιμοποιούνται από κάποιον άλλων χρήστη. Αν δε χρησιμοποιούνται ελέγχει το πόσο ισχυρός είναι κωδικός και στην περίπτωση που δεν είναι αρκετά ισχυρός στέλνει το ανάλογο μήνυμα. Αν όλα τα στοιχεία που έδωσε ο χρήστης πληρούν τις προϋποθέσεις, τότε το κρυπτογραφείται μονόδρομα και δημιουργείται μια νέα εγγραφή στον πίνακα app.users με τα στοιχεία που έδωσε ο χρήστης. Στη συνέχεια εκδίδεται ένα JWT access token το οποίο αποθηκεύεται σε HttpOnly cookie, σύμφωνα με τον οδηγό του OWASP για τη διαχείριση συνεδριών [4], το οποίο αποτρέπει scripts να αποκτήσουν πρόσβαση στο cookie όπως δείχνει το CWE-1004 [5]. Εναλλακτικά ο χρήστης αν δε θέλει να συνδεθεί με τοπικό λογαριασμό μπορεί να συνδεθεί με Google OAuth. Αυτό μπορεί να το κάνει επιλέγοντας σύνδεση μέσω Google, όταν το πατήσει το frontend κάνει redirect στο backend, το οποίο με τη σειρά του κάνει στην google, όπου ο χρήστης μπορεί να συνδεθεί και μόλις ολοκληρώσει τη σύνδεση τότε κάνει η google callback στο backend το οποίο ταυτοποιεί τα στοιχεία και δημιουργεί τον λογαριασμό αν δεν υπάρχει και κατόπιν κάνει και αυτό callback στο frontend στέλνοντας και το JWT access token, ακριβώς όπως γίνεται και στη σύνδεση με τοπικό λογαριασμό.

Το δεύτερο σενάριο αφορά τη συγγραφή και δημοσίευση ταξιδιωτικής ιστορίας. Ο χρήσης θα πρέπει να έχει τουλάχιστον ρόλο verified και να έχει έγκυρο JWT session. Κατόπιν, ο χρήστης μπορεί να μεταβεί στη σελίδα των ταξιδιωτικών ιστοριών και να επιλέξει δημιουργία νέας ταξιδιωτικής ιστορίας. Εφόσον μεταβεί στη σελίδα, τότε μπορεί να συμπληρώσει τον τίτλο, την περιγραφή της ιστορίας και το πλήρες κείμενο σε μορφή markdown. Στη συνέχεια μπορεί να επιλέξει μία ή περισσότερες θεματικές κατηγορίες από το check box μενού. Επιλέγει τη χώρα και πόλη (τα οποία φορτώνονται δυναμικά από το API γεωγραφικών δεδομένων) που αφορά την ταξιδιωτική ιστορία. Προαιρετικά μπορεί να ανεβάσει φωτογραφίες (οι οποίες χρησιμοποιούνται για προβολή στην περισκόπηση της ιστορίας), όπου ανεβαίνουν στον διακομιστή και επιστρέφουν ένα URL. Όταν ο χρήστης έχει ολοκληρώσει τη συγγραφή μπορεί να πατήσει δημοσίευση και το σύστημα δημιουργεί μια εγγραφή στον πίνακα app.stories. Μόλις ολοκληρωθεί η δημιουργία, όλοι οι χρήστες της εφαρμογής μπορούν να δουν και να διαβάσουν τη γεωγραφική ιστορία, μεταβαίνοντας στις γεωγραφικές ιστορίες και επιλέγοντας την ιστορία που δημιούργησε ο χρήστης.

Το τρίτο σενάριο αφορά την ενεργοποίηση ταυτοποίησης δύο παραγόντων. Ο χρήστης εφόσον είναι συνδεδεμένος στην εφαρμογή και χρησιμοποιεί τοπικό λογαριασμό, μπορεί να πάει στις ρυθμίσεις του προφίλ του και να ενεργοποιήσει το 2FA με φυσικό κλειδί ασφάλειας USB (στα πλαίσια αυτής της εφαρμογής χρησιμοποιήθηκαν μόνο προϊόντα της Yubico, αλλά θεωρητικά θα λειτουργούν κλειδιά και από άλλες εταιρίες εφόσον υποστηρίζουν το πρωτόκολλο FIDO2 [6] ή WebAuthn [7]). Για να το ενεργοποιήσει επιτυχώς πρέπει να βάλει ένα κλειδί όπου κατά τη διαδικασία προσθήκης του κλειδιού, το backend δημιουργεί μια τυχαία κρυπτογραφική πρόκληση και την αποστέλλει στο fronend. Το fronend με τη σειρά του καλεί το WebAuthn API του περιηγητή, ζητώντας με αυτόν τον τρόπο από τον χρήστη να αγγίξει το κλειδί ασφάλειας έτσι ώστε να δημιουργηθεί στο κλειδί ένα public/private key pair. Με αυτά τα κλειδιά υπογράφει την πρόκληση και την επιστρέφει μαζί με το δημόσιο κλειδί. Κατόπιν το fronend στέλνει τα δεδομένα στο backend, το οποίο επαληθεύει την υπογραφή και

αποθηκεύει το δημόσιο κλειδί στον πίνακα webauthn_credentials και θέτει το πεδίο is_two_factor_enabled σε αληθές. Όταν έχει ολοκληρωθεί αυτή η διαδικασία, για τις επόμενες συνδέσεις ο χρήστης θα πρέπει να βάζει τον κωδικό πρόσβασης, καθώς και να επιβεβαιώνει την ταυτότητα του χρησιμοποιώντας το καταχωρημένο φυσικό κλειδί ασφάλειας USB.

Το τέταρτο και τελευταίο σενάριο χρήσης αφορά τη διαχείριση κατηγοριών από τον admin. Ο ρόλος του χρήστη θα πρέπει να είναι admin και να είναι συνδεδεμένος με ένα έγκυρο JWT token. Στη συνέχεια ο χρήστης μπορεί να πάει στο admin panel και να μεταβεί στη διαχείριση κατηγοριών. Εκεί μπορεί να συμπληρώσει το όνομα μιας νέας κατηγορίας και να δώσει ένα μοναδικό κωδικό. Όταν το κάνει αυτό θα μπορεί να πατήσει δημιουργία και στη συνέχεια το backend θα δημιουργήσει μια νέα εγγραφή με την κατηγορία στον πίνακα app.story_categories. Επιπλέον, ο χρήστης μπορεί να διαγράψει υπάρχουσες κατηγορίες. Η διαγραφή υπάρχουσας κατηγορίας έχει ως αποτέλεσμα να αφαιρεθεί από ταξιδιωτικές ιστορίες ή οδηγούς. Τα αποτελέσματα αυτών των ενεργειών είναι άμεσα αισθητά.

3.2 Μη λειτουργικές απαιτήσεις

Με τον όρο μη λειτουργικών απαιτήσεων αναφερόμαστε σε απαιτήσεις που καθορίζουν ποιοτικά κριτήρια λειτουργίας, περιορισμούς ασφάλειας, αποδώσεις και τη συντηρησιμότητα της εφαρμογής.

Κατά τη σχεδίασης της εφαρμογής TravelTales, δόθηκε μεγάλη έμφαση στην ασφάλεια του. Για αυτόν τον λόγο μελετήθηκαν και χρησιμοποιήθηκαν προηγμένες τεχνικές θωράκισης καθώς και προτροπές από οργανισμούς όπως τον OWASP.

3.2.1 Ασφάλεια και ιδιωτικότητα

Ξεκινώντας από την επικοινωνία του πελάτη προς τον εξυπηρετητή αλλά και μεταξύ των εξυπηρετητών. Χρησιμοποιείται το πρωτόκολλο TLS 1.3, το οποίο αποτρέπει επιθέσεις υποκλοπής δεδομένων από man in the middle. Στη συνέχεια δόθηκε έμφαση σε επιθέσεις όπως το cross site scripting (XSS) [8], για αυτόν τον λόγο δε χρησιμοποιήθηκε το local storage του περιηγητή. Για να υπάρχει μεγαλύτερη ασφάλεια για τα JWT tokens και για να μη χρησιμοποιηθεί το local storage, χρησιμοποιήθηκαν κάποια flags. Κάποια από αυτά είναι, το HttpOnly, το οποίο αποτρέπει την πρόσβαση του token από οποιοδήποτε script. Το secure, το οποίο διασφαλίζει ότι το cookie θα αποστέλλεται μόνο μέσω κρυπτογραφημένων HTTPS συνδέσεων. Τελευταίο είναι το SameSite=Lax/Strict, το οποίο παρέχει ισχυρή προστασία έναντι επιθέσεων cross site request forgery (CSRF).

Επιπρόσθετα, η αρχιτεκτονική passwordless και FIDO2 για τους τοπικούς λογαριασμούς, μπορεί να προστατέψει τον χρήστη από επιθέσεις όπως phishing attacks ή υποκλοπή γνωστικών στοιχείων, καθώς προσθέτει στην ταυτοποίηση του χρήστη το επίπεδο κρυπτογράφησης από υλικό. Με αυτόν τον τρόπο ο χρήστης κατέχει δύο γνωρίσματα, κάτι το οποίο γνωρίζει και κάτι το οποίο κατέχει. Αξίζει να σημειωθεί πως έγινε επιπλέον μελέτη για την προσθήκη ταυτοποίησης βάση τοποθεσίας και βάση συμπεριφοράς, όπου μπορούν να χρησιμοποιηθούν ως έξτρα τρόποι επιβεβαίωσης της ταυτοποίησης του χρήστη αλλά δεν υλοποιήθηκαν λόγο του μικρού χρονικού πλαισίου.

3.2.2 Απόδοση συστήματος και διαχείριση πόρων

Η εφαρμογή TravelTales αν και δεν έχει σχεδιαστεί για να είναι πλήρως scalable (εφόσον ακολουθεί ένα μονολιθικό μοντέλο), έχει σχεδιαστεί για να τρέχει αποδοτικά ακόμα και σε περιβάλλοντα με περιορισμένους πόρους (όπως φθηνά cloud instances).

Ένα μεγάλο πλεονέκτημα είναι η ασύγχρονη non-blocking I/O αρχιτεκτονική. Αυτό το χαρακτηριστικό σε αντίθεση με thread-per-request αρχιτεκτονικές (όπου κάθε σύνδεση δεσμεύει ένα λειτουργικό νήμα όπου είναι σε επίπεδο λειτουργικού συστήματος, το οποίο συνήθως είναι περίπου 1MB) είναι πιο αποδοτικό και σε χρονική απόκριση αλλά και σε χωρητικότητα μνήμης. Συγκεκριμένα το Ktor, το επιτυγχάνει με τη χρήση του Netty (το οποίο είναι το προεπιλεγμένο engine) και των coroutines της Kotlin.

Μιλώντας γενικά για την αρχιτεκτονική της Kotlin για τα Coroutines, ο προγραμματιστής θα πρέπει να έχει κάποια πράγματα στο μυαλό του. Το πρώτο είναι πως η Kotlin, επιτυγχάνει τις καλύτερες επιδώσεις στα νήματα τρέχοντας ταυτόχρονα (να σημειωθεί πως όταν λέμε ταυτόχρονα δε σημαίνει πως εκτελούνται παράλληλα αλλά η εκτέλεσης τους γίνεται σε επικαλυπτόμενα χρονικά διαστήματα). Αυτό το κάνει χρησιμοποιώντας το στυλ προγραμματισμού continuation-passing style (CPS) [9], το οποίο είναι γνωστό κατά κύριο λόγο στον συναρτησιακό προγραμματισμό. Έχοντας αυτά στο νου, μπορούμε να δούμε πως μπορεί εύκολα ένας κακογραμμένος κώδικας που κρατάει το runtime, να δημιουργήσει runtime starvation. Αυτό συμβαίνει γιατί δεν υπάρχει κάποιος scheduler, όπως θα είναι σε ένα νήμα σε επιπέδου λειτουργικού συστήματος, όπου ο scheduler κάνει preemptive scheduling. Σαφώς, μια διεργασία που κρατάει το runtime, δεν είναι σωστά σχεδιασμένη αλλά το πρόβλημα είναι πως σε ένα τέτοιο σενάριο, είναι δύσκολος ο εντοπισμός του προβλήματος λόγο του ότι θα υπάρχουν μόνο latency spikes και hangs.

3.2.3 Συντηρησιμότητα και επεκτασιμότητα

Για να εξασφαλιστεί η συντηρησιμότητα και η επεκτασιμότητα, χρησιμοποιήθηκε το μοτίβο clean architecture & Service-Repository. Βάση αυτού το μοτίβου, το backend έχει διαχωρίσει πλήρως τους πίνακες, τα αντικείμενα πρόσβασης (DAOs), των αποθετηρίων (Repositories) και των υπηρεσιών (Services). Με αυτόν τον τρόπο μελλοντικά αν χρειαστεί να γίνει η αλλαγή της βάσης δεδομένων από PostreSQL σε MongoDB ή σε κάποια άλλη, θα χρειαστεί να γραφτούν μόνο τα αποθετήρια, ενώ το business logic (υπηρεσίες) και τα Routes θα παραμείνουν απολύτως ανέπαφα.

3.3 Αρχιτεκτονική συστήματος

Η αρχιτεκτονική της εφαρμογής TravelTales βασίζεται στο μοντέλο 3-Tier Decoupled Architecture, το οποίο εξασφαλίζει την πλήρη ανεξαρτησία μεταξύ της διεπαφής (Vue.js), της εφαρμογής (Ktor) και της βάσης δεδομένων (PostgreSQL).

3.3.1 Επίπεδο παρουσίασης

Το επίπεδο παρουσίασης αφορά το frontend, το οποίο είναι αναπτυγμένο σε Vue.js και χρησιμοποιεί τεχνολογίες όπως το Vue Router 4, ο οποίος αναλαμβάνει το client-side routing. Χρησιμοποιεί Navigation Guards για να ελέγχει αν ο χρήστης έχει τα απαραίτητα δικαιώματα πρόσβασης σε προστατευόμενες σελίδες όπως το Admin Panel. Το οποίο εξασφαλίζει πως αν ο χρήστης αποσυνδεθεί ή το cookie ταυτοποίησης έχει λήξει, θα ανακατευθυνθεί αυτόματα στη σελίδα login. Για την επικοινωνία του frontend με το backend χρησιμοποιείται ο client Axios HTTP. Ο Axios έχει ρυθμιστεί έτσι ώστε να συμπεριλαμβάνει αυτόματα τα HttpOnly cookies ταυτοποίησης σε κάθε εξερχόμενο HTTP αιτήματος προς το backend (για να γίνει η ρύθμιση αυτή χρειάζεται απλώς να τεθεί η παράμετρος withCredentials = true). Επίσης, έχει χρησιμοποιηθεί και ο συνδυασμός PrimeVue 4 και Tailwind CSS, όπου προσφέρουν ένα ενοποιημένο σύστημα σχεδίασης γραφικών διεπαφών. Η χρήση των utility κλάσεων που παρέχει το Tailwind σε συνδυασμό με τα PrimeVue θέματα, δίνει τη δυνατότητα να

υπάρχει δυναμική εναλλαγή μεταξύ φωτεινού και σκοτεινού θέματος, καθώς εξασφαλίζει responsive layouts που προσαρμόζονται με αυτόματο τρόπο σε οποιοδήποτε μέγεθος οθόνης.

3.3.2 Επίπεδο εφαρμογής

Το επίπεδο εφαρμογής αφορά το backend, το οποίο είναι είναι ένας Ktor διακομιστής που εκτελείται πάνω στο JVM. Εσωτερικά του Ktor, η επεξεργασία αιτημάτων ξεκινάει από το HTTP αίτημα, στη συνέχεια ελέγχεται από το CORS, μετά γίνεται η πιστοποίηση του JWT (στην περίπτωση που χρειάζεται), μετά περνάει στις υπηρεσίες και στο repository logic και τέλος καταλήγει στο REST Endpoint. Πιο αναλυτικά, το CORS (Cross Origin Resource Sharing) είναι ένας μηχανισμός που υλοποιείται από τους περιηγητές για να αποτρέψει επιθέσεις όπως cross-site attacks. Ελέγχει αν το HTTP αίτημα είναι από εγκεκριμένη προέλευση (οι προελεύσεις μπορούν να οριστούν στις ρυθμίσεις του plugin). Στην περίπτωση που η προέλευση του αιτήματος δεν είναι από εγκεκριμένη προέλευση, τότε το αίτημα δεν εξυπηρετείται και επιστρέφεται ένα μήνυμα λάθους για CORS violation. Η πιστοποίηση του JWT γίνεται με το plugin Authentication και Authorization. Το οποίο ελέγχει αν η υπογραφή του JWT είναι σωστή, χρησιμοποιώντας το μυστικό κλειδί, επιπλέον ελέγχει αν το χρήστης έχεις τα απαιτούμενα δικαιώματα. Τέλος, υπάρχει και το StatusPage plugin, το οποίο λειτουργεί ως κεντρικός μηχανισμός διαχείρισης σφαλμάτων. Στη περίπτωση που κάποια υπηρεσία βγάλει σφάλμα τότε επιστρέφει το κατάλληλο HTTP status code σε μορφή JSON (η μορφή της απάντησης καθορίζεται από το AutoNegotiation plugin), με αυτόν τον τρόπο δεν θα επιστρέψει ποτέ ευαίσθητα stack traces στον τελικό χρήστη.

3.3.3 Επίπεδο δεδομένων

Για την υλοποίηση του επιπέδου δεδομένων, χρησιμοποιήθηκε η βάση δεδομένων PostgreSQL. Για την διασύνδεση με το επίπεδο εφαρμογής και την διαχείριση των δεδομένων, χρησιμοποιήθηκε το JetBrains exposed ORM.

Η διαχείριση των συναλλαγών με τη βάση δεδομένων γίνεται μέσα από ένα ελεγχόμενο transaction context. Πιο αναλυτικά το exposed, παρέχει συναρτήσεις όπως η suspendTransaction, η οποία μεταφέρει την εκτέλεση των ερωτημάτων sql σε ένα εξειδικευμένο thread (το οποίο είναι σε επίπεδο λογισμικού) το οποίο επιλέγεται από το dispacthers.IO των coroutines, το οποίο διαχειρίζεται ένα thread pool. Με αυτόν τον τρόπο αποφεύγουμε το να μπλοκάρει το κύριο νήμα της εφαρμογής.

Κατά την αρχικοποίηση του συστήματος, πρέπει να γίνει ένα seeding στη βάση δεδομένων έτσι ώστε να έχει τα απαραίτητα δεδομένα. Αυτό περιλαμβάνει ένα data set (cities500.json [10]) το οποίο είναι ένα κατάλογος πόλεων με πληθυσμό μεγαλύτερο από 500 άτομα. Με αυτόν τον τρόπο ο χρήστης αντί να γράψει την πόλη που αφορά η ιστορία του, τη διαλέγει μέσα από ένα μενού.

3.4 Σχεδιασμός βάσης δεδομένων

Για να υπάρχει ακεραιότητα των δεδομένων και να εξασφαλιστεί η ταχύτητα εκτέλεσης σύνθετων συσχετίσεων, σχεδιάστηκε ένα αυστηρά ορισμένο σχεσιακό σχήμα.

3.4.1 Αναλυτική περιγραφή πινάκων βάση δεδομένων

Ακολουθούν οι πίνακες που χρησιμοποιήθηκαν για την αποθήκευση δεδομένων της εφαρμογής TravelTales.

Πίνακας 1: app.users (Αποθήκευση Στοιχείων Χρηστών)

Αποθηκεύει τα διαπιστευτήρια, τα στοιχεία προφίλ και την κατάσταση ασφάλειας των χρηστών.

Όνομα Πεδίου	Τύπος Δεδομένων	Περιορισμοί (Constraints)	Περιγραφή / Χρήση
id	UUID	Primary Key, Default: uuid_generate_v4()	Μοναδικό αναγνωριστικό χρήστη.
username	VARCHAR(50)	Unique, Not Null	Μοναδικό όνομα χρήστη για αναφορά και URL.
password	VARCHAR(150)	Nullable	Κρυπτογραφημένος κωδικός (null για Google OAuth).
email	VARCHAR(50)	Unique, Not Null	Διεύθυνση ηλεκτρονικού ταχυδρομείου.
is_local_user	BOOLEAN	Not Null	true για τοπική εγγραφή, false για Google Auth.
picture	VARCHAR(255)	Nullable	URL της φωτογραφίας προφίλ του χρήστη.
banner_picture	VARCHAR(255)	Nullable	URL της εικόνας εξωφύλλου του προφίλ.
role	VARCHAR(20)	Not Null, Default: 'READER'	Ρόλος συστήματος (ADMIN, WRITER, READER, UNVERIFIED).
first_name	VARCHAR(100)	Nullable	Όνομα χρήστη.
last_name	VARCHAR(100)	Nullable	Επώνυμο χρήστη.
is_two_factor_enabled	BOOLEAN	Not Null, Default: false	Ένδειξη αν έχει ενεργοποιηθεί 2FA (WebAuthn).
bio	TEXT	Nullable	Σύντομο βιογραφικό σημείωμα χρήστη.
created_at	BIGINT	Not Null	Timestamp δημιουργίας λογαριασμού (Epoch milliseconds).

updated_at	BIGINT	Not Null	Timestamp τελευταίας τροποποίησης.

Πίνακας 2: app.stories (Ταξιδιωτικές Ιστορίες)

Αποθηκεύει τις ιστορίες που συγγράφουν οι χρήστες, μαζί με τις γεωγραφικές τους σημάνσεις.

Όνομα Πεδίου	Τύπος Δεδομένων	Περιορισμοί (Constraints)	Περιγραφή / Χρήση
id	UUID	Primary Key	Μοναδικό αναγνωριστικό ιστορίας.
title	VARCHAR(255)	Not Null	Τίτλος της ταξιδιωτικής ιστορίας.
preview_content	TEXT	Nullable	Σύντομο κείμενο προεπισκόπησης για τις κάρτες.
full_content	TEXT	Not Null	Το πλήρες σώμα της ιστορίας σε μορφή Markdown.
author_username	VARCHAR(50)	Foreign Key -> app.users(username)	Ο συγγραφέας της ιστορίας.
categories	TEXT	Nullable	Σειριοποιημένη λίστα (JSON/CSV) κατηγοριών.
created_at	BIGINT	Not Null	Timestamp δημιουργίας (Epoch milliseconds).
updated_at	BIGINT	Not Null	Timestamp τελευταίας επεξεργασίας.
published	BOOLEAN	Not Null, Default: true	Κατάσταση δημοσίευσης (draft αν είναι false).
images	TEXT	Nullable	Σειριοποιημένα URLs φωτογραφιών της ιστορίας.
city_id	UUID	Foreign Key -> cities(id), Nullable	Συσχετισμένη πόλη προορισμού.
country_id	UUID	Foreign Key -> countries(id), Nullable	Συσχετισμένη χώρα προορισμού.

Πίνακας 3: countries (Χώρες)

Αποθηκεύει τις χώρες του κόσμου για τη γεωγραφική ταξινόμηση.

Όνομα Πεδίου	Τύπος Δεδομένων	Περιορισμοί (Constraints)	Περιγραφή / Χρήση
id	UUID	Primary Key	Μοναδικό αναγνωριστικό χώρας.
name	VARCHAR(255)	Not Null	Επίσημο όνομα χώρας.
code	VARCHAR(2)	Unique, Not Null	ISO 2-letter code της χώρας (π.χ. "GR", "US").

Πίνακας 4: cities (Πόλεις)

Αποθηκεύει τις πόλεις του κόσμου, τις συντεταγμένες τους και τον πληθυσμό τους.

Όνομα Πεδίου	Τύπος Δεδομένων	Περιορισμοί (Constraints)	Περιγραφή / Χρήση
id	UUID	Primary Key	Μοναδικό αναγνωριστικό πόλης.
name_en	VARCHAR(255)	Not Null	Όνομα της πόλης με λατινικούς χαρακτήρες (ASCII).
name_gr	VARCHAR(255)	Nullable	Όνομα της πόλης με ελληνικούς χαρακτήρες.
lat	DOUBLE PRECISION	Not Null	Γεωγραφικό πλάτος (Latitude) της πόλης.
lng	DOUBLE PRECISION	Not Null	Γεωγραφικό μήκος (Longitude) της πόλης.
feature_class	VARCHAR(10)	Not Null	Κατηγοριοποίηση Geonames (π.χ. "P" για κατοικημένη περιοχή).
feature_code	VARCHAR(10)	Not Null	Ειδικότερος κωδικός Geonames.
population	BIGINT	Not Null	Πληθυσμός της πόλης (χρήσιμο για ταξινόμηση/φίλτρα).

timezone	VARCHAR(100)	Not Null	Ζώνη ώρας της πόλης (π.χ. "Europe/Athens").
country_id	UUID	Foreign Key -> countries(id), Not Null	Η χώρα στην οποία ανήκει η πόλη.

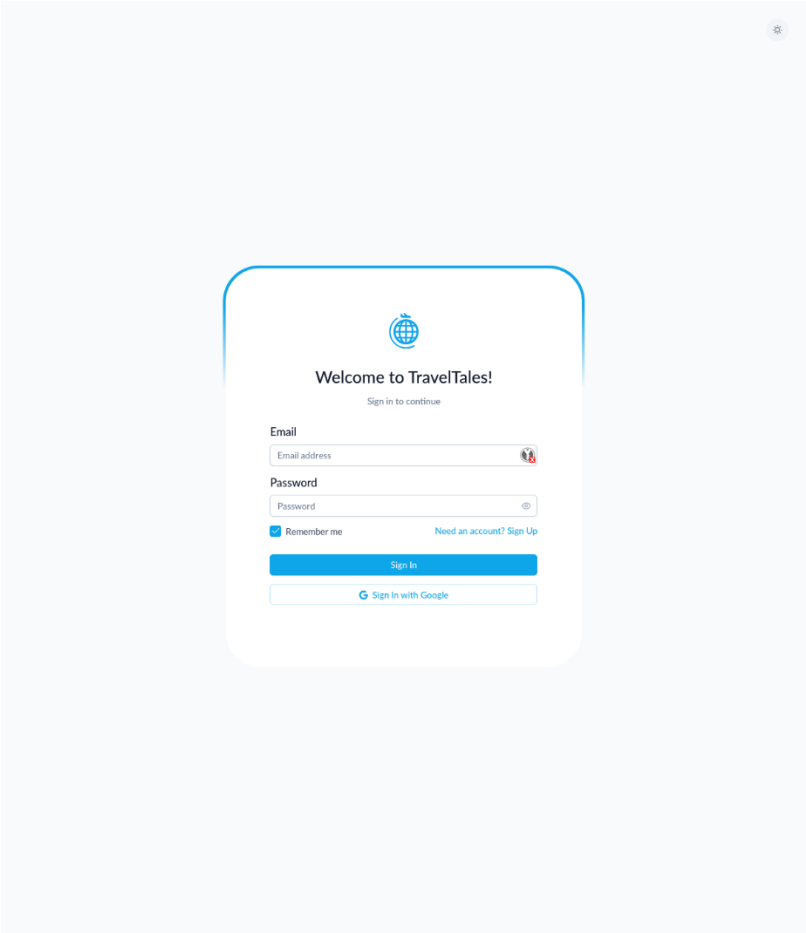
Πίνακας 5: webauthn_credentials (Διαπιστευτήρια 2FA)

Αποθηκεύει τα κρυπτογραφικά κλειδιά για την ταυτοποίηση δύο παραγόντων των χρηστών.

Όνομα Πεδίου	Τύπος Δεδομένων	Περιορισμοί (Constraints)	Περιγραφή / Χρήση
id	UUID	Primary Key	Μοναδικό αναγνωριστικό διαπιστευτηρίου.
user_id	UUID	Foreign Key -> app.users(id), Not Null	Ο χρήστης στον οποίο ανήκει το κλειδί.
credential_id	VARCHAR(255)	Unique, Not Null	Το αναγνωριστικό που παράγει ο browser/security key.
public_key	TEXT	Not Null	Το Δημόσιο Κρυπτογραφικό Κλειδί (PEM/JWK format).
sign_counter	BIGINT	Not Null, Default: 0	Μετρητής υπογραφών για την αποτροπή replay attacks.

3.4.2 Στρατηγική χρήση ευρετηρίων και ακεραιότητα δεδομένων

Για να εξασφαλιστεί η μέγιστη δυνατή ταχύτητα και αξιοπιστία κατά την εκτέλεση ερωτημάτων, χρησιμοποιήθηκαν ευρετήρια δυαδικών δέντρων. Με αυτόν τον τρόπο εξασφαλίζουμε πως όταν θα γίνεται η αναζήτηση χρηστών κατά τη διαδικασία του login, η οποία ολοκληρώνεται σε χρόνο O(logn), ο οποίος είναι σχεδόν ακαριαίος, ανεξαρτήτως από το μέγεθος της βάσης. Επιπλέον, χρησιμοποιήθηκαν δύο περιορισμοί αναφοράς ξένων κλειδιών (Foreign key referential integrity). Ο ένας είναι τύπου on delete cascade, όπου χρησιμοποιήθηκε στον πίνακα app.webauthn_credentials. Με αυτόν τον τρόπο εξασφαλίζουμε πως στην περίπτωση που γίνει διαγραφή ενός χρήστη ο οποίος χρησιμοποιούσε hardware key, θα γίνει αυτόματα η διαγραφή του κλειδιού από τη βάση, αποφεύγοντας με αυτόν τον τρόπο "ορφανές" εγγραφές. Αντίστοιχα, για τον πίνακα app.stories χρησιμοποιήθηκε το on delete set null/restrict, όπου στην περίπτωση που γίνει διαγραφή μίας χώρας ή πόλης, δε δείχνουν τα πεδία city_id και country_id σε δεδομένα που δεν υπάρχουν, αντιθέτως γίνεται η τιμή τους null. Τέλος, μπορεί να γίνει χρήση σύνθετων ευρετηρίων για τη δραστική επιτάχυνση των ερωτημάτων αναζήτησης τοποθεσιών βάσει γεωγραφικού εύρους τιμών, για τον πίνακα cities στα πεδία lat (latitude) και lng (longitude).



Equation 1

Κεφάλαιο 4ο: Υλοποίηση Frontend

Κάθε διεπαφή χρήστη αποτελεί τον συνδετικό κρίκο μεταξύ του τελικού χρήστη και των εσωτερικών μηχανισμών που κάνουν την εφαρμογή να δουλεύει. Με τη χρήση της αρχιτεκτονική Single Page Application (SPA), η αυθεντικοποίηση και η διαχείριση συνεδριών, συχνά εκθέτουν ευαίσθητα δεδομένα, όπως access tokens, καθιστώντας την εφαρμογή ευάλωτη σε επιθέσεις υποκλοπής δεδομένων. Για την αποφυγή τέτοιων επιθέσεων, η υλοποίηση του frontend έγινε με γνώμονα το σύγχρονο αρχιτεκτονικό πρότυπο BFF (Backend For Frontend). Με αυτόν τον τρόπο μετατρέπουμε το fontend σε ένα stateless δέκτη που αλληλεπιδρά με το backend αποκλειστικά μέσω προστατευμένων και κρυπτογραφημένων cookies.

Το παρόν κεφάλαιο αναλύει τις τεχνολογίες που χρησιμοποιήθηκαν για τον σχεδιασμό και υλοποίηση του frontend. Δίνει βαρύτητα στους μηχανισμούς διαχείρισης συνεδριών, στην υλοποίηση του reactive state management για την ταυτοποίηση του χρήστη και τις τεχνικές λεπτομέρειες των για την ταυτοποίηση με τη χρήση φυσικού κλειδιού (WebAuthn).

4.1 Αρχιτεκτονική δομή και τεχνολογίες

Η επιλογή των τεχνολογιών έγινε με γνώμονα την επεκτασιμότητα, την ασφάλεια, την απόδοση και τη συμβατότητα με σύγχρονα πρότυπα ανάπτυξης λογισμικού.

4.1.1 Vue 3 και composition API

Η Vue 3 επιλέχθηκε λόγο της εύκολης συντήρησης και της απλής λογικής που παρέχει. Ειδικά με την έκδοση 3 η οποία εισήγαγε τη μηχανή reactivty. Η reactivity είναι μια ριζικά επανασχεδιασμένη μηχανή η οποία βασίστηκε σε ES6 proxies. Αυτή η αλλαγή εισήγαγε τον καλύτερο και αποδοτικότερο εντοπισμό αλλαγών σε σύνθετες δομές δεδομένων (όπως εμφωλευμένα αντικείμενα και πίνακες) με πολύ λιγότερο υπολογιστικό κόστος. Για τον προγραμματιστή αυτό σημαίνει πως δε χρειάζεται να χειρίζεται ο ίδιος τις αλλαγές που τυχόν συμβαίνουν στα δεδομένα.

Για την ανάπτυξη των τμημάτων χρησιμοποιήθηκε μόνο το composition API και το συντακτικό πρότυπο "<-script setup>". Με αυτήν την προσέγγιση αυτή υπάρχει πιο καθαρός κώδικας, καθώς δεν υπάρχει η ανάγκη για τη χρήση αυστηρών μπλοκ (όπως data, methods, computed, mounted) όπου χρησιμοποιούνται από το options API, αλλά μπορούν να οργανωθούν γύρω από συγκεκριμένες λειτουργίες. Ο κώδικας μπορεί να γίνει πιο επαναχρησιμοποιήσιμος με τη χρήση του composability, το οποίοι επιτρέπει τη δημιουργία αυτόνομων συναρτήσεων. Για παράδειγμα, η συνάρτηση "useAuth", μπορεί να χρησιμοποιηθεί σε οποιοδήποτε σημείο της εφαρμογής, χωρίς περιορισμού όπως υπήρχε με το options API. Επιπλέον, ο compiler της Vue προσφέρει βελτιστοποιήσεις, όπου μπορεί να αναλύσεις στατικά τα πρότυπα και να ανυψώσει (hoisting) τα στατικά στοιχεία, μειώνοντας σημαντικά τον χρόνο που χρειάζεται για να επανασχεδιαστεί το εικονικό DOM.

4.1.2 Vite και bundler

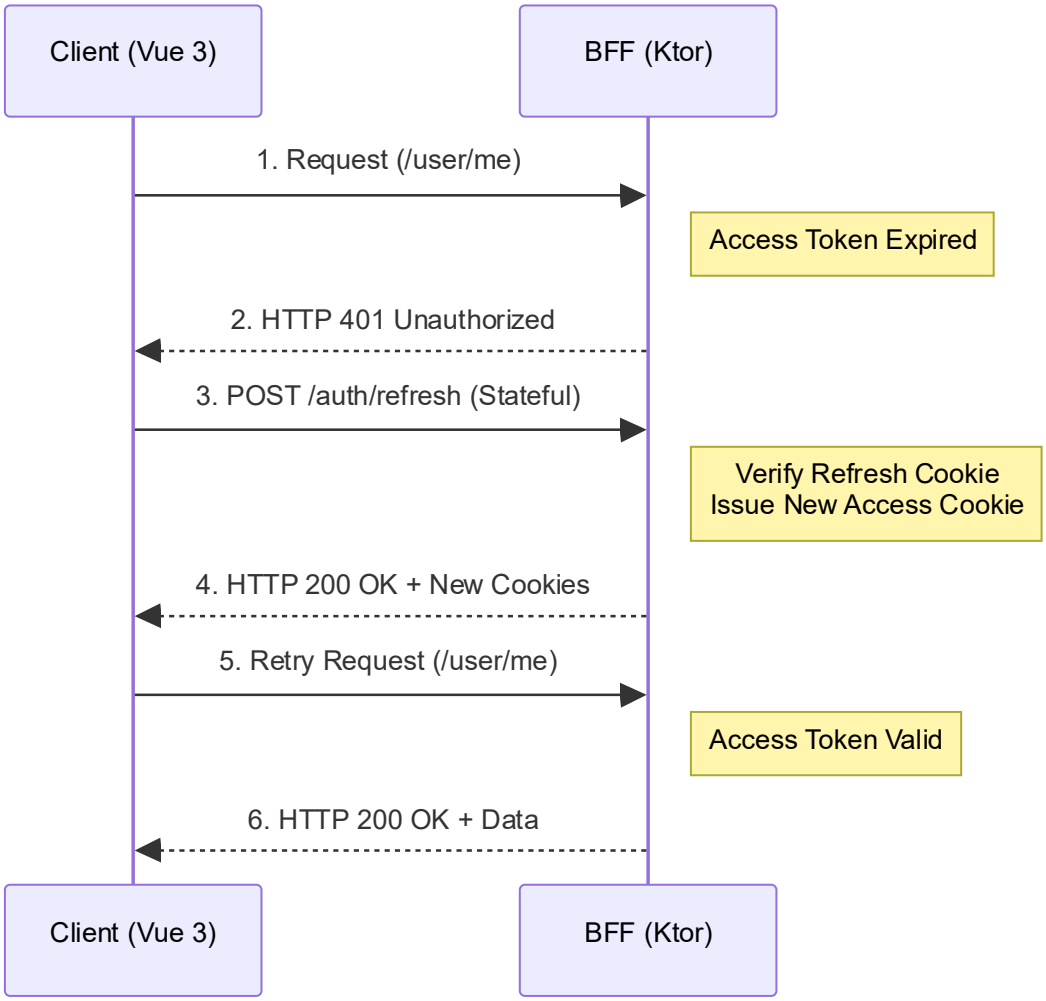
Για τη μεταγλώττιση της εφαρμογής και το πακετάρισμα της, χρησιμοποιήθηκε το vite. Σε αντίθεση με το webpack, το vite προσφέρει αρκετά αρκετά πλεονεκτήματα στην εμπειρία ανάπτυξης λογισμικού. Ένα από τα μεγαλύτερα πλεονεκτήματα είναι το native ESM dev server, για το οποίο γίνεται partial bundling του κώδικα χρησιμοποιώντας τα ES modules, προσφέροντας απίστευτα γρήγορες εκκινήσεις της εφαρμογής άσχετα από το μέγεθος της. Επιπλέον, για τα pre-bundle τα esbuild προσφέρουν εξαιρετικά γρήγορο πακετάρισμα κατά την μεταγλώττιση για εξωτερικές βιβλιοθήκες (μπορεί να είναι 10 έως 100 φορές πιο γρήγορο για βιβλιοθήκες που έχουν γραφτεί σε java script [11]). Τέλος, ένα σημαντικό πλεονέκτημα είναι το production bundling με rollup. Το rollup μπορεί να αφαιρέσει μη χρησιμοποιούμενο κώδικα χρησιμοποιώντας εξαιρετικά προηγμένες τεχνικές όπως το tree shaking (αφαιρεί μη χρησιμοποιούμενο κώδικα) και code splitting (τεμαχίζει τμήματα κώδικα για καλύτερη βελτιστοποίηση). Το τελικό αποτέλεσμα διασφαλίζει πως ο τελικός χρήστης θα κατεβάσει μόνο τα απολύτων απαραίτητα αρχεία κατά την πρώτη φόρτωση της σελίδας, δημιουργώντας μια καλύτερη εμπειρία.

4.1.3 CSS και design tokens με PrimeVue 4 και Tailwind

Για τη σχεδίαση της διεπαφής επιλέχθηκε το PrimeVue 4 με συνδυασμό του Tailwind. Ο συνδυασμός μπόρεσε να γίνει λόγο της νέας έκδοσης του PrimeVue, ο οποίος εισήγαγε νέα αρχιτεκτονική βασισμένη σε μεταβλητές CSS και design tokens, το οποίο επέτρεψε την άψογη ενσωμάτωση με τα utility first μοντέλα του Tailwind. Συγκεκριμένα, τα design tokens μετατρέπουν χρώματα, αποστάσεις και γενικότερα ιδιότητες της CSS σε tokens (για παράδειγμα --primary-color). Για αυτό το λόγο μπορεί να γίνει η δυναμική αλλαγή θεμάτων, όπως την εγγενή υποστήριξη του σκοτεινού θέματος σε επίπεδο συστήματος χωρίς διπλό κώδικα CSS. Ένα ακόμη πλεονέκτημα για την ανάπτυξη δυναμικών γραφικών είναι τα breakpoints (sm, md, lg) που προσφέρει το Tailwind. Με αυτόν τον τρόπο είναι πιο εύκολη η υλοποίηση γραφικών συστατικών για διαφορετικού μεγέθους οθόνες καθώς προσαρμόζονται αυτόματα, με μικρές προσαρμογές από τον προγραμματιστή.

4.2 Διαχείριση συνεδριών και BFF interceptors

Η χρήστη του μοτίβου BFF (Backend for frontend) επιβάλει σημαντικές αλλαγές στον τρόπο διαχείρισης των συνεδριών. Λογο της χρήσης του HttpOnly cookie, το frontend δεν έχει πρόσβαση στο περιεχόμενο του JWT token. Για αυτόν τον λόγο η εφαρμογή πρέπει να βασίζεται σε έμμεσους μηχανισμούς. Ένας μηχανισμός που μπορούμε να χρησιμοποιήσουμε είναι τα BFF inteceptors και στο διάγραμμα μπορούμε να δούμε πως χρησιμοποιούνται για να γίνει η διατήρηση της συνεδρίας.



Σχήμα 1: διατήρηση της συνεδρίας

4.2.1 Το μοτίβο BFF client interceptor

Όταν ένας χρήστης στέλνει ένα αίτημα προς το API, ο περιηγητής επισυνάπτει αυτόματα τα cookies της συνεδρίας (για αυτόν τον λόγο πρέπει να ρυθμίζονται σωστά τα CORS). Στην περίπτωση που το access token (το οποίο έχει μικρή διάρκεια ζωής των 10 λεπτών για λόγους ασφάλειας) έχει λήξει, τότε το backend θα επιστρέψει κωδικό σφάλματος "HTTP 401 Unauthorized".

Για να βελτιώσουμε την εμπειρία του χρήστη, έτσι ώστε να μην αποσυνδέετε κάθε 10 λεπτά, χρησιμοποιούμε τον μηχανισμό Silent token refresh (με τη χρήση ενός JWT refresh token) με τη συνάρτηση authenticatedFetch. Με αυτόν τον τρόπο έχουμε δημιουργήσει έναν interceptor δικτύου.

```
export async function authenticatedFetch(url, options = {}) {

    const defaultOptions = {
        credentials: 'include', // Επιβάλλουμε στον browser να στέλνει τα cookies
        ...options
    };

    let response = await fetch(url, defaultOptions);
```

```
    // Ανίχνευση λήξης του Access token
    if (response.status === 401) {
        try {
            // Αποστολή αιτήματος ανανέωσης στο BFF χρησιμοποιώντας το Refresh Token Cookie
            const refreshRes = await fetch(window.API_BASE_URL + '/auth/refresh', {
                method: 'POST',
                credentials: 'include'
            });

            if (refreshRes.ok) {
                // Εάν η ανανέωση πετύχει, επαναλαμβάνουμε την αρχική αίτηση
                response = await fetch(url, defaultOptions);
            } else {
                // Εάν το Refresh Token έχει επίσης λήξει, η συνεδρία τερματίζεται
                window.dispatchEvent(new CustomEvent('session-expired'));
            }
        } catch (e) {
            console.error('Refresh token failed', e);
            window.dispatchEvent(new CustomEvent('session-expired'));
        }
    }

    return response;
}
```

4.2.2 Αναλυτική επεξήγηση του κύκλου ζωή ενός αιτήματος

Όταν πρέπει να δημιουργηθεί ένα αίτημα στέλνεται στην authenticatedFect, η οποία δέχεται το URL μαζί με τις παραμέτρους του αιτήματος. Εξασφαλίζει να είναι έγκυρα τα πιστοποιητικά και να υπάρχει η ιδιότητα "credentials: 'include'" (Αυτή η ιδιότητα προσδιορίζει στον περιηγητή να συμπεριλάβει τα cookies SameSite/HttpOnly). Στη συνέχεια γίνεται η απόπειρα να σταλθεί στο αίτημα στο backend, όπου θα γίνει η επαλήθευση του Access Toekn cookie, στην περίπτωση που δεν υπάρξει κάποιο πρόβλημα θα σταλθεί το HTTP status code 200 και ο κύκλος ζωής του αιτήματος θα ολοκληρωθεί. Στην περίπτωση που έχουν λήξη τα πιστοποιητικά, θα επιστραφεί το HTTP status code 401 Unauthorized. Τότε θα γίνει μια απόπειρα για αθόρυβη ανανέωση, κατά την οποία θα αποσταλεί ένα νέο αίτημα post στο end point "/auth/refresh", το οποίο θα περιλαμβάνει αντί για το access token, το refresh token (το οποίο έχει ρυθμιστή σε διάρκεια 15 ημερών). Μόλις ολοκληρωθεί ο έλεγχος του refresh token και είναι έγκυρο, τότε θα εκδοθεί ένα νέο access token και θα απαντήσει με HTTP status code 200 και θα συμπεριλαμβάνει το access token. Όταν έχει ολοκληρωθεί η ενημέρωση του access token, τότε το frontend θα προσπαθήσει ξανά να κάνει το αίτημα που προσπαθούσε να κάνει. Τέλος, στην περίπτωση που έχει λήξη και το refresh token, τότε θα επιστραφεί το HTTP status code 403, το οποίο σημαίνει πως έχει λήξη το token και θα γίνει αναπροσδιορισμός (μέσω του event session-expired) του χρήστη στη σελίδα σύνδεσης.

4.2.3 Διαχείριση custom event και navigation guards

Το event session-expired είναι τμήμα της κεντρικής δομής της εφαρμογής TravelTales και πάντα εκτελεί τρις καθορισμένες ενέργειες. Πρώτα καθορίζει η τοπική κατάσταση του χρήστη, στη συνέχεια διαγράφονται τυχόν δεδομένα του χρήστη που υπάρχουν στην τοπική αποθήκευση του περιηγητή (όπως προτιμήσεις για το θέμα της εφαρμογής). Τέλος, ο vue router θα ανακατευθύνει τον χρήστη στη σελίδα σύνδεσης, προβάλλοντας και ένα pop-up μήνυμα πως η συνεδρία του έχει λήξη.

4.3 Δυναμική διαχείριση κατάστασης και ταυτοποίηση χρήστη

Για να διατηρηθεί η αυθεντικοποίηση του χρήστη σε όλα τα κομμάτια της Vue, δημιουργήθηκε το "useAuth.js". Χάρης το οποίο δεν είναι αναγκαία η χρήση εξωτερικών βιβλιοθηκών για τη διαχείριση κατάστασης (βιβλιοθήκες όπως Pinia ή το Vuex). Το useAuth υλοποιεί ένα ελαφρύ singleton αξιοποιώντας με αυτόν τον τρόπο τις εγγενείς δυνατότητες της Vue 3.

```
import { ref } from 'vue';

import { useLayout } from '@/layout/composables/layout';
import { AuthService } from '@/service/AuthService';

// Global Singleton State
const authState = ref({
    username: null,
    picture: null,
    role: null,
    firstName: null,
    lastName: null,
    bio: null,
    isLocalUser: false,
```

```
    loggedin: false
});

async function initAuth() {
    try {
        const me = await AuthService.getMe();
        authState.value.username = me.username || 'User';
        authState.value.picture = me.picture || null;
        authState.value.role = me.role || 'UNVERIFIED';
        authState.value.firstName = me.firstName || null;
        authState.value.lastName = me.lastName || null;
        authState.value.bio = me.bio || null;
        authState.value.isLocalUser = me.isLocalUser !== false;
        authState.value.loggedin = true;

        const { layoutConfig, applyTheme } = useLayout();
        const settings = await AuthService.getSettings();
        if (settings.themePreference !== undefined) {
            layoutConfig.themePreference = settings.themePreference;
            if (typeof window !== 'undefined') {
                localStorage.setItem('themePreference', settings.themePreference);
            }
            applyTheme();
        }
    } catch (e) {
        console.warn("User is not logged in or session expired.");
        logoutState();
    }
}

function processToken() {
    // Re-verify session identity dynamically since we have a verified new backend cookie
implicitly via REST
    initAuth();
}

async function logout() {
    try {
        await AuthService.logout();
    } catch (e) {
        console.error(e);
    }
    logoutState();
}

function logoutState() {
    authState.value.username = null;
    authState.value.picture = null;
    authState.value.role = null;
```

```
    authState.value.firstName = null;
    authState.value.lastName = null;
    authState.value.bio = null;
    authState.value.isLocalUser = false;
    authState.value.loggedin = false;
}

// Automatically check session cookie on import mount dynamically
initAuth();

export function useAuth() {
    return {
        authState,
        initAuth,
        processToken,
        logout
    };
}
```

4.3.1 Ο λόγος χρήσης του global singleton state

Δηλώνοντας τη μεταβλητή authState έξω από τον ορισμό της συνάρτησης useAuth, διασφαλίζουμε ότι όποτε χρησιμοποιείται η useAuth σε διαφορετικό τμήματα, όλα τα τμήματα θα μοιράζονται και θα παρατηρούν την ίδια ακριβώς δυναμική αναφορά. Δηλαδή, αν αλλάξει η κατάσταση σε ένα τμήμα, θα αλλάζει και για όλα τα υπόλοιπα.

4.3.2 Συγχρονισμός προτιμήσεων του γραφικού περιβάλλοντος

Μια λεπτομέρεια για την οποία δόθηκε προσοχή, είναι ο αυτόματος συγχρονισμός των ρυθμίσεων για το γραφικό περιβάλλον (όπως η προτίμηση ανοιχτού ή σκούρου θέματος). Η διαδικασία συγχρονισμού, γίνεται κατά τη διαδικασία σύνδεσης ταυτοποίησης του χρήστη.

```
async function initAuth() {
    try {
        // Ανάκτηση των στοιχείων του χρήστη από το BFF
        const me = await AuthService.getMe();
        authState.value.username = me.username || 'User';
        authState.value.picture = me.picture || null;
        authState.value.role = me.role || 'UNVERIFIED';
        authState.value.firstName = me.firstName || null;
        authState.value.lastName = me.lastName || null;
        authState.value.bio = me.bio || null;
        authState.value.isLocalUser = me.isLocalUser !== false;
        authState.value.loggedin = true;

        // Ανάκτηση και εφαρμογή των ρυθμίσεων εμφάνισης
        const { layoutConfig, applyTheme } = useLayout();
        const settings = await AuthService.getSettings();
        if (settings.themePreference !== undefined) {
            layoutConfig.themePreference = settings.themePreference;
            if (typeof window !== 'undefined') {
                localStorage.setItem('themePreference', settings.themePreference);
            }
            applyTheme(); // Δυναμική εφαρμογή του CSS Theme
        }
    } catch (e) {
        console.warn("User is not logged in or session expired.");
        logoutState();
    }
}
```

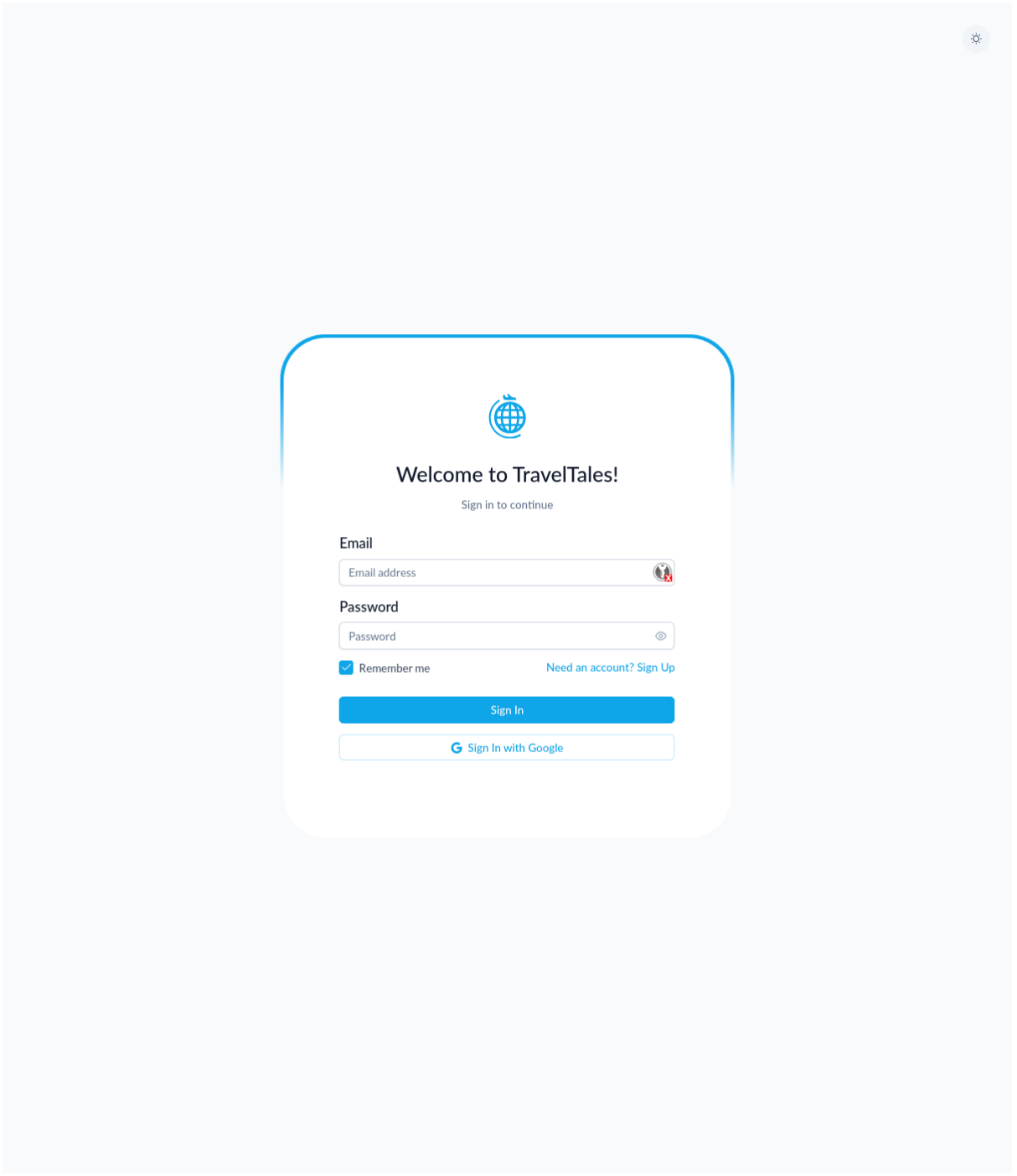
Η διαδικασία αυτή διασφαλίζει ότι μόλις ο χρήστης συνδεθεί, οι επιλογές του χρήστη θα εφαρμοστούν αμέσως χωρίς να υπάρχουν προβλήματα όπως να φορτώνει πρώτα το προεπιλεγμένο θέμα και μετά το θέμα του χρήστη (αυτό μπορεί να δημιουργήσει effects κατά τη φόρτωση). Επιπλέον, η υλοποίηση του θέματος σε backend είναι απαραίτητη για τη σωστή μεταφορά των επιλογών του χρήση και στην εφαρμογή του android.

4.4 Υλοποίηση της φόρμας σύνδεσης

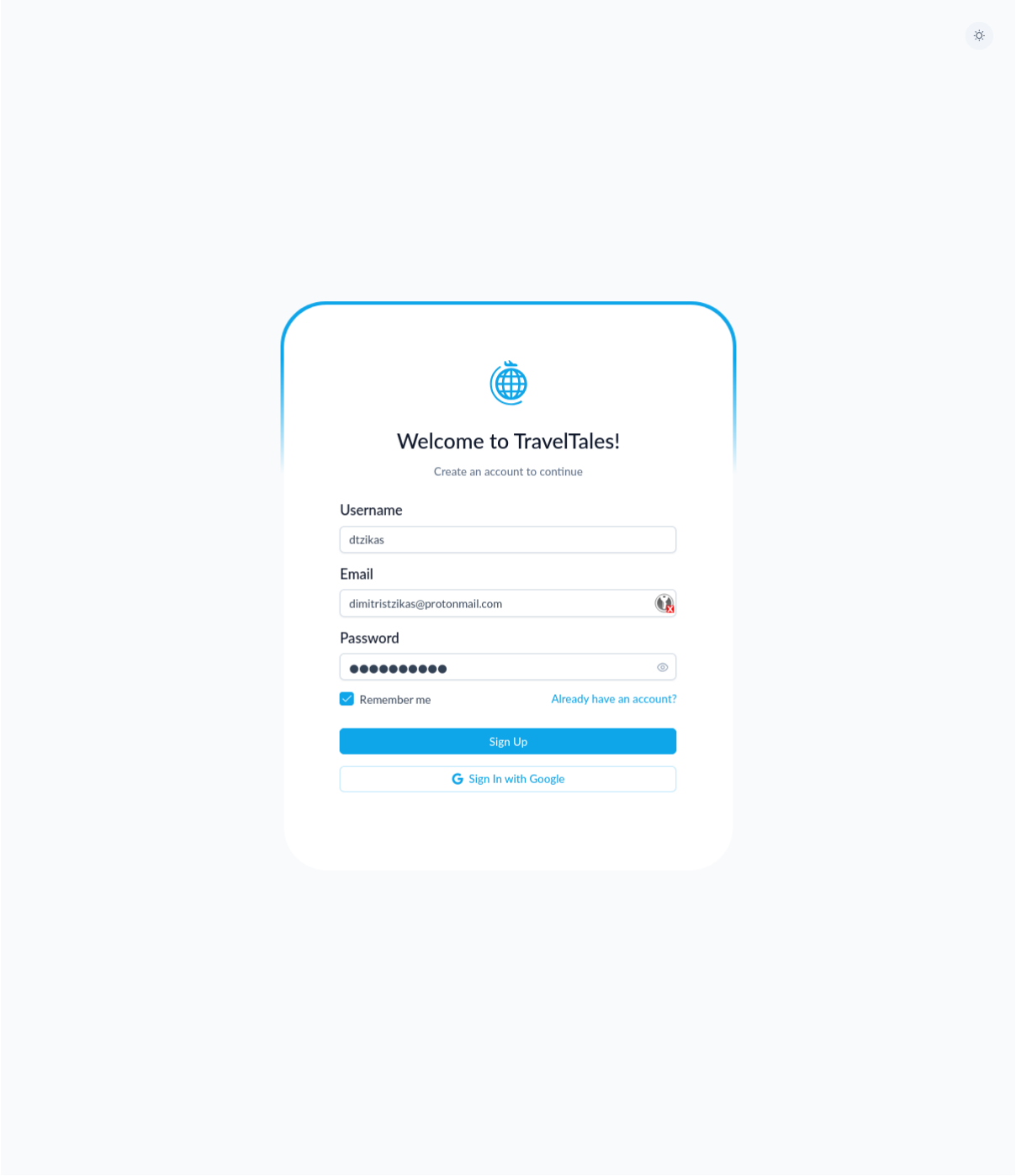
Η υλοποίησης της φόρμας σύνδεσης του χρήστη είναι ένα ιδιαίτερο τμήμα, καθώς ο προγραμματιστής πρέπει να είναι προσεκτικός με την υλοποίηση της. Αυτό συμβαίνει γιατί είναι ένα σημαντικό τμήμα για αλληλεπίδραση με το σύστημα ασφάλειας τους backend.

4.4.1 Η αρχιτεκτονική κομμάτι και πρότυπο (Component & Template)

Ένα κομμάτια που χρησιμοποιούνται από την PrimeVue εξασφαλίζουν την προσβασιμότητα και την αισθητική. Για παράδειγμα, χρησιμοποιείται το password κομμάτι, το οποίο προστατεύει τον κωδικό χρήστη από timing attacks στο γραφικό περιβάλλον αλλά προσφέρει και εγγενώς δυνατότητες απόκρυψης και εμφάνισης του κωδικού. Επιπλέον, έχουμε και την αλληλεπιδραστική αλλαγή κατάστασης, όπου υλοποιείται από τον προγραμματιστή με τη χρήση των "v-if" και "v-else".


Welcome to TravelTales!
Sign in to continue
Email
Email address
Password
Password
Remember me
Need an account? Sign Up
Sign In
Sign In with Google

Equation 2



Equation 3

4.4.2 Η ροή τοπικής σύνδεσης

Όταν ο χρήστης υποβάλει τη φόρμα, εκτελείται η μέθοδος submitAuth, η οποία προσφέρει έναν τελευταίο έλεγχο ορθότητας της φόρμας πριν αποσταλεί στο backend.

```
const submitAuth = async () => {
    errorMsg.value = '';
    if (!email.value || !password.value || (isSignup.value && !username.value)) {
        errorMsg.value = 'Please fill out all fields.';
        return;
```

```
    }

    loading.value = true;
    try {
        if (isSignup.value) {
            await AuthService.signup(email.value, password.value, username.value);
            await AuthService.login(email.value, password.value, checked.value);
            processToken();
            router.push('/');
        } else {
            const result = await AuthService.login(email.value, password.value,
checked.value);

            // Έλεγχος εάν ο χρήστης έχει ενεργοποιημένο το 2FA (WebAuthn)
            if (result && result.requires2FA) {
                handle2FAChallenge(result.authResponse); // Μετάβαση στη ροή WebAuthn
            } else if (result) {
                processToken();
                router.push('/');
            }
        }
    } catch (err) {
        errorMsg.value = err.message || "An error occurred.";
    } finally {
        loading.value = false;
    }
};
```

4.5 Υλοποίηση WebAuthn / Passkeys στον frontend

Η υλοποίηση και ενσωμάτωση του WebAuthn (FIDO2) στο γραφικό περιβάλλον αποτελεί μια εξαιρετικά απαιτητική απαιτητική κρυπτογραφική διαδικασία, καθώς χρειάζεται άμεση αλληλεπίδραση με το υλικό του λειτουργικού σ## 9.1 Σύνοψη και Αποτίμηση του Έργου.

4.5.1 Διαχείριση δυαδικών πινάκων

Το WebAuthn API του περιηγητή, χρησιμοποιεί αποκλειστικά προσωρινούς δυαδικούς πίνακες, μέσω των οποίων αναπαριστά κρυπτογραφικές προκλήσεις, υπογραφές και credentials. Αυτό δημιουργεί ασυμβατότητα με το πρωτόκολλο επικοινωνίας HTTP καθώς υποστηρίζει μόνο κείμενο. Για να επιλυθεί αυτό το πρόβλημα, δημιουργήθηκαν δύο συναρτήσεις, η bufferToBase64 και η base64ToBuffer, σύμφωνα με το πρότυπο RFC 4648 για Base64URL.

```
// Μετατροπή ArrayBuffer σε string Base64Url (αφαίρεση +, / και = padding)
const bufferToBase64 = (buffer) => {
    let binary = '';
    const bytes = new Uint8Array(buffer);
    const len = bytes.byteLength;
    for (let i = 0; i < len; i++) {
        binary += String.fromCharCode(bytes[i]);
```

```
    }
    return window.btoa(binary)
        .replace(/\+/g, '-')
        .replace(/\//g, '_')
        .replace(/=/g, '');
};

// Μετατροπή string Base64Url πίσω σε ArrayBuffer για το WebAuthn API
const base64ToBuffer = (base64) => {
    const binary = window.atob(base64.replace(/-/g, '+').replace(/_/g, '/'));
    const bytes = new Uint8Array(binary.length);
    for (let i = 0; i < binary.length; i++) {
        bytes[i] = binary.charCodeAt(i);
    }
    return bytes.buffer;
};
```

Όπως μπορεί να δεις κάποιος και στον κώδικα, οι συναρτήσεις μετατρέπουν τρις χαρακτήρες, τον +, _ και το =. Αυτοί οι χαρακτήρες χρησιμοποιούνται από το Base64 αλλά έχουν ειδική σημασία στα αιτήματα HTTP. Με αυτόν τον τρόπο μπορούμε να εγγυηθούμε την σωστή αλληλεπίδραση του API με το πρωτόκολλο HTTP.

4.5.2 Η διαδικασία ταυτοποίησης με το WebAuthn

Όταν το backend απαιτήσει τον έλεγχο 2FA, παρέχει μια κρυπτογραφική πρόκληση και τη λίστα των εγγεγραμμένων κλειδιών του χρήστη. Η συνάρτηση handle2FAChallenge, αναλαμβάνει τα ολοκληρώσει αυτή την διαδικασία με τη συνεργασία του περιηγητή.

```
const handle2FAChallenge = async (authResponse) => {
    try {
        errorMsg.value = 'Please insert and touch your hardware key.';

        // 1. Προετοιμασία των PublicKeyCredentialRequestOptions
        const getOptions = {
            publicKey: {
                // Μετατροπή του Base64Url challenge σε ArrayBuffer
                challenge: base64ToBuffer(authResponse.challenge),
                timeout: authResponse.timeout,
                rpId: authResponse.rpId, // Ταυτοποίηση του Domain (π.χ. localhost)
                allowCredentials: authResponse.allowCredentials.map(c => ({
                    type: 'public-key',
                    id: base64ToBuffer(c.id) // Μετατροπή των επιτρεπόμενων Credential IDs
                })),
                userVerification: 'preferred' // Ζητά βιομετρικά στοιχεία εάν είναι
διαθέσιμα
            }
        };

        // 2. Επίκληση του Hardware Authenticator
```

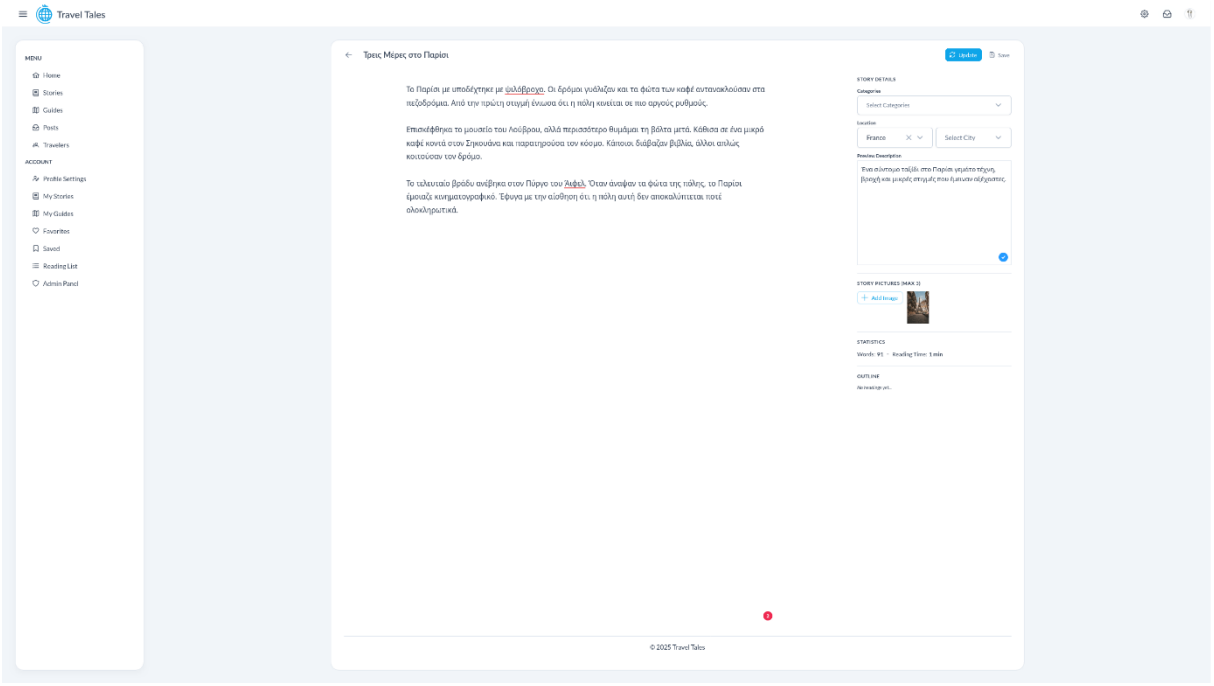
```
        const assertion = await navigator.credentials.get(getOptions);

        // 3. Κατασκευή των δεδομένων απάντησης (Assertion Response)
        const loginData = {
            email: email.value,
            id: assertion.id, // Base64Url αναγνωριστικό του κλειδιού
            rawId: bufferToBase64(assertion.rawId),
            type: assertion.type,
            response: {
                // Κρυπτογραφική υπογραφή και δεδομένα του authenticator
                authenticatorData: bufferToBase64(assertion.response.authenticatorData),
                clientDataJSON: bufferToBase64(assertion.response.clientDataJSON),
                signature: bufferToBase64(assertion.response.signature),
                userHandle: assertion.response.userHandle ?
bufferToBase64(assertion.response.userHandle) : null
            }
        };

        // 4. Αποστολή στο Backend για επαλήθευση της υπογραφής
        await AuthService.finish2FALogin(loginData, checked.value);
        processToken();
        router.push('/');
    } catch (err) {
        console.error(err);
        errorMsg.value = '2FA failed: ' + (err.message || 'Verification failed');
    }
};
```

4.5.3 Κρυπτογραφική ανάλυση

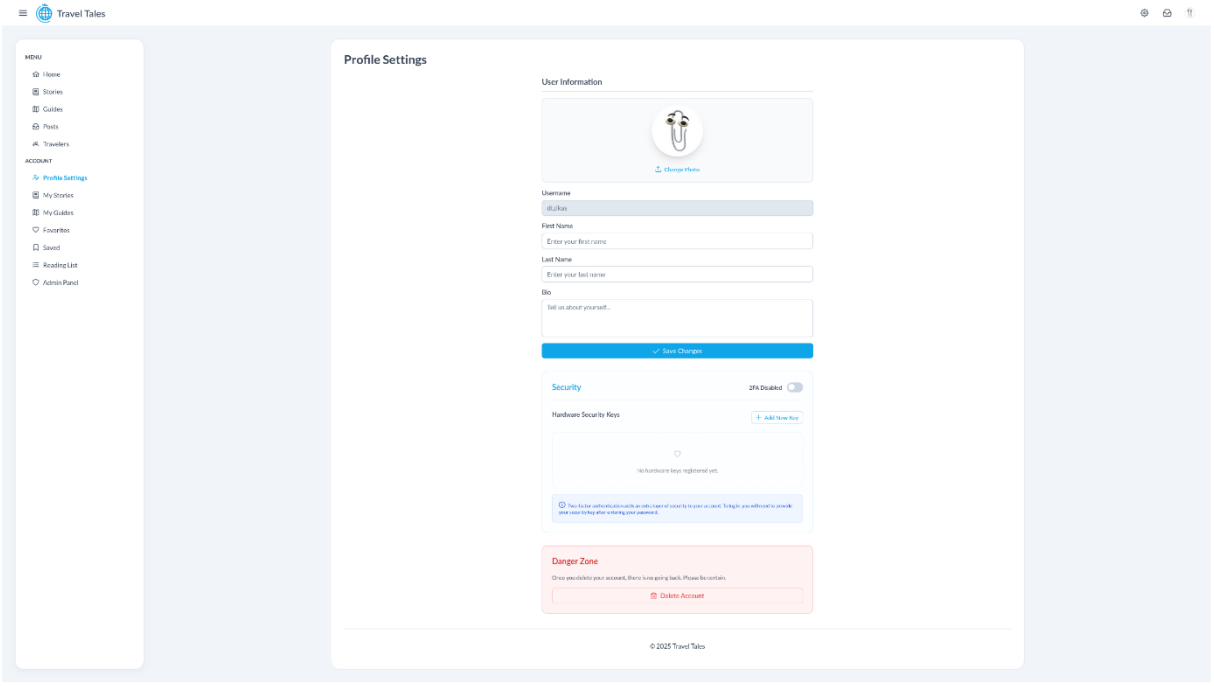
Το πρώτο πράγμα που πρέπει να γίνει είναι η διασφάλιση του domain name, για να αποτρέψουμε τυχόν επιθέσεις phishing. Αυτό γίνεται μέσω του περιηγητή κάνοντας επαλήθευση του domain (να σημειωθεί πως στην πτυχιακή δε λειτουργεί πλήρως καθώς δεν υπήρχε διαθέσιμο domain). Κατόπιν ο authenticator λαμβάνει την κρυπτογραφική πρόκληση, τη συνδυάζει με τα δεδομένα του πελάτη και τα υπογράφει χρησιμοποιώντας το ιδιωτικό κλειδί που υπάρχει στο φυσικό επίπεδο του κλειδιού USB. Τέλος, παρέχονται και πληροφορίες σχετικά με την κατάσταση που βρισκόταν ο authenticator και τη στιγμή της υπογραφής. Τέτοιες πληροφορίες μπορεί να είναι μια απαρίθμηση των υπογραφών ή αν ο χρήστης πιστοποίησε την υπογραφή.

4.6 Διαχείριση σφαλμάτων και UX βελτιστοποιήσεις

Όταν δημιουργείται ένα σύστημα αυθεντικοποίησης υψηλών προδιαγραφών, πρέπει να υπάρχει υπόψη πως η διαχείριση αποτυχιών θα πρέπει να γίνεται με κομψό τρόπος, καθώς δε θα πρέπει να εκθέτει ευαίσθητες πληροφορίες ασφάλειας αλλά και να μπορεί να καθοδηγήσει τον χρήστη.



Equation 4



Equation 5

4.6.1 Διαχείριση ακύρωσης και απόρριψη επιβεβαίωσης

Κατά την πιστοποίηση του WebAuthn, ο χρήστης μπορεί να ακυρώσει τη διαδικασία ή να μην επιβεβαιώσει την υπογραφή. Για αυτόν τον λόγο υπάρχουν τρις διαφορετικές κατηγορίες περιπτώσεων που διαχειρίζεται η μέθοδος handle2FAChallenge. Η πρώτη είναι αν το σφάλμα είναι τύπου NotAllowedError, το οποίο επιστρέφεται αν ο χρήστης ακυρώσει τη διαδικασία. Η δεύτερη είναι σε

περίπτωση παρέλθει ο χρόνος αναμονής και τέλος για ένα οποιοδήποτε άλλο σφάλμα που θα εμφανιστεί ένα γενικό μήνυμα λάθους.

4.6.2 Loading states και concurrent submissions

Για να αποφύγουμε διπλές υποβολές που θα μπορούσαν να δημιουργήσουν πολλαπλά προβλήματα, όπως πολλαπλά sessions ή database locks, χρησιμοποιείται η αλληλεπιδραστική μεταβλητή loading.

```
<Button :label="isSignup ? 'Sign Up' : 'Sign In'" :disabled="loading" class="w-full"
@click="submitAuth"></Button>
```

Με αυτόν τον τρόπο εξασφαλίζουμε πως το κουμπί δε θα μπορεί να χρησιμοποιηθεί αλλά θα ενημερωθεί και ο χρήστης για την κατάσταση του μιας οπτικής επιβεβαίωσης.

4.7 Συμπεράσματα

Η υλοποίηση τον frontend και το σύστημα αυθεντικοποίησης του TravelTales, αποδεικνύει πως η υψηλή ασφάλεια είναι αρκετά δύσκολη αλλά μπορεί να συνυπάρξει με μια καλή εμπειρία χρήστη. Αν και υπάρχουν πολλά θέματα ασφάλειας που δεν προσπαθεί να λύση, έχουν επιλυθεί αρκετά συχνά θέματα ασφάλειας. Ένα από τα σημαντικότερα είναι η ασφάλεια των JWT tokens, όπου η εφαρμογή δεν τα εκθέτει στο JavaScript runtime (χρησιμοποιώντας την απομόνωση του BFF), εξασφαλίζοντας την προστασία από επιθέσεις XSS (Cross-Site Scripting). Από την άλλη εξασφαλίζει μια καλύτερη εμπειρία χρήστη με την αόρατη διαχείριση tokens μέσω της μεθόδου authenticatedFetch. Τέλος, μπορούμε να πούμε πως με την άμεση επικοινωνία του Vue 3 και το native WebAuthn API του περιηγητή, καταφέραμε να υλοποιήσουμε μια state of the art εμπειρία σύνδεσης πολλαπλών

Κεφάλαιο 5ο: Υλοποίηση backend

Στην καρδιά του συστήματος της εφαρμογής TravelTales είναι το backend σύστημα, το οποίο αναλαμβάνει τις βασικές λειτουργίες της εφαρμογής. Όπως τη διαχείριση της επιχειρηματικής λογικής, τη διαχείριση της βάσης δεδομένων αλλά και την υλοποίηση και επιβολή των κανόνων ασφάλειας και πιστοποίησης. Στην BFF αρχιτεκτονική που επιλέχθηκε, το backend σύστημα δουλεύει ως ένας φύλακας, ο οποίος δεν εκτελεί απλώς stateless λειτουργίες, αλλά χειρίζεται και τα states για αιτήματα που έχουν ευαίσθητες πληροφορίες. Αξίζει να σημειωθεί πως ένας από τους σημαντικότερους λόγους που επιλέχθηκε η BFF, είναι επειδή η αρχική σχεδίαση του συστήματος θα γινόταν με τη χρήση microservices για καλύτερη υποστήριξη του scalability με τεχνολογίες όπως kubernetes, αλλά λόγο πολυπλοκότητας και χρονικού πλαισίου δεν υλοποιήθηκε.

Το οικοσύστημα που επιλέχθηκε για την ανάπτυξη του backend, είναι σύγχρονο, ασύγχρονο οικοσύστημα υψηλής απόδοσης. Το οποίο βασίζεται στη γλώσσα Kotlin (ένας λόγος που επιλέχθηκε συγκεκριμένα η Kotlin είναι η υποστήριξη της σε android συσκευές, έτσι ώστε να υπάρχει μικρότερη σύγχυση για τον full stack προγραμματιστή) και στο framework Ktor. Στο κεφάλαιο αυτό θα αναλύσουμε την αρχιτεκτονική των micro-modules του διακομιστή, την υλοποίηση του κρυπτογραφικού κατακερματισμού Argon2id, τον σχεδιασμό του JWT Authentication pipeline και την ενοποιήσει του τμημάτων μεταξύ τους.

5.1 Η τεχνολογική στοίβα του backend και η αρχιτεκτονική των micro modules

Η επιλογή των τεχνολογιών έγινε με γνώμονα την υλοποίηση ενός συστήματος το οποίο θα έχει τη δυνατότητα για ασύγχρονες υλοποίησης, non-blocking και ικανό να διαχειρίζεται χιλιάδες ταυτόχρονες συνδέσεις με ελάχιστη κατανάλωση πόρων.

Πίνακας 6

Layer	Components
Application Pipelines & Interceptors	Routing Engine; Session Handling; JWT Authentication Provider
Security Services	UserService (Argon2id Verification, JWT Access/Refresh Generation); WebAuthnManager (WebAuthn4J FIDO2 Validation Engine)
Data Access & ORM	JetBrains Exposed ORM (DSL Mode); Asynchronous Transaction Handler (suspendTransaction)
Database	PostgreSQL (UUID-OSSP, App Schema)

5.1.1 Kotlin για την ανάπτυξη λογισμικού

Η γλώσσα Kotlin επιλέχθηκε ως κύρια γλώσσα. Είναι μαι σύγχρονη γλώσσα προγραμματισμού με στατικό έλεγχο τύπων, η οποία εκτελείται μέσα στο περιβάλλον του Java Virtual Machine και προσφέρει πλήρη συμβατότητα με το τεράστιο οικοσύστημα της Java. Δύο από τα μεγαλύτερα χαρακτηριστικά της που χρησιμοποιήθηκαν είναι, τα coroutines και το null safety. Τα coroutines

επιτρέπουν στον προγραμματιστή να γράφει ασύγχρονο κώδικα με σειριακό τρόπο. Συγκεκριμένα, αντι ο προγραμματιστής να έχει σύνθετα callbacks ή reactive streams, μπορεί απλώς να γράφει με έναν σειριακό τρόπο. Τα coroutines εξασφαλίζουν πως τμήματα του κώδικα μπορεί να κάνει αναστολή της εκτέλεσης τους και να επανέρθει, με αυτόν τον τρόπο δε θα μπλοκάρει το thread του λειτουργικού συστήματος. Το null safety, επιτρέπει τον αυστηρό διαχωρισμό τον null με μη null τύπων δεδομένων σε επίπεδο μεταγλωττιστή. Με αυτόν τον τρόπο μπορούν να υπάρχουν λιγότερα απρόβλεπτα σφάλματα με NullPointerException.

5.1.2 Το Ktor framework

Το Ktor framework, σε αντίθεση με το παραδοσιακό και βαρύ Spring Boot, είναι ένα ελαφρύ και ασύγχρονο framework σχεδιασμένο από τη JetBrains ειδικά για την Kotlin. Χρησιμοποιεί μηχανές όπως Netty ή Jetty και βασίζεται σε ένα μοντέλο pipeline. Πρακτικά αυτό σημαίνει πως κάθε HTTP αίτημα περνάει από κάποιες φάσεις όπου μπορούν να προστεθούν με τη μορφή plugins (Authentication, Content Negotiation, Sessions, CORS), επιτρέποντας την κατασκευή modular κώδικα.

5.1.3 JetBrains Exposed ORM και POstgreSQL

Για την εύκολη και επεκτάσιμη επικοινωνία με τη βάση δεδομένων PostgreSQL, χρησιμοποιήθηκε το JetBrains Exposed, το οποίο είναι ένα ελαφρύ ORM γραμμένο σε Kotlin. Το Exposed προσφέρει δύο σημαντικά αφαιρετικά επίπεδα. Το πρώτο είναι το DAO (Database Access Objects), το οποίο επιτρέπει την εύκολη συγγραφή SQL ερωτημάτων μέσα από κώδικα Kotlin. Μέσα από αυτό εξασφαλίζεται ότι οποιοδήποτε λάθος σε όνομα στήλης ή τύπου δεδομένων θα εντοπιστεί κατά τη μεταγλώττιση.

5.2 Κρυπτογραφικός κατακερματισμός Argon2id

Ένα συχνό λάθος που συμβαίνει μέχρι και στις σημερινές εφαρμογές, είναι η αποθήκευση κωδικών σε κρυπτογραφημένη μορφή με τη χρήση μην ασφαλών αλγορίθμων (MD5, SHA-1, SHA-256 χωρίς salt) ή ακόμη και σε απλό κείμενο. Για αυτό για την εφαρμογή TravelTales επιλέχθηκε το Argon2id.

5.2.1 Γιατί επιλέχθηκε το Argon2id

Το Argon2id είναι ένας υβριδικός αλγόριθμος κρυπτογράφησης όπου συνδυάζει χαρακτηριστικά του Argon2i και του Argon2d. Αξίζει να σημειωθεί πως έχει ανακηρυχθεί ως ο νικητής του Password Hashing Competition (PHC) τον Ιούλιο του 2015 [12] και προτείνεται επίσης από το IETF (RFC 9106) [13] και το OWASP. Ένα λόγος που του δόθηκε προβάδισμα είναι η ανάγκη μεγάλης μνήμης για τον υπολογισμό ενός hash. Αυτό καθιστά επιθέσεις brute force με χρήση εξειδικευμένου εξοπλισμού εξαιρετικά αργές και ακριβές για τον επιτιθέμενο. Ακόμη ένα μεγάλο πλεονέκτημα του είναι το side-channel timing resistance. Αυτό σημαίνει πως ο επιτιθέμενος δεν μπορεί να κάνει ανάλυση χρόνου για να σπάσει την κρυπτογράφηση.

5.2.2 Η υλοποίηση του Argon2id στο backend

Η επαλήθευση των κωδικών υλοποιείται στην κλάση UserService.kt και χρησιμοποιεί τη βιβλιοθήκη de.mkammerer.argon2.

```
suspend fun authenticateByEmail(request: LoginRequest): User? {
val databaseUser = userRepository.getUserByEmail(request.email) ?: return null
val password = databaseUser.password ?: return null
val isValid =
```

```
Argon2Factory.create(Argon2Factory.Argon2Types.ARGON2id)
.verify(password, request.password.toCharArray())
return if (isValid) databaseUser else null
}
```

Η μέθοδος verify ανακτά παραμέτρους όπως το salt, iterations, memory και parallelism που βρίσκονται στο hash string της βάσης δεδομένων, κατόπιν εκτελεί τον κρυπτογραφικό υπολογισμό και επιστρέφει "αληθές" μόνο όταν το αποτέλεσμα που βγάλει είναι ίδιο με το αποτέλεσμα που βρίσκεται στη βάση δεδομένων.

5.3 Η υλοποίηση του JWT authentication pipeline

Όταν γίνει η επιτυχή ταυτοποίηση του χρήστη, πρέπει με κάποιον τρόπο να γνωστοποιεί πως έχει τακτοποιηθεί επιτυχώς. Για τον σκοπό αυτό χρησιμοποιήσαμε το JWT.

Πίνακας 7

JWT ACCESS TOKEN (HttpOnly Cookie)

HEADER	Algorithm (HS256), Type (JWT)
PAYLOAD	id, username, role, expiresAt (10m)
SIGNATURE	HMAC-SHA256(Header + Payload, SECRET)

5.3.1 Η στρατηγική του διπλού token (Access και refresh)

Για να εξισορροπηθεί η ασφάλεια με την εμπειρία χρήστη, υλοποιήθηκε μια στρατηγική δύο token. Το ένα είναι το token πρόσβασης, το οποίο έχει διάρκεια ζωής 10 λεπτών και περιέχει βασικές πληροφορίες του χρήστη και χρησιμοποιείται για την έγκριση αιτημάτων που απαιτούν ταυτοποίηση. Το δεύτερο είναι το token ανανέωσης, το οποίο έχει διάρκεια ζωής 15 ημερών και χρησιμοποιείται για τη δημιουργία νέου token πρόσβασης.

```
fun generateAccessToken(user: User): String {
```

```
return JWT.create()
.withAudience(jwtData.audience)
.withIssuer(jwtData.issuer)
.withSubject(user.username)
.withClaim("username", user.username)
.withClaim("id", user.id)
.withClaim("role", user.role.name)
.withExpiresAt(Date(System.currentTimeMillis() + 600000)) // 10 λεπτά
.sign(Algorithm.HMAC256(jwtData.secret))
}
```

5.3.2 Η δυναμική μετάφραση cookies σε header (BFF bridge)

Επειδή τα tokens τα αποθηκεύουμε σε HttpOnly cookies για την προστασία τους από XSS, ο jwt verifier του ktor δεν μπορεί να τα διαβάσει επειδή περιμένει να τα διαβάσει στις κεφαλίδες. Για να λύσουμε αυτό το πρόβλημα δημιουργήσαμε το authentication στο αρχείο JWTAuthentication.kt.

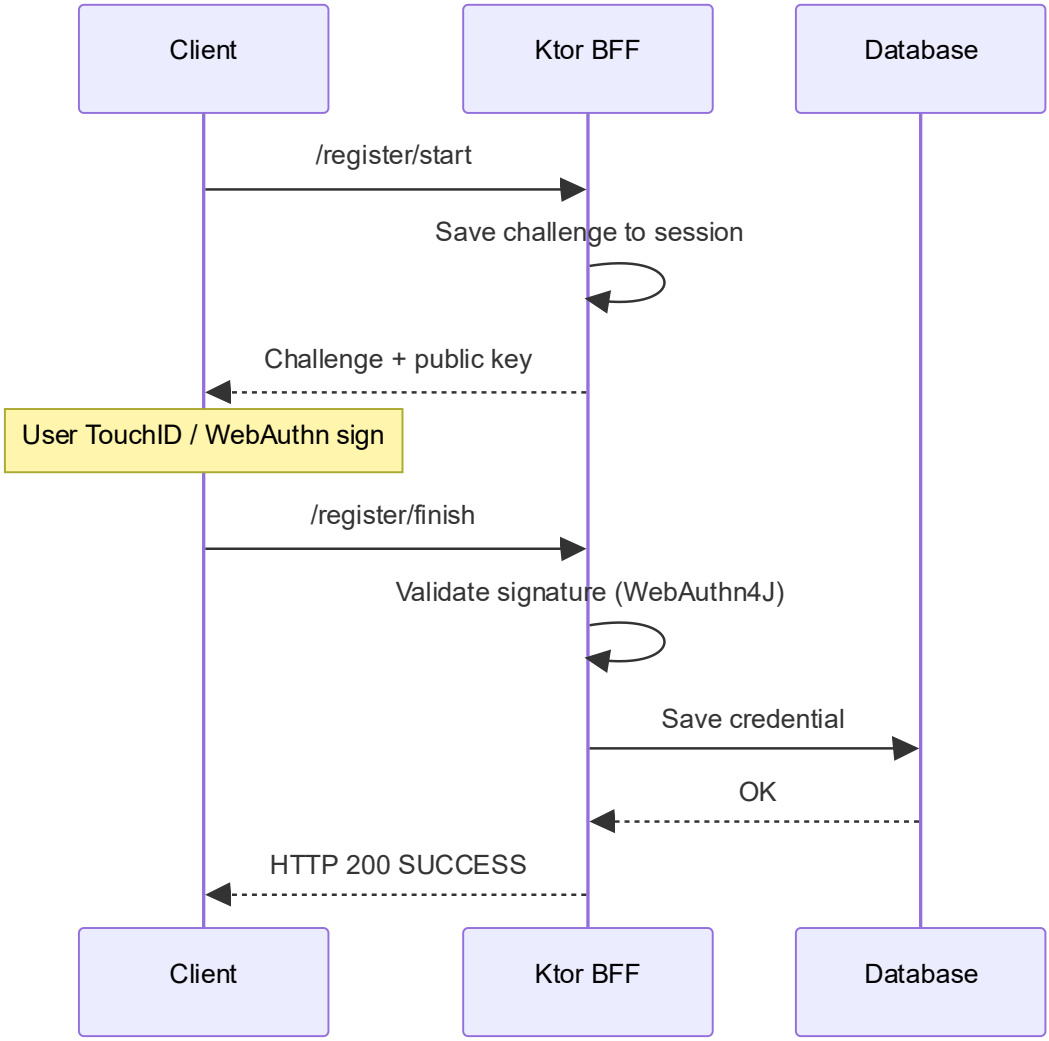
```
fun Application.configureJWTAuthentication(config: ApplicationConfig) {
    authentication {
        jwt("jwt-auth") {
            realm = config.property("jwt.realm").getString()
            val jwtSecret = config.property("jwt.secret").getString()
// Custom αναχαίτιση του Header για εξαγωγή από το Cookie
            authHeader { call ->
                val cookieValue = call.request.cookies["auth_token"]
                if (cookieValue != null) {
                    try {
// Μετατροπή του cookie σε εσωτερικό Bearer Header
                        io.ktor.http.auth.HttpAuthHeader
                            .Single("Bearer", cookieValue)
                    } catch (e: Exception) {
                        null
                    }
                } else {
// Fallback στον παραδοσιακό τρόπο (Authorization Header)
                    call.request.parseAuthorizationHeader()
                }
            }
            verifier {
                JWT
                    .require(Algorithm.HMAC256(jwtSecret))
                    .withAudience(config.property("jwt.audience").getString())
                    .withIssuer(config.property("jwt.issuer").getString())
                    .build()
            }
            validate { credential ->
                if
(credential.payload.audience.contains(config.property("jwt.audience").getString())) {
                    JWTPrincipal(credential.payload)
                } else {
                    null
                }
            }
        }
    }
}
```

Με αυτόν τον τρόπο μπορούμε να συνδέσουμε τα HttpOnly cookies με τα στάνταρ πρωτόκολλα χωρίς να αλλοιώνει τη ροή του Ktor pipeline.

5.4 Υλοποίηση της αυθεντικοποίησης WebAuthn

Το πιο κρίσιμο και κρυπτογραφικά θέμα είναι η ενοποίηση του backend με το πρότυπο WebAuthn (FIDO2) μέσω της βιβλιοθήκης WebAuthn4J της FIDO.

Σχήμα 2: WebAuthn lifecycle



5.4.1 Η διαδικασία εγγραφής

Η εγγραφή μιας φυσικής συσκευής πιστοποίησης αποτελείται από δύο φάσεις. Η πρώτη φάση η έναρξη της διαδικασίας. Στο backend αυτό υλοποιείται με το end point /2fa/register/start, το οποίο δημιουργεί μια κρυπτογραφική πρόκληση και την αποθηκεύει στην συνεδρία του χρήστη για να αποτρέψει relay attacks.

```
post("/register/start") {
    val principal = call.principal<JWTPrincipal>()
    val userId = UUID.fromString(principal?.payload?.getClaim("id")?.asString())
    val challenge = UUID.randomUUID().toString()
    val session = call.sessions.get<UserSession>() ?: UserSession("", "")

// Αποθήκευση του challenge στη συνεδρία
```

```
    call.sessions.set(session.copy(webAuthnChallenge = challenge))

    val user = queryWebAuthnUser(userId) // Ανάκτηση στοιχείων χρήστη
    call.respond(
        WebAuthnRegistrationStartResponse(
            challenge =
Base64.getUrlEncoder().withoutPadding().encodeToString(challenge.toByteArray()),
            user = user,
            rp = WebAuthnRp(rpId, rpName),
            pubKeyCredParams = listOf(PubKeyCredParam("public-key", -7)), // ES256 (ECDSA
over NIST P-256)
        ),
    )
}
```

Η παράμετρος ES256 καθορίζει τον αλγόριθμο υπογραφής από τη συσκευή του χρήστη να είναι ο ECDSA με SHA-256. Ο συγκεκριμένος αλγόριθμος είναι το πιο χρησιμοποιούμενο πρότυπο για κρυπτογραφικές συσκευές (passkeys).

Στη δεύτερη φάση ολοκληρώνεται η διαδικασία εγγραφής της συσκευής. Όταν το frontend ολοκληρώσει την επαλήθευση, στείλει ένα attestation object μέσω του end point /2fa/register/finish.

```
post("/register/finish") {
    val principal = call.principal<JWTPrincipal>()
    val userId = UUID.fromString(principal?.payload?.getClaim("id")?.asString())
    val request = call.receive<WebAuthnRegistrationFinishRequest>()
    val session = call.sessions.get<UserSession>()
    val savedChallenge =
        session?.webAuthnChallenge ?: return@post call.respond(HttpStatusCode.BadRequest,
"No challenge found")
    try {
        val webAuthnManager = WebAuthnManager.createNonStrictWebAuthnManager()
        val origin = Origin(call.request.headers["Origin"] ?: "http://localhost:5173")
        val registrationRequest =
            RegistrationRequest(
                Base64.getUrlDecoder().decode(request.response.attestationObject),
                Base64.getUrlDecoder().decode(request.response.clientDataJSON),
            )
        val registrationParameters =
            RegistrationParameters(
                ServerProperty(origin, rpId,
DefaultChallenge(savedChallenge.toByteArray()), null),
                null,
                false,
            )
// Εκτέλεση κρυπτογραφικής επαλήθευσης της υπογραφής
        val result = webAuthnManager.validate(registrationRequest, registrationParameters)
        suspendTransaction {
            val objectConverter = ObjectConverter()
```

```
            WebAuthnCredentialsTable.insert {
                it[WebAuthnCredentialsTable.userId] = userId
                it[credentialId] =
                    Base64.getEncoder().encodeToString(
                        result.attestationObject!!
                            .authenticatorData.attestedCredentialData!!
                            .credentialId,
                    )
// Αποθήκευση του Δημόσιου Κλειδιού σε μορφή CBOR (Concise Binary Object Representation)
                it[publicKey] =
                    Base64.getEncoder().encodeToString(
                        objectConverter.cborConverter.writeValueAsBytes(
                            result.attestationObject!!
                                .authenticatorData.attestedCredentialData!!
                                .coseKey,
                        ),
                    )
                it[signCount] = result.attestationObject!!.authenticatorData.signCount
                it[name] = request.name
            }
            UserTable.update({ UserTable.id eq userId }) {
                it[isTwoFactorEnabled] = true
            }
        }
        call.respond(HttpStatusCode.OK, mapOf("status" to "success"))
    } catch (e: Exception) {
        call.respond(HttpStatusCode.BadRequest, e.message ?: "Verification failed")
    }
}
```

5.5 Ασφάλεια συνεδριών, νηματοποίηση και ασύγχρονη εκτέλεση

Δύο μεγάλα ζητήματα σε κάθε backend σύστημα υψηλών αποδόσεων, είναι η διαχείριση των νημάτων και η αφαίρεση blocking λειτουργιών από το κύριο event loop του διακομιστή.

5.5.1 Συναλλαγές χωρίς blocking

Οι περισσότερες βιβλιοθήκες ORM (ισχύει για τη JetBrains Exposed) εκτελούν τα SQL ερωτήματα με σύγχρονο τρόπο. Αν αφήσουμε τα ερωτήματα να λειτουργούν με αυτόν τον τρόπο μ΄μέσα στο Ktor routing, θα έχουμε ως αποτέλεσμα να μπλοκάρουμε το κύριο thread του Netty, ως αποτέλεσμα να μειώνονται οι επιδώσεις του συστήματος. Για να λύσουμε αυτό το πρόβλημα υλοποιήθηκε ο βοηθητικός μηχανισμός suspendTransaction.

```
import kotlinx.coroutines.Dispatchers
import org.jetbrains.exposed.sql.transactions.experimental.newSuspendedTransaction

suspend fun <T> suspendTransaction(block: suspend () -> T): T =
newSuspendedTransaction(Dispatchers.IO) {
block()
```

```
}
```

Χρησιμοποιούμε τον dispatcher των Kotlin Coroutines (Dispatchers.IO), ο οποίος είναι ειδικά σχεδιασμένος και βελτιστοποιημένος για να διαχειρίζεται εργασίες I/O. Χρησιμοποιεί ένα δυναμικά επεκτεινόμενο pool από νήματα. Στην πράξη, όταν εκτελέσουμε ένα SQL ερώτημα, αντί να μπλοκάρει το κύριο νήμα γιατί περιμένει την απάντηση από τη βάση δεδομένων, σταματά τη λειτουργία του για να εξυπηρετηθούν άλλα αιτήματα HTTP και το SQL ερώτημα εκτελείται σε ένα thread του Dispatchers.IO. Όταν λάβει απάντηση από τη βάση δεδομένων, το coroutine θα επανέλθει στο κύριο νήμα και θα συνεχίσει η εκτέλεση του. Με αυτόν τον τρόπο έχουμε λιγότερο χρόνο αναμονής στο κύριο νήμα, αυξάνοντας τις επιδώσεις του συστήματος.

5.5.2 Stateful Ktor συνεδρίες για αποτροπή επιθέσεων replay

Το WebAuthn απαιτεί stateful διαχείριση των κρυπτογραφικών προκλήσεων. Αν δεν υπήρχε state στο backend, τουλάχιστον για τα αιτήματα που χρειάζονται πιστοποιήσεις, θα έπρεπε να εμπιστεύεται την απάντηση που στέλνει ο πελάτης για την κρυπτογραφική πρόκληση, δημιουργώντας ευκαιρίες για replay επιθέσεις. Μια δεύτερη λύση θα ήταν να υπάρχει μια προσωρινή μνήμη στη βάση (ή σε κάποιο άλλο σύστημα) αυξάνοντας τα I/O.

Για το backend σύστημα του TravelTales επιλέχθηκε να λυθεί αυτό το ζήτημα με τη χρήση του plugin συνεδρίας (Ktor Session Plugin)

```
install(Sessions) {
    cookie<UserSession>("USER_SESSION") {
        cookie.path = "/"
        cookie.httpOnly = true
        cookie.secure = true // Μόνο μέσω HTTPS σε production
    }
}
```

Με τη χρήση αυτού το plugin, η κρυπτογραφική πρόκληση αποθηκεύεται στη συνεδρία του χρήστη (η οποία είναι κρυπτογραφικά προστατευμένη). Όταν ο χρήστης καλέσει ένα end point το οποίο απαιτεί πιστοποίηση, τότε το backend θα συγκρίνει την κρυπτογραφική πρόκληση με αυτή που είναι αποθηκευμένη στο cookie της συνεδρίας. Όταν ολοκληρωθεί επιτυχώς διαγράφεται αμέσως, με αυτόν τον τρόπο δε γίνεται να χρησιμοποιηθεί ξανά η ίδια πρόκληση.

5.6 Συμπεράσματα

Η υλοποίηση του backend συστήματος της εφαρμογής του TravelTales, δείχνει πως χρησιμοποιώντας κάποιος σύγχρονες τεχνολογίες μπορεί να προσφέρει καλή ασφάλεια και απόδοση σε ένα σύστημα, αλλά και πως όσο περισσότερες επιθέσεις προσπαθεί να αντιμετωπίσει ένα σύστημα, τόσο πιο πολύπλοκο γίνεται.

Οι στόχοι που πετύχαμε με το backend της εφαρμογή είναι η ασφάλεια των κωδικών. Όπου χρησιμοποιήθηκε το Argon2id, το οποίο μέχρι και σε περίπτωση πλήρους διαρροής της βάσης δεδομένων, οι κωδικοί ασφάλειας των χρηστών θα παραμείνουν ασφαλείς από επιθέσεις brute force. Η χρήση των JWT πρόσβασης και ανανέωσης με HttpOnly/SameSite cookies με συνδυασμό την αυτόματη μετάφραση του Bearer header, εξασφαλίζει πως δεν μπορούν να γίνουν XSS επιθέσεις καθώς μικραίνει

και την επιφάνεια επίθεσης μαζί με την επιφάνεια ζημιάς σε περίπτωση που κλαπεί το token πρόσβασης. Επιπλέον, η χρήση του WebAuthn με την αποθήκευση κλειδιών σε μορφή CBOR/COSEKey, προστατεύει τους χρήστες από επιθέσεις phishing και κλωνοποίησης συσκευών, το οποίο είναι επιπέδου τραπεζικής ασφάλειας. Τέλος, με τη χρήση των coroutines της Kotlin καταφέραμε να διασφαλίσουμε μια μην blocking λειτουργία, με αποτέλεσμα την αποδοτικότερη και γρηγορότερη λειτουργία του διακομιστή.

Κεφάλαιο 6ο: Υλοποίηση εφαρμογής android

Στο πλαίσιο της πτυχιακής υλοποιήθηκε και μια εφαρμογή που αποτελεί την επέκταση της πλατφόρμας στο οικοσύστημα των κινητών συσκευών android. Στόχος της εφαρμογής είναι να υλοποιεί πλήρως τις λειτουργίες τις ιστοσελίδας χρησιμοποιώντας τις εγγενείς δυνατότητες της κινητής συσκευής.

Η ανάπτυξη της εφαρμογής σε android δεν είναι μια απλή εφαρμογή, αλλά χρησιμοποιήθηκε το Kotlin multiplatform (KMP) και το Compose multiplatform. Με αυτόν τον τρόπο η επέκταση της εφαρμογής για υποστήριξη σε συσκευές apple είναι πιο εύκολη καθώς μπορεί να χρησιμοποιηθεί από κοινού η επιχειρηματική λογική, η επικοινωνία δικτύου αλλά και οι τοπικές βάσεις δεδομένων.

Σε αυτό το κεφάλαιο αναλύεται η αρχιτεκτονική της android εφαρμογής, ο σχεδιασμός της τοπικής βάσης δεδομένων room database KMP, ο μηχανισμός για την ασφαλή αποθήκευση δεδομένων αλλά και η σχεδίασης της δυναμικής διεπαφής με Material 3.

6.1 Η τεχνολογική στοίβα και η αρχιτεκτονική του Kotlin multiplatform (KMP)

Το KMP (Kotlin multiplatform) παρέχει τη δυνατότητα, σαν πολλά άλλα cross platform frameworks, της ανάπτυξης μιας εφαρμογής για πολλαπλές συσκευές. Η διαφορά όμως του KMP είναι πως αντί να προσθέτει overhead στην υλοποίηση κάνοντας την εφαρμογή χειρότερη σε θέματα εκτέλεσης, μεταγλωττίζει τον κοινό κώδικα απευθείας σε byte code για το JVM για τις ανάγκες του android και σε native binaries με τη χρήση του LLVM για τις ανάγκες του ios.

Πίνακας 8

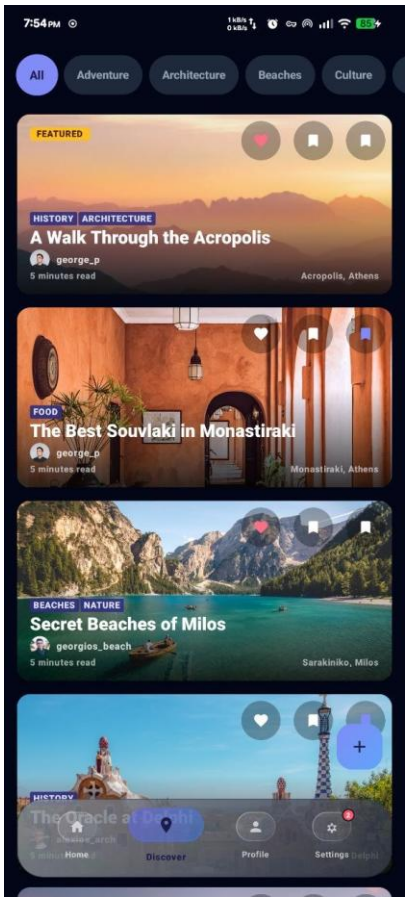
COMPOSE MULTIPLATFORM UI (commonMain)	
Shared Material 3 UI Components, AppTheme, WindowsSizeClassProvider	
SHARED LOGIC LAYER (commonMain)	
ViewModels (MVI State Flows), Use Cases, Repositories, Domain Models	
Ktor Client, Room Database Schema, Koin DI Core	
ANDROID PLATFORM LAYER (androidMain)	**IOS PLATFORM LAYER (iosMain)**
getDatabaseBuilder (Android Ctx)	getDatabaseBuilder (File Path)
SecureSettings (AES256 Prefs)	SecureSettings (iOS Keychain)
AndroidLocationTracker (GPS)	IosLocationTracker (CoreLocation)
MapView (Google Maps SDK)	MapView (MapKit SDK)

6.1.1 Η δομή της εφαρμογής και των κοινών τμημάτων

Η εφαρμογή χωρίζεται σε δύο κύριες κατηγορίες, τη composeApp και την android. Το composeApp αποτελεί τον κοινό κώδικα με τα επιμέρους τμήμα του (όπως commonMain, androidMain, iosMain/wasmJsMain/jvmMain). Το android, το οποίο είναι ένας ελαφρύς wrapper που χρησιμοποιείται για την ανάπτυξη του τελικού APK/AAB αρχείου για τις συσκευές android.

6.1.2 Ο μηχανισμός except/actual

Το KMP για να μπορέσει να έχει καθαρό κώδικα χωρίς τη χρήση πολύπλοκων runtime guards ή reflections, χρησιμοποιεί το συντακτικό πρότυπο expect/actual. Το expect χρησιμοποιείται για να δηλώσει την υπογραφή μιας συνάρτησης ή μιας κλάσης όπου χρησιμοποιείται σαν "συμβόλαιο". Από την άλλη το actual, πρέπει να υλοποιηθεί υποχρεωτικά σε κάθε πλατφόρμα που υποστηρίζεται χρησιμοποιώντας τα εκάστοτε εγγενή APIs της.



Equation 6

6.2 Τοπική αποθήκευση

Για να διασφαλίσουμε τη σωστή λειτουργία της εφαρμογής σε συνθήκες περιορισμένης ή ανύπαρκτης σύνδεσης στο διαδίκτυο, υλοποιήθηκε η εφαρμογή με έναν μηχανισμό προσωρινής αποθήκευσης δεδομένων. Για αυτόν τον σκοπό χρησιμοποιήθηκε η βιβλιοθήκη Room database, η οποία παρέχει ένα επίπεδο αφαίρεσης για τη SQLite. Με αυτόν τον τρόπο υπάρχει type safe πρόσβαση στη βάση με αυτόματη δημιουργία SQL εγγράφων σε Kotlin data objects.

6.2.1 Ο ορισμός του schema και των entries

Για την αναπαράσταση των πινάκων γίνεται μέσω κλάσεων της Kotlin με κατάλληλα annotations της Room. Για παράδειγμα, η δήλωση της οντότητας StoryEntity:

```
package com.traveltales.data.local.db

import androidx.room3.Entity
```

```kotlin
import androidx.room3.PrimaryKey

// Ορισμός του πίνακα τοπικής αποθήκευσης ιστοριών
@Entity(tableName = "stories")
data class StoryEntity(
    @PrimaryKey val id: String, // Μοναδικό UUID ιστορίας
    val title: String,
    val content: String,
    val authorId: String,
    val authorUsername: String,
    val authorPicture: String?,
    val locationName: String?,
    val latitude: Double?,
    val longitude: Double?,
    val isLiked: Boolean,
    val isSaved: Boolean,
    val likesCount: Int,
    val commentsCount: Int,
    val createdAt: Long,
    val updatedAt: Long
)
```

6.2.2 Η ρυθμίσεις expect/actual του database builder

Η Room χρειάζεται πρόσβαση στο τοπικό σύστημα της κινητής συσκευής για να μπορέσει να δημιουργήσει τη βάση δεδομένων. Ανάλογα με το σύστημα που τρέχει η εφαρμογή μπορεί να διαφέρει ριζικά, χρησιμοποιείται η προσέγγιση expect/actual.

Στο αρχείο DatabaseBuilder.kt δηλώνεται η expect συνάρτηση:

```kotlin
package com.traveltales.data.local.db

import androidx.room3.RoomDatabase

// Expect δήλωση για τη δημιουργία του Room Builder
expect fun getDatabaseBuilder(ctx: Any? = null): RoomDatabase.Builder<AppDatabase>
```

Στο αρχείο DatabaseBuilder.android.kt υλοποιείται η συνάρτηση για τις συσκευές android:

```kotlin
package com.traveltales.data.local.db

import android.content.Context
import androidx.room3.Room
import androidx.room3.RoomDatabase
import com.traveltales.util.BuildConstants

// Actual υλοποίηση για Android με χρήση Context
actual fun getDatabaseBuilder(ctx: Any?): RoomDatabase.Builder<AppDatabase> {
    // Εξαγωγή του Android Application Context
```

```
    val appContext = (ctx as Context).applicationContext
    val dbName = if (BuildConstants.USE_MOCK_DATA) "roam_tripper_mock.db" else
"roam_tripper.db"
    val dbFile = appContext.getDatabasePath(dbName)

    return Room.databaseBuilder<AppDatabase>(
        context = appContext,
        name = dbFile.absolutePath
    ).fallbackToDestructiveMigration(BuildConstants.IS_DEBUG) // Καταστροφή &
επαναδημιουργία σε debug
```

Η χρήση της fallbackToDestructiveMigration, αποτρέπει την αποτυχία της εφαρμογής όταν αλλάζει το schema για λόγους ανάπτυξης της εφαρμογής, στο περιβάλλον παραγωγής αυτή η ρύθμιση απενεργοποιείται για την αποφυγή διαγραφής δεδομένων του χρήστη.

6.3 Ασφαλής αποθήκευση και ρυθμίσεις του multiplatform

Η αποθήκευση ευαίσθητων δεδομένων όπως είναι τα JWT tokens ανανέωσης και πρόσβασης, αποτελεί ένα κρίσιμο σημείο για την ασφάλεια της εφαρμογής. Η αποθήκευση ευαίσθητων πληροφοριών δεν πρέπει ποτέ να γίνεται σε sharedPreferences, καθώς τα δεδομένα αποθηκεύονται σε απλό κείμενο XML και μπορεί εύκολα μια να γίνουν προσβάσιμα αν το κινητό είναι rooted ή με τη χρήση adb backup επιθέσεων.

Για την αντιμετώπιση αυτού του κινδύνου χρησιμοποιήθηκε η βιβλιοθήκη Multiplatform settings σε συνδυασμό με το jetpack security API της Google.

6.3.1 Η βιβλιοθήκη jetpack security

Η βιβλιοθήκη παρέχει αυτόματη κρυπτογράφηση δύο επιπέδων μέσω της κλάσης EncryptedSharedPreferences. Το πρώτο επίπεδο είναι η κρυπτογράφηση των κλειδιών ρυθμίσεων και η δεύτερη η κρυπτογράφηση των τιμών. Επιπλέον, το κλειδί που χρησιμοποιείται για την κρυπτογράφηση, αποθηκεύεται στο android keystore, το οποίο είναι στο φυσικό επίπεδο, καθιστώντας επιθέσεις λογισμικού αδύνατες.

6.3.2 Η υλοποίηση του secure settings provider

Ένα συχνό πρόβλημα που υπάρχει με τη χρήση του android keystore είναι η αλλοίωση των κλειδιών. Αυτό μπορεί να συμβεί με πολλούς τρόπους, όπου κάποιοι από τους οποίους είναι η αναβάθμιση του λειτουργικού συστήματος, η αλλαγή βιομετρικών δεδομένων και η επαναφορά της συσκευής από κάποιο backup. Αυτό θα έχει ως αποτέλεσμα η εφαρμογή να μην μπορεί να ανοίξει και να βγάζει συνέχεια σφάλμα. Το πρόβλημα αυτό λύθηκε με τη χρήση του fail closed, όπου στην περίπτωση που αλλοιωθούν τα κλειδιά, θα γίνει ολική διαγραφή αυτών και θα ξεκινήσει η διαδικασία σύνδεσης του χρήστη από την αρχή.

```
package com.traveltales.util

import androidx.security.crypto.EncryptedSharedPreferences
import androidx.security.crypto.MasterKey
import com.russhwolf.settings.Settings
```

```
import com. russhwolf.settings.SharedPreferencesSettings
import java.security.KeyStore

private const val SECURE_PREFS_FILE = "secure_auth_prefs"

// Actual παροχή Secure Settings για Android
actual fun provideSecureSettings(): Settings {
    return try {
        createEncryptedSettings()
    } catch (e: Exception) {
        // Διαδικασία ανάκαμψης σε περίπτωση αλλοίωσης του KeyStore
        runCatching { appContext.deleteSharedPreferences(SECURE_PREFS_FILE) }
        runCatching {
            val keystore = KeyStore.getInstance("AndroidKeyStore")
            keystore.load(null)
            keystore.deleteEntry(MasterKey.DEFAULT_MASTER_KEY_ALIAS)
        }
        createEncryptedSettings() // Επαναδημιουργία καθαρών κλειδιών
    }
}

// Δημιουργία κρυπτογραφημένων SharedPreferences
private fun createEncryptedSettings(): Settings {
    // 1. Δημιουργία Master Key στο KeyStore με AES256_GCM
    val masterKey = MasterKey.Builder(appContext)
        .setKeyScheme(MasterKey.KeyScheme.AES256_GCM)
        .build()

    // 2. Αρχικοποίηση των EncryptedSharedPreferences
    val prefs = EncryptedSharedPreferences.create(
        appContext,
        SECURE_PREFS_FILE,
        masterKey,
        EncryptedSharedPreferences.PrefKeyEncryptionScheme.AES256_SIV,
        EncryptedSharedPreferences.PrefValueEncryptionScheme.AES256_GCM
    )

    // 3. Επιστροφή KMP Settings wrapper
    return SharedPreferencesSettings(prefs)
}
```

6.4 Διασύνδεση της εφαρμογής με το backend

Η επικοινωνία με το backend είναι αρκετά εύκολη καθώς υπάρχει υποστήριξη μέσω του ktor client. Αυτός ο client είναι σχεδιασμένος ειδικά για KMP, ο οποίος υποστηρίζει ασύγχρονες I/O κλήσεις με των coroutines μετάφραση των JSON αρχείων σε αντικείμενα με τη χρήση της βιβλιοθήκης kotlinx.serialization.

6.4.1 Η στρατηγική του διπλού HTTP πελάτη.

Για την υλοποίηση της αυθεντικοποίησης του χρήστη, υλοποιήθηκε μια στρατηγική δύο πελατών. Ο ένας χρησιμοποιείται μόνο για αιτήματα ανανέωσης του token πρόσβασης, όπου δεν υποστηρίζει αυτόματους interceptors για τον τερματισμό ατέρμονων βρόχων. Ο δεύτερος, ο οποίος είναι και κύριο πελάτης για τα αιτήματα της εφαρμογής. Υποστηρίζει interceptors για την αυτόματη αποφυγή ατέρμονων βρόχων αλλά και για τον χειρισμό αποτυχημένων αιτημάτων.

6.4.2 Η ανανέωση των tokens

Για την καλύτερη εμπειρία χρήστη, δημιουργήθηκε ένας custom interceptor ο οποίος ελέγχει αν η ημερομηνία λήξης του token πλησιάζει. Στην περίπτωση που η λήξη του είναι κοντά, στέλνει ένα αίτημα για την ανανέωση του.

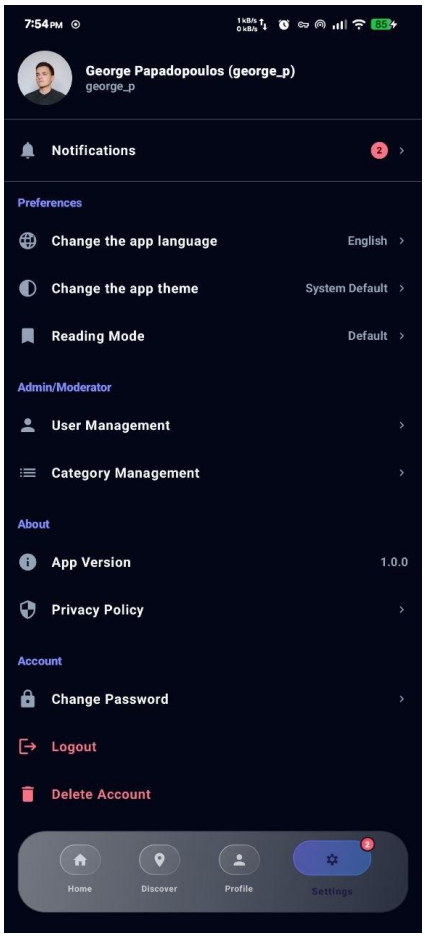
```
// Απόσπασμα από το NetworkModule.kt
install(createClientPlugin("ProactiveAuth") {
    onRequest { request, _ ->
        val backendHost = Url(baseUrl).host
        val path = request.url.encodedPath
        val isAuthExcluded = path.contains("login") || path.contains("signup") ||
path.contains("auth")

        if (request.url.host == backendHost && !isAuthExcluded) {
            // Έλεγχος εάν το αποθηκευμένο token έχει λήξει
            if (sessionManager.token != null && sessionManager.isTokenExpired()) {
                infoLog("Auth", "ProactiveAuth: Token is nearing expiry. Refreshing BEFORE
request...")
                val currentToken = sessionManager.token

                // Εκτέλεση σύγχρονης ανανέωσης μέσω του Refresh Client
                val result = performRefresh(currentToken, sessionManager,
authSessionManager, refreshApi)

                // Εάν η ανανέωση πετύχει, εισάγουμε το νέο token στο Header της αίτησης
                result?.accessToken?.let { newToken ->
                    request.header(HttpHeaders.Authorization, "Bearer $newToken")
                }
            }
        }
    }
})
```

6.4.3 Network resilience

Για να αντιμετωπιστούν προβλήματα προσωρινής αποτυχίας του δικτύου, χρησιμοποιήθηκε το HttpRequestRetry plugin. Με τη χρήση αυτού του plugin, κάθε αίτημα έχει έναν μέγιστο αριθμό επαναλήψεων, επιλογή σε τι σφάλματα θα γίνονται οι επαναλήψεις αλλά και η χρήση εκθετικής καθυστέρησης (σε περίπτωση πολλών πελατών θα μπορούσε να μπει και jitter για την αντιμετώπιση του thundering herd).

6.5 Υπηρεσίες τοποθεσίας και παρακολούθηση τοποθεσίας

Ένα από τα βασικά χαρακτηριστικά της εφαρμογής TravelTales είναι η δυνατότητα εύρεσης ιστοριών που έχουν δημοσιευθεί κοντά στον στην τοποθεσία του χρήστη. Για να μπορέσει να γίνει αυτό με έναν αποδοτικό τρόπο, αναπτύχθηκε η κλάση AndroidLocationTracker, η οποία χρησιμοποιεί τα callbacks του android με έναν ασύγχρονο τρόπο.



Equation 7

6.5.1 Υλοποίηση του android location tracker

Η κλάση AndroidLocationTracker υλοποιεί τη διεπαφή LocationTracker:

```
package com.traveltales.domain.location

import android.annotation.SuppressLint
import android.content.Context
import android.location.Location
import android.location.LocationListener
import android.location.LocationManager
import com.traveltales.domain.util.Result
import kotlinx.coroutines.channels.awaitClose
import kotlinx.coroutines.flow.Flow
import kotlinx.coroutines.flow.callbackFlow
```

```
class AndroidLocationTracker(private val context: Context) : LocationTracker {

    private val locationManager by lazy {
        context.getSystemService(Context.LOCATION_SERVICE) as LocationManager
    }

    // Παρατήρηση της τοποθεσίας σε πραγματικό χρόνο ως Flow
    @SuppressLint("MissingPermission")
    override fun observeLocation(): Flow<Result<UserLocation, LocationError>> =
callbackFlow {
        // 1. Έλεγχος δικαιωμάτων (Runtime Permissions)
        if (!context.hasLocationPermission()) {
            trySend(Result.Error(LocationError.PermissionDenied))
            close()
            return@callbackFlow
        }

        // 2. Δημιουργία του Location Listener
        val listener = object : LocationListener {
            override fun onLocationChanged(location: Location) {
                // Αποστολή των συντεταγμένων στο Flow stream
                trySend(Result.Success(location.toUserLocation()))
            }
            override fun onProviderDisabled(provider: String) {
                trySend(Result.Error(LocationError.GpsDisabled))
            }
        }

        try {
            // 3. Εγγραφή για λήψη ενημερώσεων από GPS και Network Providers
            locationManager.requestLocationUpdates(LocationManager.GPS_PROVIDER, 5000L,
10f, listener)
            locationManager.requestLocationUpdates(LocationManager.NETWORK_PROVIDER, 5000L,
10f, listener)
        } catch (e: Exception) {
            trySend(Result.Error(LocationError.Unknown))
            close()
        }

        // 4. Απελευθέρωση πόρων κατά τον τερματισμό του Flow (Clean up)
        awaitClose {
            locationManager.removeUpdates(listener)
        }
    }
}
```

6.5.2 Μετατροπή callback σε ασύγχρονα flow με callbackflow

Το callbackFlow είναι ένα εξαιρετικά προηγμένο εργαλείο το οποίο βασίζεται στις coroutines της Kotlin και επιτρέπει τη χρήση παλιότερων API με ένα σύγχρονο τρόπο. Η μέθοδος awaitClose είναι κρίσιμη για την αποφυγή υπερβολικής χρήσης μνήμης, διασφαλίζει πως όταν η παρακολούθηση της τοποθεσίας σταματήσει θα αφαιρεθεί από τη μνήμη ο LocationListener, χρησιμοποιώντας λιγότερη μπαταρία ως αποτέλεσμα.

6.6 Η αλληλεπιδραστική διεπαφή χρήστη και το jetpack compose multiplatform

Για την ανάπτυξη όλων των γραφικών έχει χρησιμοποιηθεί το compose multiplatform. Η compose είναι μια δηλωτική βιβλιοθήκη σχεδίασης γραφικών όπου έχει γίνει αρκετά διάσημη τα τελευταία χρόνια.

6.6.1 Αρχιτεκτονική μοντελοποίησης της κατάστασης των γραφικών

Για την ανάπτυξη της εφαρμογής χρησιμοποιήθηκε το πρότυπο MVVM (Model View View Model) με UDF (Unidirectional Data Flow). Βάση αυτών των δύο, τα κοινά γραφικά μοντέλα έχουν μια μοναδική και αμετάβλητη κατάσταση. Η βασική αλληλεπίδραση με το γραφικό γίνεται μέσω του intent/action. Επιπλέον, τα composables παρακολουθούν την κατάσταση τους (UiState) έτσι όστε να επανασχεδιάζονται αυτόματα όταν αυτή αλλάζει.

```
// Παράδειγμα δήλωσης UiState στο commonMain
data class StoryListUiState(
    val stories: List<StoryUiModel> = emptyList(),
    val isLoading: Boolean = false,
    val error: String? = null
)
```

6.6.2 Το δυναμικό σύστημα θεμάτων και τα Material 3 animations

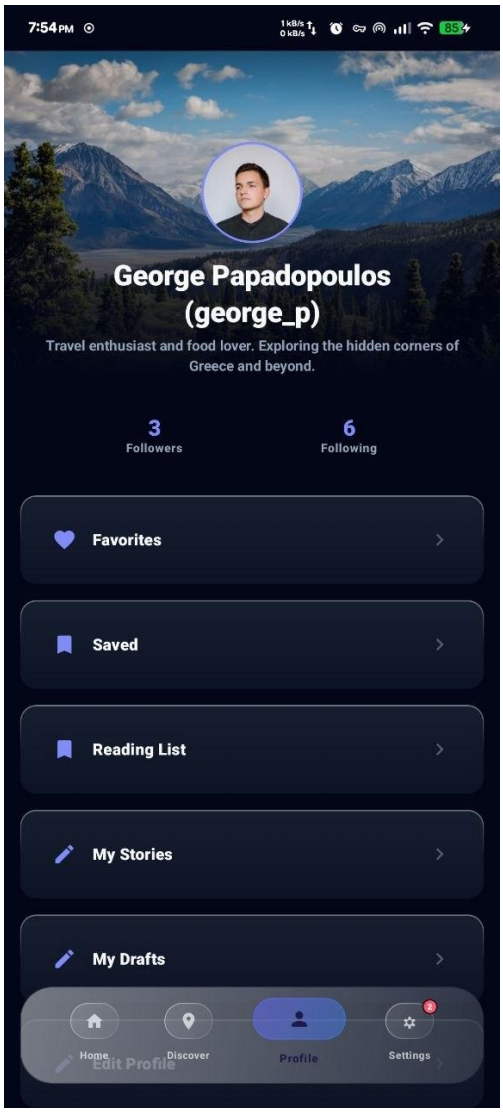
Για τον δυναμικό τρόπο αλλαγής θεμάτων στην εφαρμογή χρησιμοποιήθηκε το πρότυπο του Material design 3. Η υλοποίηση του έγινε στην κλάση Theme.kt όπου υπάρχει ένα σύστημα μετάβασης θεμάτων με τη χρήση animations.

```
@Composable
fun AppTheme(themeMode: ThemeMode = ThemeMode.SYSTEM, content: @Composable () -> Unit) {
    val darkTheme = when (themeMode) {
        ThemeMode.LIGHT -> false
        ThemeMode.DARK -> true
        ThemeMode.SYSTEM -> isSystemInDarkTheme()
    }

    // Επιλογή του κατάλληλου χρωματικού σχήματος (Light / Dark)
    val targetColorScheme = if (darkTheme) DarkColors else LightColors

    // Ομαλή μετάβαση χρωμάτων (500ms tween animation)
    val colorScheme = animateColorScheme(targetColorScheme)

    SystemAppearance(darkTheme) // Ρύθμιση Status/Navigation Bars
```

```
    val extendedColors = ExtendedColors(
        success = ColorSuccess,
        roleAdmin = ColorRoleAdmin,
        roleModerator = ColorRoleModerator,
        roleVerified = ColorRoleVerified,
        roleUnverified = ColorRoleUnverified,
        shimmerBase = ColorShimmerBase,
        shimmerHighlight = ColorShimmerHighlight,
        warning = ColorWarning,
        onImage = ColorOnImage,
        imageScrim = ColorImageScrim
    )

    // Παροχή adaptive window sizes και χρωμάτων
    WindowSizeClassProvider {
        CompositionLocalProvider(LocalExtendedColors provides extendedColors) {
            MaterialTheme(
                colorScheme = colorScheme,
                typography = AppTypography,
                shapes = AppShapes,
                content = content
            )
        }
    }
}
```

6.6.3 Προσαρμοζόμενα γραφικά και η κλάση WindowSizeClassProvider

Ένα από τα σημαντικά χαρακτηριστικά για τη σχεδίαση των γραφικών διεπαφών είναι η χρήση της κλάσης WindowsSizeClassProvider, όπου κατά την εκτέλεση της εφαρμογής τρέχει ελέγχους για την κατηγοριοποίηση του μεγέθους της οθόνης της συσκευής. Χρησιμοποιώντας αυτές τις πληροφορίες μπορούμε να προσαρμόσουμε δυναμικά τις γραφικές διεπαφές. Δύο σημαντικές κατηγορίες είναι το compact screen για οθόνες κινητών και το expanded screen για οθόνες από tablets. Μια σημαντική διαφορά μεταξύ των δύο είναι πως στο compact screen η μπάρα πλοήγησης εμφανίζεται στο κάτω μέρος της οθόνης αλλά στο expanded screen εμφανίζεται στα αριστερά της οθόνης.



Equation 8

6.7 Συμπεράσματα

Η υλοποίηση της εφαρμογής με τη χρήση του KMP δείχνει πως μπορεί να αναπτυχθεί μια εφαρμογή η οποία μπορεί να χρησιμοποιήσει κώδικα μεταξύ διαφορετικών πλατφορμών χωρίς να είναι πιο αργή ή πιο βαριά. Η μεγιστοποίηση της επαναχρησιμοποίησης κώδικα, όπως της δικτυακής λογικής, του room caching schema και τη διαχείριση της κατάστασης, κάνει πιο εύκολη την υλοποίηση εφαρμογών για πολλαπλές πλατφόρμες, μειώνοντας σημαντικά την πιθανότητα για περισσότερα σφάλματα αλλά διευκολύνει και τις μελλοντικές επεκτάσεις.

Η ασφάλεια των ευαίσθητων δεδομένων γίνεται αρκετά εύκολο ζήτημα χρησιμοποιώντας την ασφάλεια που προσφέρει το keystore, το οποίο διασφαλίζει την ασφάλεια των JWT tokens. Τέλος, βλέπουμε πως ξανά οι coroutines της Kotlin κάνει πιο αποδοτικό τον κώδικα χωρίς να χρειάζεται ο προγραμματιστής να δημιουργήσει σύνθετο κώδικα για να μπορέσει να τρέχει ταυτόχρονα πολλαπλές διεργασίες.

Κεφάλαιο 7ο: Ενοποίηση συστήματος

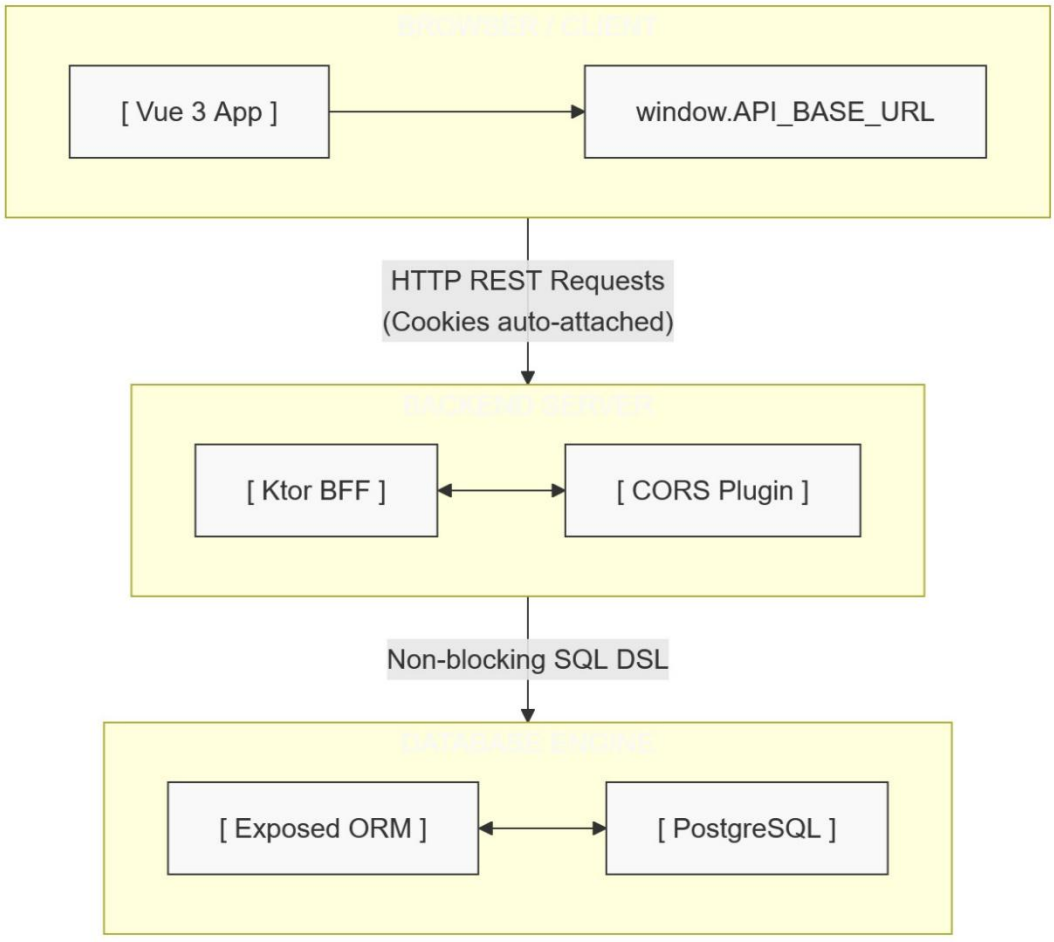
Στη φάση ενοποίησης του συστήματος προσπαθούμε να ενοποιήσουμε τα επιμέρους δομικά στοιχεία μεταξύ τους. Για την εφαρμογή TravelTales αυτά τα στοιχεία είναι το backend, το frontend, βάση δεδομένων και υπηρεσίες όπως το Google OAuth. Στο τελικό αποτέλεσμα θα πρέπει αυτά τα επιμέρους στοιχεία να λειτουργούν ως μια ενιαία ασφαλή πλατφόρμα.

Συνοπτικά στο παρόν κεφάλαιο θα αναφερθούν οι τρόποι με τους οποίους επιτεύχθηκε η διασύνδεση των τμημάτων του TravelTales. Θα αναφερθούν πιο αναλυτικά η πολιτική CORS (Cross Origin Resource Sharing), η ενοποίηση με τη βάση δεδομένων και η αρχικοποίηση της, η διασύνδεση με το Google OAuth και WebAuthn και τέλος, οι διαφορές μεταξύ του περιβάλλοντος ανάπτυξης και παραγωγής.

7.1 Αρχιτεκτονική διασύνδεση

Η επικοινωνία μεταξύ του frontend και του backend γίνεται μέσω ασύγχρονων κλήσεων HTTP REST. Η ιστοσελίδα τρέχει στον περιηγητή του χρήστη, η εφαρμογή στο κινητό του χρήστη και το backend τρέχει στον διακομιστή ως αυτόνομη υπηρεσία στον περιβάλλον JVM.

Σχήμα 3: Αρχιτεκτονική διασύνδεση



Για να είναι πιο ευέλικτο το API, δεν αναγράφεται στα end points το base url αλλά φορτώνεται δυναμικά με τη χρήση της μεταβλητής window.API_BASE_URL για το frontend. Στην περίπτωση που η εφαρμογή τρέχει σε περιβάλλον ανάπτυξης, τότε το base url θα είναι το localhost:8080, διαφορετικά αν είναι παραγωγής θα πάρει το ανάλογο domain που έχει οριστεί, χωρίς να υπάρχει ανάγκη για αλλαγές στον πηγαίο κώδικα.

7.2 Πολιτική CORS (Cross Origin Resource Sharing)

Εφόσον το frontend και το backend εκτελούνται σε διαφορετικές θύρες, πρέπει να ρυθμιστεί το CORS αναλόγως για να επιτρέπει ο περιηγητής την πρόσβαση σε πόρους από διαφορετικές προελεύσεις (αυτό συμβαίνει γιατί ο περιηγητής επιβάλλει τη Same Origin Policy). Αν δεν ορίσουμε τίποτα, το frontend δε θα μπορέσει να έχει πρόσβαση στα δεδομένα που στέλνει το backend, εκτός αν δηλώνεται ρητά στις επικεφαλίδες των αιτημάτων HTTP.

7.2.1 Το BFF μοντέλο για τα CORS

Σε ένα σύστημα το οποίο χειρίζεται την αυθεντικοποίηση με τη χρήση cookies, η ρύθμιση του CORS είναι εξαιρετικά σημαντικό και ευαίσθητο θέμα. Για αυτόν τον λόγο δε θα πρέπει ποτέ να χρησιμοποιούνται wildcards για τη ρύθμιση του access control allow origin. Πάντα θα πρέπει να ορίζεται η διεύθυνση του αιτούντος. Επιπλέον, για να μπορέσει να χρησιμοποιήσει ο περιηγητής τα cookies για τη μεταφορά credentials θα πρέπει να υπάρχει η κεφαλίδα access control allow credentials με τιμή true.

7.2.2 Ρύθμιση του CORS plugin

Η ρύθμιση του CORS στην εφαρμογή TravelTales γίνεται στο αρχείο CORS.kt

```
fun Application.configureCORS() {
    install(CORS) {
        allowCredentials = true // Επιτρέπει τη μεταφορά HttpOnly Cookies

        // 1. Εγκεκριμένα Domains Ανάπτυξης (Development)
        allowHost("localhost:5173")
        allowHost("127.0.0.1:5173")

        // 2. Εγκεκριμένα Domains Παραγωγής (Production)
        allowHost("traveltales.site:5173", schemes = listOf("http", "https"))
        allowHost("traveltales.site:8080", schemes = listOf("http", "https"))
        allowHost("traveltales.site:80", schemes = listOf("http", "https"))
        allowHost("traveltales.site", schemes = listOf("http", "https"))

        // 3. Εγκεκριμένες IPs Παραγωγής (Production Fallback)
        allowHost("95.164.212.4:5173", schemes = listOf("http", "https"))
        allowHost("95.164.212.4:8080", schemes = listOf("http", "https"))
        allowHost("95.164.212.4", schemes = listOf("http", "https"))

        // Εγκεκριμένες Μέθοδοι HTTP (CRUD Operations)
        allowMethod(HttpMethod.Get)
```

```
        allowMethod(HttpMethod.Post)
        allowMethod(HttpMethod.Options) // Απαραίτητο για το Preflight request
        allowMethod(HttpMethod.Put)
        allowMethod(HttpMethod.Delete)
        allowMethod(HttpMethod.Patch)

        // Εγκεκριμένα Headers
        allowHeader(io.ktor.http.HttpHeaders.ContentType)
        allowHeader(io.ktor.http.HttpHeaders.Authorization)
        allowHeader(io.ktor.http.HttpHeaders.CacheControl)
    }
}
```

7.2.3 Ανάλυση ροής preflight (HTTP options)

Όταν η vue στέλνει ένα αίτημα POST ή DELETE, ο περιηγητής στέλνει πρώτα ένα αίτημα (Preflight request) με σκοπό να δει αν το αίτημα που θέλει να στείλει είναι ασφαλές. Το plugin CORS του Ktor, δημιουργεί και στέλνει και απαντά σε αυτά τα αιτήματα αυτόματα. Όταν η πηγή ενός αιτήματος είναι στις επιτρεπτές διευθύνσεις, το αίτημα συνεχίζει τη ζωή του στα πιο εσωτερικά τμήματα του Ktor. Στην περίπτωση που δεν υπάρχει η πηγή του αιτήματος στις επιτρεπτές διευθύνσεις, τότε το αίτημα δε συνεχίζει και τη θέση του την παίρνει μια απάντηση σφάλματος από cross origin violation.

7.3 Βάση δεδομένων

Η βάση δεδομένων αρχικοποιείται από το backend στην περίπτωση που δεν είναι, μέσα από το plugin Exposed ORM. Με αυτόν τον τρόπο το schema της βάσεις και το μοντέλο των δεδομένων στο backend είναι πάντα το ίδιο. Επίσης, τυχόν μελλοντικές αλλαγές στο schema της βάσης δεδομένων γίνονται αυτόματα.

7.3.1 Η αρχικοποίηση με SQL

Όπως αναφέραμε δεν υπάρχει η ανάγκη για χειροκίνητη αρχικοποίηση της βάσης δεδομένων, με εξαίρεση την ενεργοποίηση του UUID.

```
CREATE DATABASE traveltales;

\connect traveltales

-- Ενεργοποίηση της επέκτασης για αυτόματη παραγωγή UUID κλειδιών
CREATE EXTENSION IF NOT EXISTS "uuid-ossp";

CREATE SCHEMA IF NOT EXISTS app;
```

Η χρήση ενός κρυπτογραφικά τυχαίου αριθμού είναι αναγκαία για να αποτρέψει τυχόν επιθέσεις που βασίζονται σε απαριθμήσεις ID. Για αυτόν τον λόγο χρησιμοποιούμε το UUID v4 της PostgreSQL. Έτσι ένας κακόβουλος χρήστης δεν μπορεί να μαντέψει τα IDs άλλων χρηστών απλώς αυξάνοντας τον αριθμό.

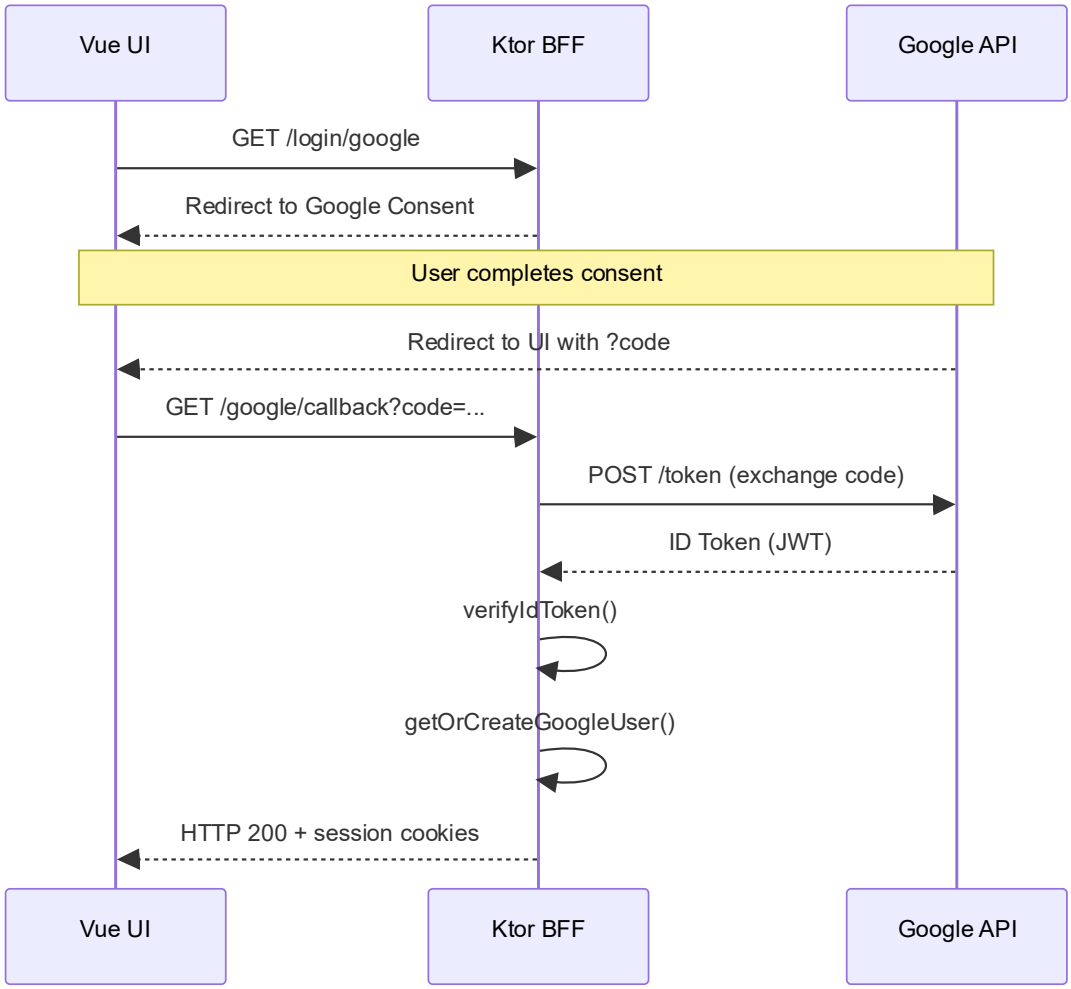
7.3.2 Exposed schema mapping

Στο backend, ο ορισμός των πινάκων της βάσης δεδομένων γίνεται με τη χρήση αντικειμένων της Kotlin. Για παράδειγμα, η δημιουργία του πίνακα webauthn_credentials χρησιμοποιεί απλές μεταβλητές με κάποιες έξτρα παραμέτρους. Όπως είναι η παράμετρος onDelete για τη μεταβλητή userId, η οποία ορίζει το ReferenceOption.CASCADE. Αυτό το όρισμα εξασφαλίζει πως κλάσεις που χρησιμοποιούν το πεδίο userId, στην περίπτωση που γίνει διαγραφή του χρήστη όλες οι εγγραφές που κάνουν αναφορά σε αυτό το πεδίο θα διαγραφούν. Με αυτόν τον τρόπο δε θα αφήσουμε ορφανές εγγραφές στη βάση δεδομένων.

```
object WebAuthnCredentialsTable : UUIDTable("app.webauthn_credentials") {
    val userId = reference("user_id", UserTable, onDelete = ReferenceOption.CASCADE)
    val credentialId = varchar("credential_id", 512).uniqueIndex()
    val publicKey = text("public_key")
    val signCount = long("sign_count")
    val name = varchar("name", 100)
    val createdAt = long("created_at")
    val updatedAt = long("updated_at")
}
```

7.4 Ενοποίηση του Google OAuth 2.0

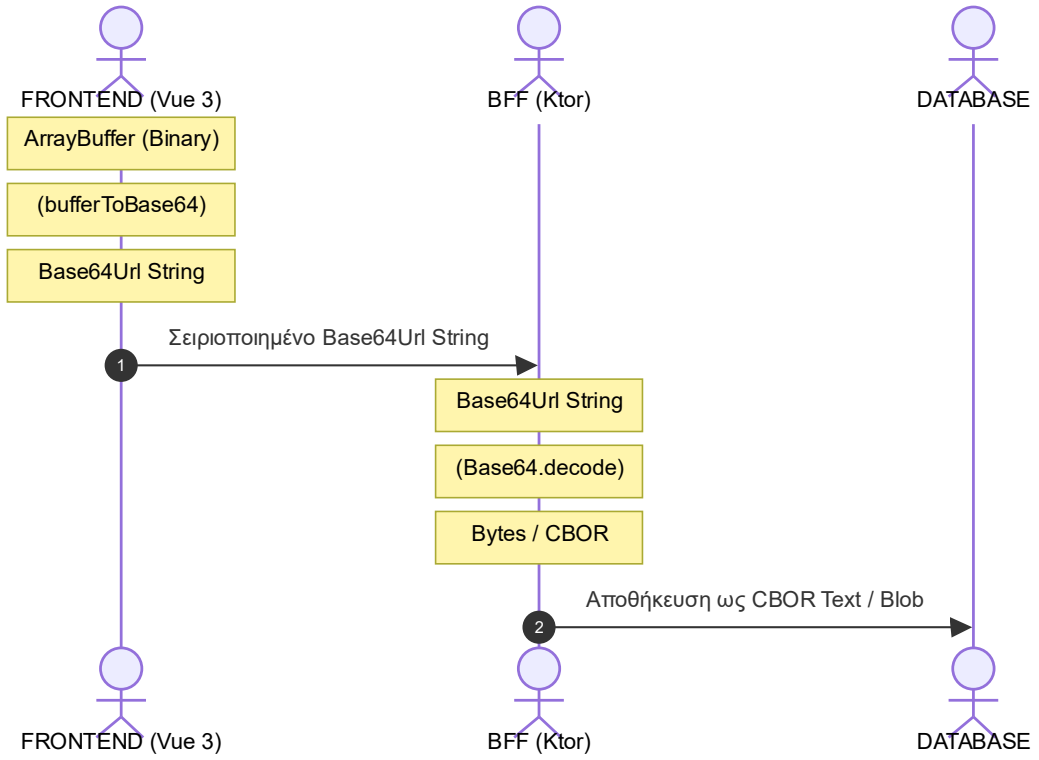
Η ενοποίηση του Google OAuth 2.0, απαιτεί τον τέλειο συντονισμό μεταξύ των τμημάτων της εφαμοργής TravelTales και της Google υπηρεσίας. Όταν ο χρήστης θέλει να συνδεθεί χρησιμοποιώντας τον λογαριασμό google που διαθέτει, επιλέγει το κουμπί "Google Login", με αποτέλεσμα το frontend να κάνει ανακατεύθυνση του περιηγητή στο end point του backend /login/google. Με τη σειρά του, το backend θα κάνει ανακατεύθυνση τον περιηγητή στο end point της google, ενσωματώνοντας το google_client_id και το redirect_uri. Όταν ο χρήστης συνδεθεί επιτυχώς, η google ανακατευθύνει τον περιηγητή στο callback (traveltales-url/login/google/callback) ενσωματώνοντας τις απαραίτητες πληροφορίες για τον χρήστη. Κατόπιν, εσωτερικά του backend γίνεται έλεγχος για το αν υπάρχει ο χρήστης ή πρέπει να γίνει εγγραφή του χρήστη. Μετά δημιουργούνται τα απαραίτητα tokens και στέλνονται ως HttpOnly cookies στον περιηγητή του χρήστη.



Σχήμα 4: Ενοποίηση του Google OAuth 2.0

7.5 Ενοποίηση WebAuthn

Η ενοποίηση του WebAuthn αποτελεί ένα πιο απλή διαδικασία καθώς χρειάζεται μόνο να γίνει μια απλή μετατροπή των κρυπτογραφικών δεδομένων από array buffers (περιηγητής) σε base64url (μεταφορά με HTTP) και τελικά να αποθηκευτεί ως CBOR/ByteArray στη βάση δεδομένων.



Σχήμα 5: Μετατροπή των κρυπτογραφικών δεδομένων

Στην εγγραφή ο περιηγητής καλεί το end point /2fa/register/star, το οποίο έχει ως αποτέλεσμα το backend να δημιουργήσει ένα μια πρόκληση και να την αποθηκεύσει στη συνεδρία και στέλνει πίσω την απάντηση στον περιηγητή. Γίνονται οι απαραίτητες μετατροπές, απαντάει στην πρόκληση, όπου γίνεται αντίστροφα η μετατροπή και στην περίπτωση που είναι όλα σωστά το backend αποθηκεύει το δημόσιο κλειδί που δημιουργήθηκε σε CBOR-encoded bytes στη βάση δεδομένων.

Στην περίπτωση της σύνδεσης, ο περιηγητής καλεί το /login/local, το backend διαπιστώνει πως ο χρήστης έχει ενεργό το 2fa και απαντάει με μια κρυπτογραφική πρόκληση. Στη συνέχεια ο περιηγητής ζητάει επιβεβαίωση από τον χρήστη και όταν τη λάβει απαντάει στην κρυπτογραφική πρόκληση και το στέλνει στο /2fa/login/finish. Το backend με τη σειρά του κάνει του κατάλληλους ελέγχους και ενημερώνει την κατάσταση των κλειδιών (ενημερώνει το μετρητή signCount).

7.6 Διαφορές του περιβάλλον ανάπτυξης με παραγωγής

Για τη σωστή ενοποίηση του συστήματος θα πρέπει να υπάρχουν τρόποι για τη ρύθμιση του συστήματος ανάλογα σε πιο περιβάλλον τρέχει. Για το backend, χρησιμοποιείται το αρχείο application.yaml για τις απαιτούμενες ρυθμίσεις.

```
ktor:
    deployment:
        port: 8080
    application:
        modules: [ gr.traveltales.ApplicationKt.module ]
jwt:
    secret: ${JWT_SECRET}
    issuer: ${ISSUER}
    audience: ${AUDIENCE}
```

```
    realm: ${REALM}
```

Στο περιβάλλον της παραγωγής, οι τιμές του ${JWT_SECRET}, ${ISSUER}, ${AUDIENCE} και ${REALM} αλλάζουν αυτόματα με τις πραγματικές χρησιμοποιώντας μεταβλητές περιβάλλοντος.

7.7 Συμπεράσματα

Η ενοποίηση του συστήματος μπορεί να γίνει ένα δύσκολο ζήτημα αλλά χρησιμοποιώντας τα κατάλληλα πρότυπα και τεχνολογίες κάποιος μπορεί να κάνει την ενοποίηση πιο εύκολη. Όταν τα επιμέρους τμήμα ενός συστήματος έχουν οργανωθεί και διασπαστεί κατάλληλα, τότε η ενοποίηση, η αποσφαλμάτωση και ανάπτυξη του συστήματος γίνεται πιο εύκολη.

Κεφάλαιο 8ο: Συμπεράσματα και Μελλοντική Εργασία

Με την ολοκλήρωση των σταδίων σχεδίασης, ανάπτυξης, ενοποίησης και αξιολόγησης της εφαρμογής TravelTales, σηματοδοτήθηκε η ολοκλήρωση μια σύγχρονης full stack αρχιτεκτονικής που θέτει στο επίκεντρο την ασφάλεια των χρηστών. Το TravelTales δεν αποτελεί απλώς μια τυπική διαδικτυακή εφαρμογή αλλά αποτελεί μια ολοκληρωμένη μελέτη αναγκών και περιπτώσεων γιατί τη σωστή ανάπτυξη μιας εφαρμογής που ενσωματώνει σύγχρονες τεχνολογίες ασφάλειας, οι οποίες δουλεύουν αρμονικά με μια καλή εμπειρία χρήστη.

Σε αυτό το τελευταίο κεφάλαιο, παρουσιάζονται αναλυτικά τα πλεονεκτήματα της εφαρμογής TravelTales στα επιμέρους τμήμα της. Αυτό συμπεριλαμβάνει την αρχιτεκτονική, τους τεχνικούς περιορισμούς και τις προκλήσεις που αντιμετωπίστηκαν για την ανάπτυξη της εφαρμογής. Επιπλέον, αναφέρονται μελλοντικές επεκτάσεις που μπορούν να αναβαθμίσουν ακόμη περισσότερο την εμπειρία χρήστη, την ασφάλεια αλλά και το καλύτερο οριζόντιο scalability της εφαρμογής.

8.1 Σύνοψη και αποτίμηση του έργου

Στον πυρήνα της σχεδίασης της εφαρμογής TravelTales είναι το πρότυπο BFF (Backend for froentend), το οποίο διαχωρίζει πλήρως τη δυναμική αλληλεπίδραση των διεπαφών χρήστη από το επιχειρηματικό επίπεδο του backend. Αξίζει να σημειωθεί πως ένας επιπλέον λόγος που επιλέχθηκε το BFF πρότυπο είναι επειδή η αρχική σχεδίαση της εφαρμογής θα γινόταν με δύο backend. Το ένα θα εξυπηρετούσε κινητές συσκευές και το άλλο την ιστοσελίδα. Με τη σειρά τους θα επικοινωνούσαν με άλλα micro services, όπου το κάθε micro service θα είχε ένα μικρό σύνολο ευθυνών. Αυτό θα επέτρεπε το καλύτερο οριζόντιο scalability αλλά λόγο του χρονικού πλαισίου δεν υλοποιήθηκε, το οποίο θα μπορεί να υλοποιηθεί μελλοντικά. Τα θετικά της πιο απλής υλοποίησης και της μη διάσπασης της εφαρμογής σε επιμέρους κομμάτια, προσφέρει πιο γρήγορες επιδώσεις για μικρότερο αριθμό πελατών.

8.2 Πλεονεκτήματα του συστήματος

Η αρχιτεκτονική προσέγγιση που χρησιμοποιεί το TravelTales προσφέρει σημαντικά πλεονεκτήματα σε σύγκριση με άλλες μονολιθικές ή SPA (single page application) προσεγγίσεις. Τα πλεονεκτήματα αξίζουν να σημειωθούν είναι η ασφάλεια που παρέχεται, η υψηλή απόδοση με τη χρήση ασύγχρονων νημάτων αλλά και η βελτιστοποιημένη εμπειρία χρήστη (UX).

Πίνακας 9

SECURITY FIRST	Total XSS Immunity: JWTs are physically unreachable via document.cookie
SECURITY FIRST	Total CSRF Immunity: Strict SameSite=Lax cookie constraints
SECURITY FIRST	Phishing-Resistant: WebAuthn Origin Binding (rpId)
PERFORMANCE & SCALABILITY	Asynchronous Non-blocking Engine: Ktor & Kotlin Coroutines
PERFORMANCE & SCALABILITY	Highly Optimized DB Access: Exposed DSL with suspendTransaction
PERFORMANCE & SCALABILITY	Tree-Shaken & Fast Assets: Vite Build Pipeline
USER EXPERIENCE (UX)	Passwordless Biometrics: TouchID / FaceID keyless entry
USER EXPERIENCE (UX)	Seamless Session: Silent token refresh via Fetch API Interceptor
USER EXPERIENCE (UX)	Instant Visual Feedback: PrimeVue 4 accessible components

Δύο από τα σημαντικότερα προτερήματα της ασφάλειας που προσφέρει η εφαρμογή TravelTales είναι η προστασία από XSS (cross site scripting) και CSRF (cross site request forgery), χάρης τη χρήση των JWT tokens πρόσβασης και ανανέωσης που χρησιμοποιούν αποκλειστικά HttpOnly, Secure και SameSite=Lax cookies. Επιπλέον, η κρυπτογραφική ανθεκτικότητα που υπάρχει λόγο της χρήσης του Argon2id αλλά και τη χρήση πολυπαραγοντικής ταυτοποίησης

Η υψηλή απόδοση που επιτυγχάνεται με τη χρήση των coroutines της Kotlin οι οποίες επιτρέπουν την ταυτόχρονη εκτέλεση των λειτουργιών σε νήματα. Επιπλέον, η Exposed ORM που επιτρέπει την ταυτόχρονη εκτέλεση ερωτημάτων SQL μέσω του Dispatchers.IO έτσι ώστε να μπορούν να λειτουργήσουν και στα ερωτήματα οι coroutines.

8.3 Περιορισμοί και προκλήσεις της υλοποίησης

Η ανάπτυξη του συστήματος του TravelTales εμφάνισε πολλά προβλήματα και δυσκολίες κατά την ανάπτυξη του. Παρουσιάστηκαν πολλοί τεχνική περιορισμοί με αποτέλεσμα να γίνει επανασχεδίαση της εφαρμογής αλλά και να πεταχτεί κώδικας που πήρε μεγάλα χρονικά διαστήματα για την ανάπτυξη του.

Ένας από τους πρώτους τεχνικούς περιορισμού που εμφανίστηκε ήταν η ασυμβατότητα των binary buffers, που επιστρέφει ο περιηγητής χρησιμοποιώντας το web API του, με τη μεταφοράς τους μέσω HTTP. Αυτό είχε ως αποτέλεσμα τη δημιουργία των base64url για την κωδικοποίηση και αποκωδικοποίηση των binary buffers. Η υλοποίηση έπρεπε να γίνει για το backend αλλά και για το frontend, επιπλέον υπήρχε η απαίτηση για αυστηρές δοκιμές για την αποφυγή αλλοίωσης των κατά τη μεταφορά.

Προβλήματα εμφανίστηκαν και κατά τη διάρκεια της ανάπτυξης της εφαρμογής ως stateless. Έγινε γρήγορα αντιληπτό πως ο μηχανισμός ταυτοποίησης ήταν ευπαθείς σε επιθέσεις replay. Για αυτόν τον λόγο έγινε υποχρεωτική η χρήση state στην εφαρμογή. Αυτό δημιούργησε την ανικανότητα του συστήματος να κλιμακωθεί οριζόντια χρησιμοποιώντας έναν load balancer μπροστά από πολλαπλά instances. Υπάρχουν τεχνικές (sharding by parity ή deterministic routing based on key parity) που μπορούν να διαμοιράσουν την κίνηση έτσι ώστε να υπάρχει μια μικρή κλιμάκωση αλλά δεν είναι ιδανικό.

Ένα τελευταίο πρόβλημα που δεν έχει κάποια λύση, όσο αναφορά το πλαίσιο της εφαρμογής, είναι ο περιορισμός και ασυμβατότητα των λειτουργικών συστημάτων με το WebAuthn. Στην περίπτωση που δεν υπάρχει υποστήριξη για σύνδεση μέσω κάποιου φυσικού κλειδιού από το λειτουργικό σύστημα ή από τον περιηγητή (αν και οι περισσότεροι το υποστηρίζουν) τότε δε θα δουλέψει το 2FA.

8.4 Μελλοντικές επεκτάσεις και βελτιώσεις

Ένα από τα πρώτα πράγματα που θα ήταν καλό να γίνουν, είναι η αφαίρεση του session από το Ktor και να δημιουργηθεί ένα νέο session το οποίο θα υλοποιείται σε Redis. Με αυτόν τον τρόπο θα λυθεί το πρόβλημα της κλιμάκωσης, καθώς το Ktor θα μπορεί να έχει πολλαπλά instances χωρίς να δημιουργείται κάποιο πρόβλημα, αλλά ούτε από τη μεριά του Redis θα υπάρχει κάποιο πρόβλημα καθώς υποστηρίζει clustering.

Ένα σημαντικό κενό ασφάλειας που υπάρχει στην εφαρμογή είναι οι επιθέσεις session Hijacking. Όπου αν κάποιος αποκτήσει πρόσβαση στη βάση δεδομένων των cookies που έχει ο περιηγητής, μπορεί να τα χρησιμοποιήσει για να έχει πρόσβαση στην εφαρμογή. Ένας τρόπος που μπορεί να διορθωθεί είναι το Demonstrating Proof of Possession (DPoP) το οποίο ορίζεται στο RFC 9449. Βάση αυτού το frontend υπογράφει κάθε αίτημα με ένα ζεύγος κλειδιών και το backend τα επαληθεύει, έτσι ακόμη και να κλαπούν τα cookies δε θα έχει πρόσβαση ο χρήστης στο ιδιωτικό κλειδί της συσκευής.

Μια ωραία επιπρόσθετη λειτουργία θα ήταν η καλύτερη τμηματοποίηση του κώδικα και παραμετροποίηση του έτσι ώστε να υπάρχει καλύτερη συνεργασία με προγράμματα όπως το kubernetes. Με αυτόν τον τρόπο θα υπάρχει καλύτερο και αυτόματο scalability, δίνοντας καλύτερες επιδώσεις και πιο οικονομικό hosting σε δομές όπως το AWS και Azure.

8.5 Συμπεράσματα

Με την εκπόνηση της παρούσας πτυχιακής εργασίας ως proof of concept, μπορούμε να έχουν σύγχρονες διαδικτυακές εφαρμογές χωρίς να θυσιάζουμε την εμπειρία του χρήστη. Επιπλέον, ανάλογα με το τι απαιτήσεις υπάρχουν για το σύστημα, μπορεί να είναι εύκολη ή δύσκολη η ανάπτυξη του. Αυτό συμβαίνει γιατί για πολλές προσπάθειες καλύψεις κενών ασφάλειας απαιτούνται μεγάλες αλλαγές στο σύστημα.

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] T. a. B. M. a. K. A. Acar, «Single Password Authentication,» *Computer Networks,* June 2013.

[2] JetBrains, «Kotlin Programming - The State of Developer Ecosystem in 2020 Infographic,» JetBrains, June 2020. [Ηλεκτρονικό]. Available: https://www.jetbrains.com/lp/devecosystem-2020/kotlin/. [Πρόσβαση 17 February 2026].

[3] A. Parecki, P. De Ryck και D. Waite, «OAuth 2.0 for Browser-Based Applications,» Internet Engineering Task Force (IETF), 2025.

[4] O. Foundation, «Session Management Cheat Sheet,» OWASP Foundation, 2026.

[5] MITRE, «CWE-1004: Sensitive Cookie Without 'HttpOnly' Flag,» MITRE, 2026.

[6] Microsoft, «What is FIDO2,» Microsoft, 2024. [Ηλεκτρονικό]. Available: https://www.microsoft.com/en-us/security/business/security-101/what-is-fido2. [Πρόσβαση February 2026].

[7] M. (. W. Docs), «Web Authentication API,» Mozilla, [Ηλεκτρονικό]. Available: https://developer.mozilla.org/en-US/docs/Web/API/Web_Authentication_API. [Πρόσβαση February 2026].

[8] O. Foundation, «Cross Site Scripting (XSS),» OWASP Foundation, [Ηλεκτρονικό]. Available: https://owasp.org/www-community/attacks/xss/. [Πρόσβαση February 2026].

[9] R. (. Elizarov, Σκηνοθέτης, *Deep Dive into Coroutines on JVM.* [Φιλμ]. JetBrains, 2017.

[10] GeoNames, «GeoNames Gazetteer Download (Dump Files),» GeoNames, [Ηλεκτρονικό]. Available: https://download.geonames.org/export/dump/. [Πρόσβαση February 2026].

[11] «esbuild,» [Ηλεκτρονικό]. Available: https://esbuild.github.io/. [Πρόσβαση February 2026].

[12] «Password Hashing Competition,» 2015. [Ηλεκτρονικό]. Available: https://www.password-hashing.net/. [Πρόσβαση February 2026].

[13] D. K. J. Biryukov, «RFC 9106: Argon2 Memory-Hard Function for Password Hashing and Proof-of-Work Applications,» IETF, 2021. [Ηλεκτρονικό]. Available: https://www.rfc-editor.org/rfc/rfc9106. [Πρόσβαση February 2026].

ΠΑΡΑΡΤΗΜΑ Α : Οδηγός εγκατάσταση και ρύθμισης

Το παράτημα αυτό αποτελεί ένα πλήρη οδηγό για την πρακτική εγκατάσταση, παραμετροποίηση και εκτέλεση όλων των επιμέρους τμημάτων της εφαρμογής TravelTales. Ο οδηγός απευθύνεται σε όποιον θέλει να εκτελέσει την εφαρμογή είτε σε τοπικό περιβάλλον, είτε σε περιβάλλον παραγωγής.

Πριν ξεκινήσει κάποιος την διαδικασία εγκατάστασης πρέπει να έχει τα παρακάτω εργαλεία:

- JDK 17 ή νεότερο
- Node.js (v18.x / v20.x LTS) και npm
- Docker και docker compose (μόνο αν θέλει να εκτελέσει την βάση ως container)

Ο χρήστης μπορεί να δημιουργήσει το container με τη βάση δεδομένων με το παρακάτω αρχείο:

```
version: '3.8'

services:
  # Ορισμός της υπηρεσίας της βάσης δεδομένων
  postgres_db:
    image: postgres:16-alpine # Χρήση της ελαφριάς έκδοσης alpine
    container_name: traveltales_postgres
    restart: always
    environment:
      POSTGRES_DB: traveltales # Όνομα της βάσης δεδομένων
      POSTGRES_USER: travel_admin # Όνομα διαχειριστή βάσης
      POSTGRES_PASSWORD: SecurePassword123! # Ασφαλής κωδικός πρόσβασης
    ports:
      - "5432:5432" # Αντιστοίχιση της θύρας 5432 στο host machine
    volumes:
      - pgdata:/var/lib/postgresql/data # Μόνιμη αποθήκευση δεδομένων (volume)

volumes:
  pgdata: # Δήλωση του volume για τη διατήρηση των δεδομένων μετά την
επανεκκίνηση
```

Η εκκίνηση της βάσης δεδομένων μπορεί να γίνει με την εντολή `docker compose up -d`.

Για την εκκίνηση του backend ο χρήστης θα πρέπει να ορίσει τις παρακάτω μεταβλητές περιβάλλοντος:

```
# Ρυθμίσεις Βάσης Δεδομένων
DB_URL=jdbc:postgresql://localhost:5432/traveltales
DB_USER=travel_admin
DB_PASSWORD=SecurePassword123!

# Ρυθμίσεις Ασφάλειας JWT
JWT_SECRET=super_secret_signing_key_for_jwt_tokens_2026
JWT_ISSUER=http://localhost:8080
JWT_AUDIENCE=http://localhost:8080

# Ρυθμίσεις Google OAuth (εάν χρησιμοποιείται)
```

```
GOOGLE_CLIENT_ID=your_google_client_id.apps.googleusercontent.com
GOOGLE_CLIENT_SECRET=your_google_client_secret
```

Η εκτέλεση του backend μπορεί να γίνει με την εντολή `gradlew run`.

Τέλος, για την ιστοσελίδα θα πρέπει να γίνει εγκατάσταση των απαραίτητων node modules με τη χρήση της εντολής `npm install`, να οριστεί το base url και τέλος να γίνει η εκτέλεση του διακομιστή με την εντολή `npm run dev`.

ΠΑΡΑΡΤΗΜΑ Β : Κώδικας backend (μόνο ο router)

```
package gr.traveltales.modules

import gr.traveltales.data.db.dao.suspendTransaction
import gr.traveltales.data.db.tables.WebAuthnCredentialsTable
import gr.traveltales.data.models.*
import gr.traveltales.data.repositories.*
import gr.traveltales.data.service.*
import io.ktor.client.*
import io.ktor.client.call.*
import io.ktor.client.engine.cio.*
import io.ktor.client.plugins.contentnegotiation.*
import io.ktor.client.request.*
import io.ktor.client.request.forms.*
import io.ktor.http.*
import io.ktor.http.content.*
import io.ktor.serialization.kotlinx.json.*
import io.ktor.server.application.*
import io.ktor.server.auth.*
import io.ktor.server.auth.jwt.*
import io.ktor.server.http.content.*
import io.ktor.server.request.*
import io.ktor.server.response.*
import io.ktor.server.routing.*
import io.ktor.server.sessions.*
import io.ktor.util.cio.*
import io.ktor.util.collections.*
import io.ktor.utils.io.*
import java.util.*
import kotlinx.serialization.json.Json
import org.jetbrains.exposed.sql.selectAll

fun Application.configureRouting(redirects: ConcurrentMap<String, String>, httpClient: HttpClient) {
    val jwtData =
        JwtData(
            environment.config.property("jwt.audience").getString(),
            environment.config.property("jwt.issuer").getString(),
```

```
        environment.config.property("jwt.secret").getString()
    )
val userRepository = DatabaseUserRepository()
val userService = UserService(userRepository, jwtData)
val guideService = GuideService(DatabaseGuideRepository())
val notificationRepository = DatabaseNotificationRepository()
val notificationService = NotificationService(notificationRepository)
val followRepository = DatabaseFollowRepository()
val interactionRepository = DatabaseInteractionRepository()
val commentRepository = DatabaseCommentRepository()
val storyService = StoryService(DatabaseStoryRepository(), userRepository, followRepository, notificationService)
val categoryService = CategoryService()
val locationService = LocationService()

suspend fun Story.withStats(): Story {
    val favCount = interactionRepository.getFavoriteCount(UUID.fromString(this.id), "story")
    val svCount = interactionRepository.getSavedCount(UUID.fromString(this.id), "story")
    val commCount = commentRepository.getCommentCount(UUID.fromString(this.id), "story")
    return this.copy(favoriteCount = favCount, savedCount = svCount, commentCount = commCount)
}

suspend fun Guide.withStats(): Guide {
    val favCount = interactionRepository.getFavoriteCount(UUID.fromString(this.id), "guide")
    val svCount = interactionRepository.getSavedCount(UUID.fromString(this.id), "guide")
    val commCount = commentRepository.getCommentCount(UUID.fromString(this.id), "guide")
    return this.copy(favoriteCount = favCount, savedCount = svCount, commentCount = commCount)
}

routing {
    staticFiles("/uploads", java.io.File("uploads"))
    configureWebAuthnRoutes(userService)

    authenticate("jwt-auth") {
        post("/upload") {
            val multipart = call.receiveMultipart()
            var fileUrl = ""
            multipart.forEachPart { part ->
                if (part is PartData.FileItem) {
```

```
                val originalName = part.originalFileName ?: "image.png"
                val ext = java.io.File(originalName).extension.ifEmpty { "png" }
                val fileName = "${UUID.randomUUID()}.$ext"
                val file = java.io.File("uploads/$fileName")
                file.parentFile.mkdirs()
                part.provider().copyAndClose(file.writeChannel())
                fileUrl = "/uploads/$fileName"
            }
            part.dispose()
        }
        if (fileUrl.isNotEmpty()) {
            call.respond(mapOf("url" to fileUrl))
        } else {
            call.respond(HttpStatusCode.BadRequest, "No file uploaded")
        }
    }
}

route("/signup") {
    route("/local") {
        post {
            try {
                val request = call.receive<SignupRequest>()

                if (!userService.isEmailAvailable(request.email)) {
                    call.respond(
                        HttpStatusCode.Conflict,
                        "User already exists (Email Taken)"
                    )
                    return@post
                }

                val baseUsername = request.username ?: request.email.substringBefore("@")
                var finalUsername = baseUsername
                var counter = 1
                while (userService.userExists(finalUsername)) {
                    finalUsername = "$baseUsername${counter++}"
                }
```

```
            val now = System.currentTimeMillis()
            val user =
                User(
                    username = finalUsername,
                    password = request.password,
                    email = request.email,
                    isLocalUser = true,
                    picture = null,
                    createdAt = now,
                    updatedAt = now
                )

            if (userService.createUser(user)) {
                call.respond(HttpStatusCode.OK)
                return@post
            }

            call.respond(
                HttpStatusCode.BadRequest,
                mapOf("error" to "Invalid credentials")
            )
        } catch (e: Exception) {
            call.respond(
                HttpStatusCode.BadRequest,
                mapOf("error" to e.stackTraceToString())
            )
        }
    }
  }
}

route("/login") {
  route("/local") {
    post {
      try {
        val request = call.receive<LoginRequest>()
        val authenticatedUser = userService.authenticateByEmail(request)
```

```
if (authenticatedUser != null) {
    if (authenticatedUser.isTwoFactorEnabled) {
        val challenge = UUID.randomUUID().toString()
        val session =
            call.sessions.get<UserSession>()
                ?: UserSession("", "")
        call.sessions.set(session.copy(webAuthnChallenge = challenge))

        val credentials = suspendTransaction {
            WebAuthnCredentialsTable.selectAll()
                .where {
                    WebAuthnCredentialsTable.userId eq
                        UUID.fromString(
                            authenticatedUser.id
                        )
                }
                .map {
                    WebAuthnAllowCredential(
                        "public-key",
                        it[WebAuthnCredentialsTable.credentialId]
                    )
                }
        }

        call.respond(
            LoginResponse(
                message = "2FA required",
                requires2FA = true,
                authResponse =
                    WebAuthnLoginStartResponse(
                        challenge =
                            Base64
                                .getUrlEncoder()
                                .withoutPadding()
                                .encodeToString(
                                    challenge
                                        .toByteArray()
```

```
                    ),
                timeout = 60000,
                rpId = "localhost",
                allowCredentials = credentials
            )
        )
    )
    return@post
}

// TODO maybe make this better?
val accessToken = userService.generateAccessToken(authenticatedUser)
call.response.cookies.append(
    "auth_token",
    accessToken,
    httpOnly = true,
    path = "/"
)
if (request.rememberMe) {
    val refreshToken =
        userService.generateRefreshToken(authenticatedUser)
    call.response.cookies.append(
        "refresh_token",
        refreshToken,
        httpOnly = true,
        path = "/"
    )
} else {
    call.response.cookies.append(
        "refresh_token",
        "",
        httpOnly = true,
        path = "/",
        maxAge = 0
    )
}
call.respond(mapOf("token" to accessToken))
return@post
```

```
            }

            call.respond(
                HttpStatusCode.Unauthorized,
                mapOf("error" to "Invalid credentials")
            )
        } catch (e: Exception) {
            call.application.environment.log.error("Login error", e)
            call.respond(
                HttpStatusCode.BadRequest,
                mapOf("error" to (e.message ?: "Invalid request payload"))
            )
        }
    }
}

route("/google") {
    get {
        val clientId =
            System.getenv("GOOGLE_CLIENT_ID")
                ?: "972396710911-0fomrmbco1q0b14q4qqfakoai2er8bf9.apps.googleusercontent.com"
        val redirectUri = "http://localhost:8080/login/google/callback"
        val state = UUID.randomUUID().toString()

        call.sessions.set(UserSession(state, ""))

        val url =
            URLBuilder("https://accounts.google.com/o/oauth2/v2/auth")
                .apply {
                    parameters.append("client_id", clientId)
                    parameters.append("redirect_uri", redirectUri)
                    parameters.append("response_type", "code")
                    parameters.append("scope", "openid email profile")
                    parameters.append("state", state)
                    parameters.append("access_type", "offline")
                    parameters.append("prompt", "consent")
                }
                .buildString()
```

```
    call.respondRedirect(url)
}

route("/callback") {
    get {
        val code = call.request.queryParameters["code"]
        val state = call.request.queryParameters["state"]
        val session = call.sessions.get<UserSession>()

        if (code == null || state == null || session?.state != state) {
            call.respond(HttpStatusCode.BadRequest, "Invalid OAuth state")
            return@get
        }

        call.sessions.clear<UserSession>()

        val tokenResponse = exchangeCodeForToken(code)

        val idToken = tokenResponse.id_token
        val googleUser = verifyIdToken(idToken)

        val email = googleUser.email
        val name = googleUser.name ?: email.substringBefore("@")

        // Identify or create user
        val user =
            userService.getOrCreateGoogleUser(email, name, googleUser.picture)

        if (user == null) {
            // TODO make a better solution.
            call.respond(HttpStatusCode.BadRequest, "Google User not found")
        } else {
            val accessToken = userService.generateAccessToken(user)
            val refreshToken = userService.generateRefreshToken(user)
            call.response.cookies.append(
                "auth_token",
                accessToken,
```

```
                    httpOnly = true,
                    path = "/"
                )
                call.response.cookies.append(
                    "refresh_token",
                    refreshToken,
                    httpOnly = true,
                    path = "/"
                )
                call.respondRedirect("http://localhost:5173/auth/success")
            }
        }
    }
}

post("/logout") {
    call.response.cookies.append("auth_token", "", httpOnly = true, path = "/", maxAge = 0)
    call.response.cookies.append(
        "refresh_token",
        "",
        httpOnly = true,
        path = "/",
        maxAge = 0
    )
    call.respond(HttpStatusCode.OK)
}

post("/auth/refresh") {
    val refreshToken = call.request.cookies["refresh_token"]
    if (refreshToken == null) {
        call.respond(HttpStatusCode.Unauthorized, "No refresh token")
        return@post
    }

    try {
        val jwtSecret = application.environment.config.property("jwt.secret").getString()
        val jwtVerifier =
```

```
            com.auth0
                .jwt
                .JWT
                .require(com.auth0.jwt.algorithms.Algorithm.HMAC256(jwtSecret))
                // TODO Check if this claim is really needed in the refresh token for security reasons.
                //.withClaim("type", "refresh")
                .build()

            val decodedJWT = jwtVerifier.verify(refreshToken)
            val username = decodedJWT.getClaim("username").asString()

            val user = userService.getUser(username)
            if (user != null) {
                val newAccessToken = userService.generateAccessToken(user)
                call.response.cookies.append(
                    "auth_token",
                    newAccessToken,
                    httpOnly = true,
                    path = "/"
                )
                call.respond(HttpStatusCode.OK)
            } else {
                call.respond(HttpStatusCode.Unauthorized, "User not found")
            }
        } catch (e: Exception) {
            call.respond(HttpStatusCode.Unauthorized, "Invalid refresh token")
        }
    }

    route("/content") { route("/home") {} }

    route("/locations") {
        route("/countries") {
            get {
                try {
                    val countries = locationService.getAllCountries()
                    call.respond(countries)
                } catch (e: Exception) {
```

```
            call.respond(HttpStatusCode.InternalServerError, "Error fetching countries")
        }
    }
    get("/{code}/cities") {
        val code = call.parameters["code"]
        if (code == null) {
            call.respond(HttpStatusCode.BadRequest, "Missing country code")
            return@get
        }
        try {
            val cities = locationService.getCitiesByCountryCode(code)
            call.respond(cities)
        } catch (e: Exception) {
            call.respond(HttpStatusCode.InternalServerError, "Error fetching cities")
        }
    }
  }
}

route("/categories") {
  get {
    try {
      val categories = categoryService.getAllCategories()
      call.respond(categories)
    } catch (e: Exception) {
      call.respond(HttpStatusCode.InternalServerError, "Error fetching categories")
    }
  }

  authenticate("jwt-auth") {
    post {
      val principal = call.principal<JWTPrincipal>()
      val username = principal?.payload?.getClaim("username")?.asString()
      if (username == null) {
        call.respond(HttpStatusCode.Unauthorized, "Invalid token")
        return@post
      }
      val user = userService.getUser(username)
```

```
    if (user?.role != UserRole.ADMIN) {
        call.respond(HttpStatusCode.Forbidden, "Requires Admin role")
        return@post
    }
    try {
        val request =
            call.receive<CreateCategoryRequest>()
        val newCategory = categoryService.createCategory(request.name, request.code)
        if (newCategory != null) {
            call.respond(HttpStatusCode.Created, newCategory)
        } else {
            call.respond(HttpStatusCode.Conflict, "Category already exists")
        }
    } catch (e: Exception) {
        call.respond(HttpStatusCode.BadRequest, "Invalid request payload")
    }
}

delete("/{code}") {
    val principal = call.principal<JWTPrincipal>()
    val username = principal?.payload?.getClaim("username")?.asString()
    if (username == null) {
        call.respond(HttpStatusCode.Unauthorized, "Invalid token")
        return@delete
    }
    val user = userService.getUser(username)
    if (user?.role != UserRole.ADMIN) {
        call.respond(HttpStatusCode.Forbidden, "Requires Admin role")
        return@delete
    }
    val code =
        call.parameters["code"]
            ?: return@delete call.respond(HttpStatusCode.BadRequest)
    val success = categoryService.deleteCategory(code)
    if (success) {
        call.respond(HttpStatusCode.OK)
    } else {
        call.respond(HttpStatusCode.NotFound)
```

```
        }
      }
    }
}

route("/stories") {
    get {
        try {
            val page = call.request.queryParameters["page"]?.toIntOrNull()
            val pageSize = call.request.queryParameters["pageSize"]?.toIntOrNull()
            val search = call.request.queryParameters["search"]
            val categories =
                call.request.queryParameters["categories"]?.split(",")?.filter {
                    it.isNotEmpty()
                }

            val sort = call.request.queryParameters["sort"]
            val cityId = call.request.queryParameters["cityId"]
            val countryId = call.request.queryParameters["countryId"]

            val paginatedResponse =
                storyService.getStories(
                    page,
                    pageSize,
                    publishedOnly = true,
                    query = search,
                    categories = categories,
                    sort = sort,
                    cityId = cityId,
                    countryId = countryId
                )
            val itemsSummaries =
                paginatedResponse.items.map { s -> s.withStats().toSummary() }

            call.respond(
                PaginatedResponse(
                    items = itemsSummaries,
                    totalCount = paginatedResponse.totalCount,
```

```
                page = paginatedResponse.page,
                pageSize = paginatedResponse.pageSize,
                totalPages = paginatedResponse.totalPages
            )
        )
    } catch (e: Exception) {
        println("Error fetching stories: ${e.message}")
        e.printStackTrace()
        call.respond(
            HttpStatusCode.InternalServerError,
            mapOf("error" to (e.message ?: "Error fetching stories"))
        )
    }
}

get("/category") {
    try {
        val page = call.request.queryParameters["page"]?.toIntOrNull()
        val pageSize = call.request.queryParameters["pageSize"]?.toIntOrNull()
        val categories =
            call.request.queryParameters["categories"]?.split(",")?.filter {
                it.isNotEmpty()
            }

        val paginatedResponse =
            storyService.getStories(
                page,
                pageSize,
                publishedOnly = true,
                categories = categories
            )
        val itemsSummaries =
            paginatedResponse.items.map { s -> s.withStats().toSummary() }

        call.respond(
            PaginatedResponse(
                items = itemsSummaries,
                totalCount = paginatedResponse.totalCount,
```

```
                page = paginatedResponse.page,
                pageSize = paginatedResponse.pageSize,
                totalPages = paginatedResponse.totalPages
            )
        )
    } catch (e: Exception) {
        call.respond(
            HttpStatusCode.InternalServerError,
            mapOf("error" to (e.message ?: "Error fetching stories by category"))
        )
    }
}

authenticate("jwt-auth") {
    post {
        try {
            val request = call.receive<CreateStoryRequest>()
            val principal = call.principal<JWTPrincipal>()
            val username = principal?.payload?.getClaim("username")?.asString()

            if (username == null) {
                call.respond(HttpStatusCode.Unauthorized, "Invalid token")
                return@post
            }

            val user = userService.getUser(username)
            if (user == null ||
                user.role == UserRole.UNVERIFIED
            ) {
                call.respond(
                    HttpStatusCode.Forbidden,
                    "You must be verified to create a story."
                )
                return@post
            }

            val now = System.currentTimeMillis()
            val newStory =
```

```
            Story(
                id = UUID.randomUUID().toString(),
                title = request.title,
                previewContent = request.previewContent,
                fullContent = request.fullContent,
                authorUsername = username,
                categories = request.categories,
                createdAt = now,
                updatedAt = now,
                published = request.published,
                images = request.images,
                cityId = request.cityId,
                countryId = request.countryId
            )
        val success = storyService.createStory(newStory)
        if (success) {
            call.respond(HttpStatusCode.Created, newStory.withStats())
        } else {
            call.respond(
                HttpStatusCode.InternalServerError,
                "Failed to create story"
            )
        }
    } catch (e: Exception) {
        call.respond(HttpStatusCode.BadRequest, "Invalid request payload")
    }
}

put("/{id}") {
    try {
        val id =
            call.parameters["id"]
                ?: return@put call.respond(HttpStatusCode.BadRequest)
        val request = call.receive<CreateStoryRequest>()
        val principal = call.principal<JWTPrincipal>()
        val username = principal?.payload?.getClaim("username")?.asString()

        if (username == null) {
```

```
    call.respond(HttpStatusCode.Unauthorized, "Invalid token")
    return@put
}

val existingStory = storyService.getStoryById(id)
if (existingStory == null) {
    call.respond(HttpStatusCode.NotFound, "Story not found")
    return@put
}

if (existingStory.authorUsername != username) {
    call.respond(
        HttpStatusCode.Forbidden,
        "You can only edit your own stories"
    )
    return@put
}

val updatedStory =
    existingStory.copy(
        title = request.title,
        previewContent = request.previewContent,
        fullContent = request.fullContent,
        categories = request.categories,
        published = request.published,
        images = request.images,
        cityId = request.cityId,
        countryId = request.countryId
    )
val success = storyService.updateStory(updatedStory)
if (success) {
    call.respond(HttpStatusCode.OK, updatedStory.withStats())
} else {
    call.respond(
        HttpStatusCode.InternalServerError,
        "Failed to update story"
    )
}
```

```
    } catch (e: Exception) {
        call.respond(HttpStatusCode.BadRequest, "Invalid request payload")
    }
}

delete("/{id}") {
    val id =
        call.parameters["id"]
            ?: return@delete call.respond(HttpStatusCode.BadRequest)
    val principal = call.principal<JWTPrincipal>()
    val username = principal?.payload?.getClaim("username")?.asString()

    if (username == null) {
        call.respond(HttpStatusCode.Unauthorized, "Invalid token")
        return@delete
    }

    val existingStory = storyService.getStoryById(id)
    if (existingStory == null) {
        call.respond(HttpStatusCode.NotFound, "Story not found")
        return@delete
    }

    if (existingStory.authorUsername != username) {
        call.respond(
            HttpStatusCode.Forbidden,
            "You can only delete your own stories"
        )
        return@delete
    }

    val success = storyService.deleteStory(id)
    if (success) {
        call.respond(HttpStatusCode.OK, "Story deleted")
    } else {
        call.respond(HttpStatusCode.InternalServerError, "Failed to delete story")
    }
}
```

```
    }

    authenticate("jwt-auth", optional = true) {
        get("/{id}") {
            val id = call.parameters["id"]
            if (id == null) {
                call.respond(HttpStatusCode.BadRequest, "Missing or invalid id")
                return@get
            }
            try {
                val story = storyService.getStoryById(id)
                val principal = call.principal<JWTPrincipal>()
                val requesterUsername = principal?.payload?.getClaim("username")?.asString()

                if (story == null) {
                    call.respond(HttpStatusCode.NotFound, "Story not found")
                } else {
                    val storyWithFavs: Story = story.withStats()
                    call.respond(storyWithFavs)
                }
            } catch (e: Exception) {
                call.respond(HttpStatusCode.InternalServerError, "Error fetching story")
            }
        }
    }
}

route("/guides") {
    get {
        try {
            val page = call.request.queryParameters["page"]?.toIntOrNull()
            val pageSize = call.request.queryParameters["pageSize"]?.toIntOrNull()
            val search = call.request.queryParameters["search"]

            val paginatedResponse =
                guideService.getGuides(
                    page,
                    pageSize,
```

```
                publishedOnly = true,
                query = search
            )
            val itemsSummaries =
                paginatedResponse.items.map { g -> g.withStats().toSummary() }

            call.respond(
                PaginatedResponse(
                    items = itemsSummaries,
                    totalCount = paginatedResponse.totalCount,
                    page = paginatedResponse.page,
                    pageSize = paginatedResponse.pageSize,
                    totalPages = paginatedResponse.totalPages
                )
            )
        } catch (e: Exception) {
            call.respond(HttpStatusCode.InternalServerError, "Error fetching guides")
        }
    }

    authenticate("jwt-auth") {
        post {
            try {
                val request = call.receive<CreateGuideRequest>()
                val principal = call.principal<JWTPrincipal>()
                val username = principal?.payload?.getClaim("username")?.asString()

                if (username == null) {
                    call.respond(HttpStatusCode.Unauthorized, "Invalid token")
                    return@post
                }

                val user = userService.getUser(username)
                if (user == null ||
                    user.role == UserRole.UNVERIFIED
                ) {
                    call.respond(
                        HttpStatusCode.Forbidden,
```

```
                "You must be verified to create a guide."
            )
            return@post
        }

        val now = System.currentTimeMillis()
        val newGuide =
            Guide(
                id = UUID.randomUUID().toString(),
                title = request.title,
                previewContent = request.previewContent,
                fullContent = request.fullContent,
                authorUsername = username,
                createdAt = now,
                updatedAt = now,
                published = request.published,
                images = request.images
            )
        val success = guideService.createGuide(newGuide)
        if (success) {
            call.respond(HttpStatusCode.Created, newGuide.withStats())
        } else {
            call.respond(
                HttpStatusCode.InternalServerError,
                "Failed to create guide"
            )
        }
    } catch (e: Exception) {
        call.respond(HttpStatusCode.BadRequest, "Invalid request payload")
    }
}

put("/{id}") {
    try {
        val id =
            call.parameters["id"]
                ?: return@put call.respond(HttpStatusCode.BadRequest)
        val request = call.receive<CreateGuideRequest>()
```

```
val principal = call.principal<JWTPrincipal>()
val username = principal?.payload?.getClaim("username")?.asString()

if (username == null) {
    call.respond(HttpStatusCode.Unauthorized, "Invalid token")
    return@put
}

val existingGuide = guideService.getGuideById(id)
if (existingGuide == null) {
    call.respond(HttpStatusCode.NotFound, "Guide not found")
    return@put
}

if (existingGuide.authorUsername != username) {
    call.respond(
        HttpStatusCode.Forbidden,
        "You can only edit your own guides"
    )
    return@put
}

val updatedGuide =
    existingGuide.copy(
        title = request.title,
        previewContent = request.previewContent,
        fullContent = request.fullContent,
        published = request.published,
        images = request.images
    )
val success = guideService.updateGuide(updatedGuide)
if (success) {
    call.respond(HttpStatusCode.OK, updatedGuide.withStats())
} else {
    call.respond(
        HttpStatusCode.InternalServerError,
        "Failed to update guide"
    )
```

```
        }
    } catch (e: Exception) {
        call.respond(HttpStatusCode.BadRequest, "Invalid request payload")
    }
}

delete("/{id}") {
    val id =
        call.parameters["id"]
            ?: return@delete call.respond(HttpStatusCode.BadRequest)
    val principal = call.principal<JWTPrincipal>()
    val username = principal?.payload?.getClaim("username")?.asString()

    if (username == null) {
        call.respond(HttpStatusCode.Unauthorized, "Invalid token")
        return@delete
    }

    val existingGuide = guideService.getGuideById(id)
    if (existingGuide == null) {
        call.respond(HttpStatusCode.NotFound, "Guide not found")
        return@delete
    }

    if (existingGuide.authorUsername != username) {
        call.respond(
            HttpStatusCode.Forbidden,
            "You can only delete your own guides"
        )
        return@delete
    }

    val success = guideService.deleteGuide(id)
    if (success) {
        call.respond(HttpStatusCode.OK, "Guide deleted")
    } else {
        call.respond(HttpStatusCode.InternalServerError, "Failed to delete guide")
    }
```

```
        }
    }

    authenticate("jwt-auth", optional = true) {
        get("/{id}") {
            val id = call.parameters["id"]
            if (id == null) {
                call.respond(HttpStatusCode.BadRequest, "Missing or invalid id")
                return@get
            }
            try {
                val guide = guideService.getGuideById(id)
                val principal = call.principal<JWTPrincipal>()
                val requesterUsername = principal?.payload?.getClaim("username")?.asString()

                if (guide == null) {
                    call.respond(HttpStatusCode.NotFound, "Guide not found")
                } else {
                    val guideWithFavs: gr.traveltales.data.models.Guide = guide.withStats()
                    call.respond(guideWithFavs)
                }
            } catch (e: Exception) {
                call.respond(HttpStatusCode.InternalServerError, "Error fetching guide")
            }
        }
    }
}

route("/user") {
    authenticate("jwt-auth") {
        get("/me") {
            val principal = call.principal<JWTPrincipal>()
            val username = principal?.payload?.getClaim("username")?.asString()
            if (username != null) {
                val user = userService.getUser(username)
                if (user != null) {
                    val userIdUuid = UUID.fromString(user.id)
                    val followersCount = followRepository.getFollowersCount(userIdUuid)
```

```
            val followingCount = followRepository.getFollowingCount(userIdUuid)
            val followerUsernames = followRepository.getFollowers(userIdUuid).map { it.username }
            val followingUsernames = followRepository.getFollowing(userIdUuid).map { it.username }
            val likedIds =
                interactionRepository.getFavoriteItems(userIdUuid).map { it.first.toString() }
            val savedIds = interactionRepository.getSavedItems(userIdUuid).map { it.first.toString() }
            val readingListIds =
                interactionRepository.getReadingListItems(userIdUuid).map { it.first.toString() }

            call.respond(
                UserProfileDTO(
                    username = user.username,
                    picture = user.picture,
                    bannerPicture = user.bannerPicture,
                    role = user.role,
                    firstName = user.firstName,
                    lastName = user.lastName,
                    email = user.email,
                    isLocalUser = user.isLocalUser,
                    isTwoFactorEnabled = user.isTwoFactorEnabled,
                    bio = user.bio,
                    createdAt = user.createdAt,
                    updatedAt = user.updatedAt,
                    followersCount = followersCount,
                    followingCount = followingCount,
                    followerUsernames = followerUsernames,
                    followingUsernames = followingUsernames,
                    likedIds = likedIds,
                    savedIds = savedIds,
                    readingListIds = readingListIds
                )
            )
        } else {
            call.respond(HttpStatusCode.NotFound)
        }
    } else {
        call.respond(HttpStatusCode.Unauthorized)
    }
```

```
    }
}

authenticate("jwt-auth", optional = true) {
    get("/{username}/follow-info") {
        val username =
            call.parameters["username"]
                ?: return@get call.respond(HttpStatusCode.BadRequest)
        val targetUser =
            userService.getUser(username)
                ?: return@get call.respond(HttpStatusCode.NotFound)

        val followersCount =
            followRepository.getFollowersCount(UUID.fromString(targetUser.id))
        val followingCount =
            followRepository.getFollowingCount(UUID.fromString(targetUser.id))

        val principal = call.principal<JWTPrincipal>()
        val currentUsername = principal?.payload?.getClaim("username")?.asString()
        val isFollowing =
            if (currentUsername != null) {
                val currentUser = userService.getUser(currentUsername)
                if (currentUser != null) {
                    followRepository.isFollowing(
                        UUID.fromString(currentUser.id),
                        UUID.fromString(targetUser.id)
                    )
                } else false
            } else false

        call.respond(FollowInfo(followersCount, followingCount, isFollowing))
    }
}

authenticate("jwt-auth") {
    delete("/me") {
        val principal = call.principal<JWTPrincipal>()
        val username = principal?.payload?.getClaim("username")?.asString()
```

```
    if (username != null) {
        val success = userService.deleteUser(username)
        if (success) {
            call.response.cookies.append(
                "auth_token",
                "",
                httpOnly = true,
                path = "/",
                maxAge = 0
            )
            call.response.cookies.append(
                "refresh_token",
                "",
                httpOnly = true,
                path = "/",
                maxAge = 0
            )
            call.respond(HttpStatusCode.OK)
        } else {
            call.respond(
                HttpStatusCode.InternalServerError,
                "Failed to delete account"
            )
        }
    } else {
        call.respond(HttpStatusCode.Unauthorized)
    }
}

get("/settings") {
    val principal = call.principal<JWTPrincipal>()
    val username = principal?.payload?.getClaim("username")?.asString()

    if (username != null) {
        val settings = userService.getUserSettings(username)
        call.respond(settings)
    } else {
        call.respond(HttpStatusCode.Unauthorized, "Invalid token")
```

```
    }
}

post("/theme") {
    val request = call.receive<gr.traveltales.data.models.ThemeUpdate>()
    val principal = call.principal<JWTPrincipal>()
    val username = principal?.payload?.getClaim("username")?.asString()

    if (username != null) {
        userService.updateUserTheme(username, request.themePreference)
        call.respond(HttpStatusCode.OK)
    } else {
        call.respond(HttpStatusCode.Unauthorized, "Invalid token")
    }
}

post("/profile") {
    val request = call.receive<UserProfileUpdate>()
    val principal = call.principal<JWTPrincipal>()
    val username = principal?.payload?.getClaim("username")?.asString()

    if (username != null) {
        userService.updateUserProfile(
            username,
            request.firstName,
            request.lastName,
            request.bio
        )
        call.respond(HttpStatusCode.OK)
    } else {
        call.respond(HttpStatusCode.Unauthorized, "Invalid token")
    }
}
post("/banner") {
    val principal = call.principal<JWTPrincipal>()
    val username = principal?.payload?.getClaim("username")?.asString()
    if (username == null) {
        call.respond(HttpStatusCode.Unauthorized)
```

```
            return@post
        }

        val multipart = call.receiveMultipart()
        var fileName = ""

        multipart.forEachPart { part ->
            if (part is PartData.FileItem) {
                try {
                    val originalName = part.originalFileName ?: "banner.png"
                    val ext = java.io.File(originalName).extension.ifEmpty { "png" }
                    val uuid = UUID.randomUUID().toString()
                    fileName = "banner_$uuid.$ext"
                    val file = java.io.File("uploads/$fileName")
                    file.parentFile.mkdirs()
                    part.provider().copyAndClose(file.writeChannel())
                } catch (e: Exception) {
                    e.printStackTrace()
                }
            }
            part.dispose()
        }

        if (fileName.isNotEmpty()) {
            val bannerUrl = "/uploads/$fileName"
            userService.updateUserBannerPicture(username, bannerUrl)
            call.respond(HttpStatusCode.OK, mapOf("bannerPicture" to bannerUrl))
        } else {
            call.respond(HttpStatusCode.BadRequest, "No file uploaded")
        }
    }

    post("/picture") {
        val principal = call.principal<JWTPrincipal>()
        val username = principal?.payload?.getClaim("username")?.asString()
        if (username == null) {
            call.respond(HttpStatusCode.Unauthorized)
            return@post
```

```
}

val multipart = call.receiveMultipart()
var fileName = ""

multipart.forEachPart { part ->
    if (part is PartData.FileItem) {
        try {
            val originalName = part.originalFileName ?: "image.png"
            val ext = java.io.File(originalName).extension.ifEmpty { "png" }
            val uuid = java.util.UUID.randomUUID().toString()
            fileName = "$uuid.$ext"
            val file = java.io.File("uploads/$fileName")
            file.parentFile.mkdirs()
            part.provider().copyAndClose(file.writeChannel())
        } catch (e: Exception) {
            e.printStackTrace()
        }
    }
    part.dispose()
}

if (fileName.isNotEmpty()) {
    val picUrl = "/uploads/$fileName"
    userService.updateUserPicture(username, picUrl)
    val updatedUser = userService.getUser(username)
    if (updatedUser != null) {
        val accessToken = userService.generateAccessToken(updatedUser)
        val refreshToken = userService.generateRefreshToken(updatedUser)
        call.response.cookies.append(
            "auth_token",
            accessToken,
            httpOnly = true,
            path = "/"
        )
        call.response.cookies.append(
            "refresh_token",
            refreshToken,
```

```
                httpOnly = true,
                path = "/"
            )
            call.respond(mapOf("token" to accessToken))
            return@post
        }
    }
    call.respond(HttpStatusCode.BadRequest, mapOf("error" to "No file uploaded"))
}

get("/notifications") {
    val principal = call.principal<JWTPrincipal>()
    val username = principal?.payload?.getClaim("username")?.asString()
    if (username == null) {
        call.respond(HttpStatusCode.Unauthorized)
        return@get
    }
    val user = userService.getUser(username)
    if (user != null) {
        val notifications = notificationService.getNotifications(user.id)
        call.respond(notifications)
    } else {
        call.respond(HttpStatusCode.Unauthorized)
    }
}

post("/notifications/{id}/read") {
    val principal = call.principal<JWTPrincipal>()
    val username = principal?.payload?.getClaim("username")?.asString()
    if (username == null) {
        call.respond(HttpStatusCode.Unauthorized)
        return@post
    }
    val user = userService.getUser(username)
    val id = call.parameters["id"]
    if (id != null && user != null) {
        val success = notificationService.markAsRead(id, user.id)
        if (success) {
```

```
            call.respond(HttpStatusCode.OK)
        } else {
            call.respond(HttpStatusCode.NotFound)
        }
    } else {
        call.respond(HttpStatusCode.BadRequest)
    }
}

delete("/notifications/{id}") {
    val principal = call.principal<JWTPrincipal>()
    val username = principal?.payload?.getClaim("username")?.asString()
    if (username == null) {
        call.respond(HttpStatusCode.Unauthorized)
        return@delete
    }
    val user = userService.getUser(username)
    val id = call.parameters["id"]
    if (id != null && user != null) {
        val success = notificationService.deleteNotification(id, user.id)
        if (success) {
            call.respond(HttpStatusCode.OK)
        } else {
            call.respond(HttpStatusCode.NotFound)
        }
    } else {
        call.respond(HttpStatusCode.BadRequest)
    }
}

route("/interactions") {
    post("/toggle") {
        val principal = call.principal<JWTPrincipal>()
        val username = principal?.payload?.getClaim("username")?.asString()
        if (username == null) {
            call.respond(HttpStatusCode.Unauthorized)
            return@post
        }
```

```
try {
    val user =
        userService.getUser(username)
            ?: return@post call.respond(HttpStatusCode.Unauthorized)
    val request =
        call.receive<
                InteractionToggleRequest>()

    val itemId = UUID.fromString(request.itemId)

    val success =
        when (request.interactionType) {
            "favorite" ->
                interactionRepository.toggleFavorite(
                    UUID.fromString(user.id),
                    itemId,
                    request.itemType
                )

            "saved" ->
                interactionRepository.toggleSaved(
                    UUID.fromString(user.id),
                    itemId,
                    request.itemType
                )

            "reading_list" ->
                interactionRepository.toggleReadingList(
                    UUID.fromString(user.id),
                    itemId,
                    request.itemType
                )

            else -> false
        }
    call.respond(mapOf("status" to success))
} catch (e: Exception) {
    call.respond(
```

```
            HttpStatusCode.InternalServerError,
            mapOf("error" to (e.message ?: "Unknown error"))
        )
    }
}

post("/mark-as-read") {
    val principal = call.principal<JWTPrincipal>()
    val username = principal?.payload?.getClaim("username")?.asString()
    if (username == null) {
        call.respond(HttpStatusCode.Unauthorized)
        return@post
    }
    try {
        val user =
            userService.getUser(username)
                ?: return@post call.respond(HttpStatusCode.Unauthorized)
        val request =
            call.receive<
                    gr.traveltales.data.models.InteractionToggleRequest>()
        val itemId = UUID.fromString(request.itemId)
        val success =
            interactionRepository.removeFromReadingList(
                UUID.fromString(user.id),
                itemId,
                request.itemType
            )
        call.respond(mapOf("status" to success))
    } catch (e: Exception) {
        call.respond(
            HttpStatusCode.InternalServerError,
            mapOf("error" to (e.message ?: "Unknown error"))
        )
    }
}

get("/status/{itemId}") {
    val principal = call.principal<JWTPrincipal>()
```

```
    val username = principal?.payload?.getClaim("username")?.asString()
    if (username == null) {
        call.respond(HttpStatusCode.Unauthorized)
        return@get
    }
    val user =
        userService.getUser(username)
            ?: return@get call.respond(HttpStatusCode.Unauthorized)
    val itemId =
        call.parameters["itemId"]
            ?: return@get call.respond(HttpStatusCode.BadRequest)
    val itemType = call.request.queryParameters["type"] ?: "story"

    val status =
        interactionRepository.getInteractionStatus(
            UUID.fromString(user.id),
            UUID.fromString(itemId),
            itemType
        )
    call.respond(status)
}

route("/favorites") {
    get("/stories") {
        val principal = call.principal<JWTPrincipal>()
        val username = principal?.payload?.getClaim("username")?.asString()
        if (username == null) {
            call.respond(HttpStatusCode.Unauthorized)
            return@get
        }
        val user =
            userService.getUser(username)
                ?: return@get call.respond(HttpStatusCode.Unauthorized)
        val favorites =
            interactionRepository.getFavoriteItems(UUID.fromString(user.id))
        val stories: List<StorySummary> =
            favorites.filter { it.second == "story" }.mapNotNull {
                storyService
```

```
                .getStoryById(it.first.toString())
                ?.withStats()
                ?.toSummary()
            }
        call.respond(stories)
    }

    get("/guides") {
        val principal = call.principal<JWTPrincipal>()
        val username = principal?.payload?.getClaim("username")?.asString()
        if (username == null) {
            call.respond(HttpStatusCode.Unauthorized)
            return@get
        }
        val user =
            userService.getUser(username)
                ?: return@get call.respond(HttpStatusCode.Unauthorized)
        val favorites =
            interactionRepository.getFavoriteItems(UUID.fromString(user.id))
        val guides: List<GuideSummary> =
            favorites.filter { it.second == "guide" }.mapNotNull {
                guideService
                    .getGuideById(it.first.toString())
                    ?.withStats()
                    ?.toSummary()
            }
        call.respond(guides)
    }
}

route("/saved") {
    get("/stories") {
        val principal = call.principal<JWTPrincipal>()
        val username = principal?.payload?.getClaim("username")?.asString()
        if (username == null) {
            call.respond(HttpStatusCode.Unauthorized)
            return@get
        }
```

```
    val user =
        userService.getUser(username)
            ?: return@get call.respond(HttpStatusCode.Unauthorized)
    val saved =
        interactionRepository.getSavedItems(UUID.fromString(user.id))
    println(
        "DEBUG: Found ${saved.size} saved items for user ${user.username}"
    )
    saved.forEach {
        println(
            "DEBUG: Raw saved item: ID=${it.first}, Type='${it.second}'"
        )
    }

    val stories: List<StorySummary> =
        saved
            .filter { it.second.trim().lowercase() == "story" }
            .mapNotNull {
                val s =
                    storyService.getStoryById(
                        it.first.toString()
                    )
                if (s == null) {
                    println(
                        "DEBUG: Could not find story for ID ${it.first}"
                    )
                } else {
                    println(
                        "DEBUG: Found story ${s.title} for ID ${it.first}"
                    )
                }
                s?.withStats()?.toSummary()
            }
    println("DEBUG: Returning ${stories.size} saved stories")
    call.respond(stories)
}

get("/guides") {
```

```
        val principal = call.principal<JWTPrincipal>()
        val username = principal?.payload?.getClaim("username")?.asString()
        if (username == null) {
            call.respond(HttpStatusCode.Unauthorized)
            return@get
        }
        val user =
            userService.getUser(username)
                ?: return@get call.respond(HttpStatusCode.Unauthorized)
        val saved =
            interactionRepository.getSavedItems(UUID.fromString(user.id))
        val guides: List<GuideSummary> =
            saved.filter { it.second == "guide" }.mapNotNull {
                guideService
                    .getGuideById(it.first.toString())
                    ?.withStats()
                    ?.toSummary()
            }
        call.respond(guides)
    }
}

route("/reading-list") {
    get("/stories") {
        val principal = call.principal<JWTPrincipal>()
        val username = principal?.payload?.getClaim("username")?.asString()
        if (username == null) {
            call.respond(HttpStatusCode.Unauthorized)
            return@get
        }
        val user =
            userService.getUser(username)
                ?: return@get call.respond(HttpStatusCode.Unauthorized)
        val readingList =
            interactionRepository.getReadingListItems(
                UUID.fromString(user.id)
            )
        val stories: List<StorySummary> =
```

```
                readingList.filter { it.second == "story" }.mapNotNull {
                    storyService
                        .getStoryById(it.first.toString())
                        ?.withStats()
                        ?.toSummary()
                }
            call.respond(stories)
        }

        get("/guides") {
            val principal = call.principal<JWTPrincipal>()
            val username = principal?.payload?.getClaim("username")?.asString()
            if (username == null) {
                call.respond(HttpStatusCode.Unauthorized)
                return@get
            }
            val user =
                userService.getUser(username)
                    ?: return@get call.respond(HttpStatusCode.Unauthorized)
            val readingList =
                interactionRepository.getReadingListItems(
                    UUID.fromString(user.id)
                )
            val guides: List<GuideSummary> =
                readingList.filter { it.second == "guide" }.mapNotNull {
                    guideService
                        .getGuideById(it.first.toString())
                        ?.withStats()
                        ?.toSummary()
                }
            call.respond(guides)
        }
    }
  }
}

route("/comments") {
```

```
get("/{itemType}/{itemId}") {
    try {
        val itemType =
            call.parameters["itemType"]
                ?: return@get call.respond(HttpStatusCode.BadRequest)
        val itemId =
            call.parameters["itemId"]
                ?: return@get call.respond(HttpStatusCode.BadRequest)
        val comments =
            commentRepository.getCommentsForItem(UUID.fromString(itemId), itemType)
        call.respond(comments)
    } catch (e: Exception) {
        call.respond(HttpStatusCode.InternalServerError, "Error fetching comments")
    }
}

authenticate("jwt-auth") {
    post {
        try {
            val request = call.receive<CreateCommentRequest>()
            val principal = call.principal<JWTPrincipal>()
            val username = principal?.payload?.getClaim("username")?.asString()

            if (username == null) {
                call.respond(HttpStatusCode.Unauthorized, "Invalid token")
                return@post
            }

            val user =
                userService.getUser(username)
                    ?: return@post call.respond(HttpStatusCode.Unauthorized)

            val now = System.currentTimeMillis()
            val comment =
                Comment(
                    id = UUID.randomUUID().toString(),
                    userId = user.id.toString(),
                    username = user.username,
```

```
                itemId = request.itemId,
                itemType = request.itemType,
                content = request.content,
                userPicture = user.picture,
                createdAt = now,
                updatedAt = now
            )

        val success = commentRepository.addComment(comment)
        if (success) {
            call.respond(HttpStatusCode.Created, comment)
        } else {
            call.respond(
                HttpStatusCode.InternalServerError,
                "Failed to add comment"
            )
        }
    } catch (e: Exception) {
        call.respond(HttpStatusCode.BadRequest, "Invalid request payload")
    }
}

put("/{id}") {
    try {
        val commentId =
            call.parameters["id"]
                ?: return@put call.respond(HttpStatusCode.BadRequest)
        val request = call.receive<UpdateCommentRequest>()
        val principal = call.principal<JWTPrincipal>()
        val username = principal?.payload?.getClaim("username")?.asString()

        if (username == null) {
            call.respond(HttpStatusCode.Unauthorized, "Invalid token")
            return@put
        }

        val user =
            userService.getUser(username)
```

```
            ?: return@put call.respond(HttpStatusCode.Unauthorized)

        val success =
            commentRepository.updateComment(
                UUID.fromString(commentId),
                UUID.fromString(user.id),
                request.content
            )
        if (success) {
            call.respond(HttpStatusCode.OK)
        } else {
            call.respond(
                HttpStatusCode.InternalServerError,
                "Failed to update comment"
            )
        }
    } catch (e: Exception) {
        call.respond(HttpStatusCode.BadRequest, "Invalid request payload")
    }
}

delete("/{id}") {
    try {
        val commentId =
            call.parameters["id"]
                ?: return@delete call.respond(HttpStatusCode.BadRequest)
        val principal = call.principal<JWTPrincipal>()
        val username = principal?.payload?.getClaim("username")?.asString()

        if (username == null) {
            call.respond(HttpStatusCode.Unauthorized, "Invalid token")
            return@delete
        }

        val user =
            userService.getUser(username)
                ?: return@delete call.respond(HttpStatusCode.Unauthorized)
```

```
            val success =
                commentRepository.removeComment(
                    UUID.fromString(commentId),
                    UUID.fromString(user.id)
                )
            if (success) {
                call.respond(HttpStatusCode.OK, "Comment deleted")
            } else {
                call.respond(HttpStatusCode.Forbidden, "Could not delete comment")
            }
        } catch (e: Exception) {
            call.respond(HttpStatusCode.InternalServerError, "Error deleting comment")
        }
    }
  }
}

route("/user") {
  get("/search") {
    val page = call.request.queryParameters["page"]?.toIntOrNull()
    val pageSize = call.request.queryParameters["pageSize"]?.toIntOrNull()
    val search = call.request.queryParameters["search"]

    val paginatedUsers = userService.getUsers(page, pageSize, search)

    val usersDTO = paginatedUsers.items.map {
      UserSearchDTO(
        username = it.username,
        picture = it.picture,
        firstName = it.firstName,
        lastName = it.lastName,
        bio = it.bio
      )
    }

    call.respond(
      PaginatedResponse(
        items = usersDTO,
```

```
            totalCount = paginatedUsers.totalCount,
            page = paginatedUsers.page,
            pageSize = paginatedUsers.pageSize,
            totalPages = paginatedUsers.totalPages
        )
    )
}

get("/{username}") {
    val username =
        call.parameters["username"]
            ?: return@get call.respond(HttpStatusCode.BadRequest)
    val user = userService.getUser(username)
    if (user != null) {
        val followersCount = followRepository.getFollowersCount(UUID.fromString(user.id))
        val followingCount = followRepository.getFollowingCount(UUID.fromString(user.id))

        call.respond(
            UserProfileResponse(
                username = user.username,
                picture = user.picture,
                bannerPicture = user.bannerPicture,
                firstName = user.firstName,
                lastName = user.lastName,
                bio = user.bio,
                followersCount = followersCount,
                followingCount = followingCount
            )
        )
    } else {
        call.respond(HttpStatusCode.NotFound)
    }
}

get("/{username}/followers") {
    val username =
        call.parameters["username"]
            ?: return@get call.respond(HttpStatusCode.BadRequest)
```

```
    val targetUser =
        userService.getUser(username)
            ?: return@get call.respond(HttpStatusCode.NotFound)
    val followers = followRepository.getFollowers(UUID.fromString(targetUser.id))
    call.respond(
        followers.map {
            FollowerDTO(it.username, it.picture, it.firstName, it.lastName)
        }
    )
}

get("/{username}/following") {
    val username =
        call.parameters["username"]
            ?: return@get call.respond(HttpStatusCode.BadRequest)
    val targetUser =
        userService.getUser(username)
            ?: return@get call.respond(HttpStatusCode.NotFound)
    val following = followRepository.getFollowing(UUID.fromString(targetUser.id))
    call.respond(
        following.map {
            FollowerDTO(it.username, it.picture, it.firstName, it.lastName)
        }
    )
}

authenticate("jwt-auth", optional = true) {
    get("/{username}/stories") {
        val username =
            call.parameters["username"]
                ?: return@get call.respond(HttpStatusCode.BadRequest)
        val principal = call.principal<JWTPrincipal>()
        val requesterUsername = principal?.payload?.getClaim("username")?.asString()

        val publishedParam =
            call.request.queryParameters["published"]?.toBooleanStrictOrNull()

        // Security: only owner can see their own drafts
```

```
    val finalPublished =
        if (username == requesterUsername) {
            publishedParam
        } else {
            // If requester is NOT the owner, they can only see published
            // stories
            if (publishedParam == false)
                return@get call.respond(
                    HttpStatusCode.Forbidden,
                    "Cannot view drafts of another user"
                )
            true
        }

    val page = call.request.queryParameters["page"]?.toIntOrNull()
    val pageSize = call.request.queryParameters["pageSize"]?.toIntOrNull()

    val paginatedResponse =
        storyService.getStories(
            page = page,
            pageSize = pageSize,
            publishedOnly = false,
            authorUsername = username,
            published = finalPublished
        )
    val stories = paginatedResponse.items.map { it.withStats().toSummary() }
    call.respond(stories)
}

get("/{username}/guides") {
    val username =
        call.parameters["username"]
            ?: return@get call.respond(HttpStatusCode.BadRequest)
    val principal = call.principal<JWTPrincipal>()
    val requesterUsername = principal?.payload?.getClaim("username")?.asString()

    val publishedParam =
        call.request.queryParameters["published"]?.toBooleanStrictOrNull()
```

```
        // Security: only owner can see their own drafts
        val finalPublished =
            if (username == requesterUsername) {
                publishedParam
            } else {
                // If requester is NOT the owner, they can only see published guides
                if (publishedParam == false)
                    return@get call.respond(
                        HttpStatusCode.Forbidden,
                        "Cannot view drafts of another user"
                    )
                true
            }

        val page = call.request.queryParameters["page"]?.toIntOrNull()
        val pageSize = call.request.queryParameters["pageSize"]?.toIntOrNull()

        val paginatedResponse =
            guideService.getGuides(
                page = page,
                pageSize = pageSize,
                publishedOnly = false,
                authorUsername = username,
                published = finalPublished
            )
        val guides = paginatedResponse.items.map { it.withStats().toSummary() }
        call.respond(guides)
    }
}

authenticate("jwt-auth") {
    post("/follow/{targetUsername}") {
        val targetUsername =
            call.parameters["targetUsername"]
                ?: return@post call.respond(HttpStatusCode.BadRequest)
        val principal = call.principal<JWTPrincipal>()
        val currentUsername =
```

```
            principal?.payload?.getClaim("username")?.asString()
                ?: return@post call.respond(HttpStatusCode.Unauthorized)

        if (currentUsername == targetUsername) {
            call.respond(HttpStatusCode.BadRequest, "You cannot follow yourself")
            return@post
        }

        val currentUser =
            userService.getUser(currentUsername)
                ?: return@post call.respond(HttpStatusCode.Unauthorized)
        val targetUser =
            userService.getUser(targetUsername)
                ?: return@post call.respond(HttpStatusCode.NotFound)

        val success =
            followRepository.followUser(
                UUID.fromString(currentUser.id),
                UUID.fromString(targetUser.id)
            )
        if (success) {
            call.respond(HttpStatusCode.OK, "Followed user")
        } else {
            call.respond(HttpStatusCode.InternalServerError, "Failed to follow user")
        }
    }

    post("/unfollow/{targetUsername}") {
        val targetUsername =
            call.parameters["targetUsername"]
                ?: return@post call.respond(HttpStatusCode.BadRequest)
        val principal = call.principal<JWTPrincipal>()
        val currentUsername =
            principal?.payload?.getClaim("username")?.asString()
                ?: return@post call.respond(HttpStatusCode.Unauthorized)

        val currentUser =
            userService.getUser(currentUsername)
```

```
                ?: return@post call.respond(HttpStatusCode.Unauthorized)
            val targetUser =
                userService.getUser(targetUsername)
                ?: return@post call.respond(HttpStatusCode.NotFound)

            val success =
                followRepository.unfollowUser(
                    UUID.fromString(currentUser.id),
                    UUID.fromString(targetUser.id)
                )
            if (success) {
                call.respond(HttpStatusCode.OK, "Unfollowed user")
            } else {
                call.respond(HttpStatusCode.InternalServerError, "Failed to unfollow user")
            }
        }
    }
}

authenticate("jwt-auth") {
    route("/admin") {
        get("/users") {
            val principal = call.principal<JWTPrincipal>()
            val username =
                principal?.payload?.getClaim("username")?.asString()
                ?: return@get call.respond(HttpStatusCode.Unauthorized)
            val user =
                userService.getUser(username)
                ?: return@get call.respond(HttpStatusCode.Unauthorized)

            if (user.role == UserRole.ADMIN) {
                val page = call.request.queryParameters["page"]?.toIntOrNull()
                val pageSize = call.request.queryParameters["pageSize"]?.toIntOrNull()
                val search = call.request.queryParameters["search"]

                val paginatedUsers:
                    PaginatedResponse<
                        User> =
```

```
            userService.getUsers(page, pageSize, search)
        val usersDTO: List<UserAdminDTO> =
            paginatedUsers.items.map {
                UserAdminDTO(
                    id = it.id,
                    username = it.username,
                    email = it.email,
                    isLocalUser = it.isLocalUser ?: false,
                    picture = it.picture,
                    bannerPicture = it.bannerPicture,
                    role = it.role,
                    firstName = it.firstName,
                    lastName = it.lastName,
                    isTwoFactorEnabled = it.isTwoFactorEnabled,
                    bio = it.bio,
                    createdAt = it.createdAt,
                    updatedAt = it.updatedAt
                )
            }
        call.respond(
            PaginatedResponse(
                items = usersDTO,
                totalCount = paginatedUsers.totalCount,
                page = paginatedUsers.page,
                pageSize = paginatedUsers.pageSize,
                totalPages = paginatedUsers.totalPages
            )
        )
    } else {
        call.respond(HttpStatusCode.Forbidden, "Requires Admin or Moderator role")
    }
}

post("/users/{targetUsername}/role") {
    val principal = call.principal<JWTPrincipal>()
    val adminUsername =
        principal?.payload?.getClaim("username")?.asString()
            ?: return@post call.respond(HttpStatusCode.Unauthorized)
```

```
    val targetUsername =
        call.parameters["targetUsername"]
            ?: return@post call.respond(HttpStatusCode.BadRequest)
    val newRoleStr =
        call.receive<Map<String, String>>()["role"]
            ?: return@post call.respond(HttpStatusCode.BadRequest)

    try {
        val newRole = UserRole.valueOf(newRoleStr)
        val success =
            userService.updateUserRole(adminUsername, targetUsername, newRole)
        if (success) {
            call.respond(HttpStatusCode.OK)
        } else {
            call.respond(
                HttpStatusCode.Forbidden,
                "You do not have permission to change this user's role"
            )
        }
    } catch (e: IllegalArgumentException) {
        call.respond(HttpStatusCode.BadRequest, "Invalid role")
    }
}

delete("/users/{targetUsername}") {
    val principal = call.principal<JWTPrincipal>()
    val adminUsername =
        principal?.payload?.getClaim("username")?.asString()
            ?: return@delete call.respond(HttpStatusCode.Unauthorized)
    val targetUsername =
        call.parameters["targetUsername"]
            ?: return@delete call.respond(HttpStatusCode.BadRequest)

    val success = userService.adminDeleteUser(adminUsername, targetUsername)
    if (success) {
        call.respond(HttpStatusCode.OK, "User deleted successfully")
    } else {
        call.respond(
```

```
                    HttpStatusCode.Forbidden,
                    "You do not have permission to delete this user"
                )
            }
        }
    }
}

suspend fun exchangeCodeForToken(code: String): GoogleTokenResponse {
    val client =
        HttpClient(CIO) {
            install(ContentNegotiation) { json(Json { ignoreUnknownKeys = true }) }
        }

    return client
        .post("https://oauth2.googleapis.com/token") {
            setBody(
                FormDataContent(
                    Parameters.build {
                        append("code", code)
                        append(
                            "client_id",
                            System.getenv("GOOGLE_CLIENT_ID")
                                ?: "972396710911-0fomrmbco1q0b14q4qqfakoai2er8bf9.apps.googleusercontent.com"
                        )
                        append(
                            "client_secret",
                            System.getenv("GOOGLE_CLIENT_SECRET")
                                ?: "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"
                        )
                        append(
                            "redirect_uri",
                            "http://localhost:8080/login/google/callback"
                        )
                        append("grant_type", "authorization_code")
                    }
```

```
            )
        )
    }
    .body()
}

suspend fun verifyIdToken(idToken: String): GoogleTokenInfo {
    val client =
        HttpClient(CIO) {
            install(ContentNegotiation) { json(Json { ignoreUnknownKeys = true }) }
        }

    return client
        .get("https://oauth2.googleapis.com/tokeninfo") { parameter("id_token", idToken) }
        .body()
}
```

ΠΑΡΑΡΤΗΜΑ C : Κώδικας frontend

Home SPA

```
<template>
  <div class="card main-card stories-main-card no-scrollbar">
    <!-- Hero Section -->
    <section
      class="relative h-[450px] flex-shrink-0 overflow-hidden rounded-2xl mb-12 shadow-2xl group border border-surface-200 dark:border-surface-700">
      <img src="https://images.unsplash.com/photo-1476514525535-07fb3b4ae5f1?q=80&w=2070&auto=format&fit=crop"
        class="absolute inset-0 w-full h-full object-cover transition-transform duration-1000 group-hover:scale-105" />
      <div
        class="absolute inset-0 bg-gradient-to-r from-surface-900/90 via-surface-900/40 to-transparent flex flex-col justify-center p-12 text-white">
        <div class="max-w-2xl">
          <span
            class="inline-block px-4 py-1 rounded-full bg-primary text-white text-sm font-bold mb-4 animate-bounce">NEW
            ADVENTURES</span>
          <h1 class="text-5xl md:text-6xl font-black mb-6 leading-tight">Every Journey <br /><span
              class="text-primary">Has a Story.</span></h1>
          <p class="text-lg md:text-xl text-surface-200 mb-8 max-w-lg leading-relaxed">
            Discover hidden gems, share your path, and connect with a world of travelers who turn miles
            into
            memories.
          </p>
          <div class="flex gap-4">
            <Button label="Explore Stories" icon="pi pi-compass" size="large" rounded class="px-8"
@click="$router.push('/stories')" />
            <Button label="Write Your Own" severity="secondary" outlined size="large" rounded
              class="px-8 bg-surface-0/10 backdrop-blur-md" @click="$router.push('/stories/create')" />
          </div>
        </div>
      </div>
      <!-- Decorative Overlay -->
      <div class="absolute bottom-0 right-0 p-8 z-10 hidden lg:block opacity-30">
        <span class="text-8xl font-black tracking-tighter text-white">DISCOVER</span>
      </div>
```

```
</section>

<!-- Stories Carousels -->
<div class="pb-12">
  <!-- Latest Stories -->
  <section v-if="latestStories.length">
    <div class="flex justify-between items-center mb-4 px-2 ">
      <div class="flex items-center gap-3 ">
        <div class="w-1 h-8 bg-primary rounded-full"></div>
        <h2 class="text-2xl font-bold text-surface-900 dark:text-surface-0 tracking-tight">Latest
          Adventures</h2>
      </div>
      <Button label="See More" icon="pi pi-arrow-right" iconPos="right" variant="text" size="small"
        class="p-0 text-primary hover:underline" @click="$router.push('/stories')" />
    </div>
    <Carousel :value="latestStories" :numVisible="3" :numScroll="1" :responsiveOptions="responsiveOptions"
      circular :autoplayInterval="4000" class="premium-carousel">
      <template #item="slotProps">
        <div class="px-3 h-full">
          <StoryCard :story="slotProps.data" />
        </div>
      </template>
    </Carousel>
  </section>

  <!-- Most Popular Stories -->
  <section v-if="popularStories.length">
    <div class="flex justify-between items-center mb-4 px-2">
      <div class="flex items-center gap-3">
        <div class="w-1 h-8 bg-orange-500 rounded-full"></div>
        <h2 class="text-2xl font-bold text-surface-900 dark:text-surface-0 tracking-tight">Most Popular
        </h2>
      </div>
      <Button label="Explore Top" icon="pi pi-arrow-right" iconPos="right" variant="text" size="small"
        class="p-0 text-orange-500 hover:underline" @click="$router.push('/stories?sort=popular')" />
    </div>
    <Carousel :value="popularStories" :numVisible="3" :numScroll="1" :responsiveOptions="responsiveOptions"
      circular class="premium-carousel">
```

```
            <template #item="slotProps">
              <div class="px-3 h-full">
                <StoryCard :story="slotProps.data" />
              </div>
            </template>
          </Carousel>
        </section>

        <!-- Trending Stories -->
        <section v-if="trendingStories.length">
          <div class="flex justify-between items-center mb-4 px-2">
            <div class="flex items-center gap-3">
              <div class="w-1 h-8 bg-emerald-500 rounded-full"></div>
              <h2 class="text-2xl font-bold text-surface-900 dark:text-surface-0 tracking-tight">Trending Now
              </h2>
            </div>
            <Button label="Join the Trend" icon="pi pi-arrow-right" iconPos="right" variant="text" size="small"
              class="p-0 text-emerald-500 hover:underline" @click="$router.push('/stories?sort=trending')" />
          </div>
          <Carousel :value="trendingStories" :numVisible="3" :numScroll="1" :responsiveOptions="responsiveOptions"
            circular class="premium-carousel">
            <template #item="slotProps">
              <div class="px-3 h-full">
                <StoryCard :story="slotProps.data" />
              </div>
            </template>
          </Carousel>
        </section>
      </div>

      <AppFooter />
    </div>
</template>

<script setup>
import { ref, onMounted } from "vue";
import Carousel from 'primevue/carousel';
import AppFooter from '@/layout/AppFooter.vue';
```

```
import StoryCard from '@/components/StoryCard.vue';
import { authenticatedFetch } from "@/service/AuthService";

const latestStories = ref([]);
const popularStories = ref([]);
const trendingStories = ref([]);

const responsiveOptions = ref([
    { breakpoint: '1400px', numVisible: 3, numScroll: 1 },
    { breakpoint: '1199px', numVisible: 3, numScroll: 1 },
    { breakpoint: '767px', numVisible: 2, numScroll: 1 },
    { breakpoint: '575px', numVisible: 1, numScroll: 1 }
]);

async function fetchHomeContent() {
    try {
        // Fetch Latest
        const latestRes = await authenticatedFetch(window.API_BASE_URL + "/stories?page=1&pageSize=5&sort=latest");
        if (latestRes.ok) {
            const data = await latestRes.json();
            latestStories.value = data.items;
        }

        // Fetch Popular
        const popularRes = await authenticatedFetch(window.API_BASE_URL + "/stories?page=1&pageSize=5&sort=popular");
        if (popularRes.ok) {
            const data = await popularRes.json();
            popularStories.value = data.items;
        }

        // Fetch Trending
        const trendingRes = await authenticatedFetch(window.API_BASE_URL + "/stories?page=1&pageSize=5&sort=trending");
        if (trendingRes.ok) {
            const data = await trendingRes.json();
            trendingStories.value = data.items;
        }
    } catch (error) {
```

```
        console.error("Error fetching home content:", error);
    }
}

onMounted(() => {
    fetchHomeContent();
});
</script>

<style scoped>
:deep(.premium-carousel .p-carousel-content) {
    @apply gap-4;
}

:deep(.premium-carousel .p-carousel-container) {
    @apply py-4;
}

:deep(.premium-carousel .p-link) {
    @apply bg-surface-200 dark:bg-surface-700 w-8 h-8 rounded-full hover:bg-primary transition-colors duration-300;
}

:deep(.premium-carousel .p-carousel-indicator-list) {
    @apply mt-6;
}

:deep(.premium-carousel .p-carousel-indicator .p-link) {
    @apply w-2 h-2 rounded-full;
}

:deep(.premium-carousel .p-carousel-indicator.p-highlight .p-link) {
    @apply w-6 bg-primary transition-all duration-300;
}
</style>
```

Login SPA

```
<script setup>
import { ref } from 'vue';
import { useRouter } from 'vue-router';
import FloatingConfigurator from '@/components/FloatingConfigurator.vue';
import { AuthService } from '@/service/AuthService';
import { useAuth } from '@/composables/useAuth';

const router = useRouter();
const { processToken } = useAuth();

const isSignup = ref(false);
const email = ref(localStorage.getItem('rememberedEmail') || '');
const password = ref('');
const username = ref('');
const checked = ref(!!localStorage.getItem('rememberedEmail'));
const errorMsg = ref('');
const loading = ref(false);

const toggleMode = () => {
    isSignup.value = !isSignup.value;
    errorMsg.value = '';
};

const submitAuth = async () => {
    errorMsg.value = '';
    if (!email.value || !password.value || (isSignup.value && !username.value)) {
        errorMsg.value = 'Please fill out all fields.';
        return;
    }

    loading.value = true;
    try {
        if (isSignup.value) {
            await AuthService.signup(email.value, password.value, username.value);
            // Auto login after signup
            await AuthService.login(email.value, password.value, checked.value);
            processToken();
```

```
            if (checked.value) {
                localStorage.setItem('rememberedEmail', email.value);
            } else {
                localStorage.removeItem('rememberedEmail');
            }
            router.push('/');
        } else {
            const result = await AuthService.login(email.value, password.value, checked.value);
            if (result && result.requires2FA) {
                handle2FAChallenge(result.authResponse);
            } else if (result) {
                processToken();
                if (checked.value) {
                    localStorage.setItem('rememberedEmail', email.value);
                } else {
                    localStorage.removeItem('rememberedEmail');
                }
                router.push('/');
            }
        }
    } catch (err) {
        errorMsg.value = err.message || "An error occurred.";
    } finally {
        loading.value = false;
    }
};

const handle2FAChallenge = async (authResponse) => {
    try {
        errorMsg.value = 'Please insert and touch your hardware key.';

        const getOptions = {
            publicKey: {
                challenge: base64ToBuffer(authResponse.challenge),
                timeout: authResponse.timeout,
                rpId: authResponse.rpId,
                allowCredentials: authResponse.allowCredentials.map(c => ({
                    type: 'public-key',
```

```
                id: base64ToBuffer(c.id)
            })),
            userVerification: 'preferred'
        }
      };

      const assertion = await navigator.credentials.get(getOptions);

      const loginData = {
        email: email.value,
        id: assertion.id,
        rawId: bufferToBase64(assertion.rawId),
        type: assertion.type,
        response: {
          authenticatorData: bufferToBase64(assertion.response.authenticatorData),
          clientDataJSON: bufferToBase64(assertion.response.clientDataJSON),
          signature: bufferToBase64(assertion.response.signature),
          userHandle: assertion.response.userHandle ? bufferToBase64(assertion.response.userHandle) : null
        }
      };

      await AuthService.finish2FALogin(loginData, checked.value);
      processToken();
      if (checked.value) {
        localStorage.setItem('rememberedEmail', email.value);
      } else {
        localStorage.removeItem('rememberedEmail');
      }
      router.push('/');
    } catch (err) {
      console.error(err);
      errorMsg.value = '2FA failed: ' + (err.message || 'Verification failed');
    }
};

const bufferToBase64 = (buffer) => {
    let binary = '';
    const bytes = new Uint8Array(buffer);
```

```
    const len = bytes.byteLength;
    for (let i = 0; i < len; i++) {
        binary += String.fromCharCode(bytes[i]);
    }
    return window.btoa(binary).replace(/\+/g, '-').replace(/\//g, '_').replace(/=/g, '');
};

const base64ToBuffer = (base64) => {
    const binary = window.atob(base64.replace(/-/g, '+').replace(/_/g, '/'));
    const bytes = new Uint8Array(binary.length);
    for (let i = 0; i < binary.length; i++) {
        bytes[i] = binary.charCodeAt(i);
    }
    return bytes.buffer;
};

const redirectToGoogle = () => {
    window.location.href = window.API_BASE_URL + '/login/google'
}

const redirectToGithub = () => {
    console.log("Github login not implemented yet")
}
</script>

<template>
    <FloatingConfigurator />
    <div
        class="bg-surface-50 dark:bg-surface-950 flex items-center justify-center min-h-screen min-w-[100vw] overflow-hidden">
        <div class="flex flex-col items-center justify-center">
            <div
                style="border-radius: 56px; padding: 0.3rem; background: linear-gradient(180deg, var(--primary-color) 10%, rgba(33, 150, 243, 0) 30%)">
                <div class="w-full bg-surface-0 dark:bg-surface-900 py-20 px-8 sm:px-20" style="border-radius: 53px">
                    <div class="text-center mb-8">
                        <svg xmlns="http://www.w3.org/2000/svg" viewBox="0 0 512 512"
                            class="mb-8 w-16 shrink-0 mx-auto">
                            <g>
```

```
                    <path class="st0"
                        d="M286.105,88.898c-100.217,0.008-181.43,81.222-
181.438,181.438c0.008,100.225,81.222,181.438,181.438,181.446 c100.225-0.008,181.439-81.222,181.446-
181.446C467.544,170.119,386.33,88.906,286.105,88.898z M179.43,163.661 c10.19-10.181,21.832-18.91,34.562-25.85c-
6.736,10.316-12.612,21.932-17.518,34.646h-25.04 C174.01,169.444,176.627,166.463,179.43,163.661z
M153.741,197.944h34.714c-4.795,18.48-7.623,38.589-8.501,59.644h-44.06
C137.7,236.103,143.897,215.901,153.741,197.944z M153.757,342.736c-9.86-17.966-16.057-38.168-17.864-59.661h43.883
c0.877,21.106,3.849,41.164,8.661,59.661H153.757z M179.43,377.01c-2.802-2.802-5.419-5.774-7.986-8.788h25.04
c3.554,9.219,7.581,17.888,12.089,25.875c1.748,3.072,3.614,5.951,5.496,8.814C201.295,395.963,189.645,387.225,179.43,3
77.01z M273.366,420.555c-1.266-0.101-2.542-0.16-3.808-0.304c-4.153-1.706-8.282-3.96-12.393-7.024 c-12.333-9.16-
24.043-24.752-33.109-45.005h49.31V420.555z M273.366,342.736h-58.445c-5.26-18-8.518-38.184-9.481-59.661h67.926
V342.736z M273.366,257.588h-68.112c0.963-21.468,4.34-41.644,9.582-59.644h58.53V257.588z M273.366,172.458h-
49.243 c2.094-4.677,4.28-9.176,6.644-13.339c7.801-13.837,16.944-24.617,26.398-31.674c4.112-3.057,8.24-5.302,12.385-
7.008 c1.266-0.143,2.55-0.202,3.816-0.312V172.458z M418.478,197.944c9.844,17.957,16.04,38.159,17.846,59.644h-43.89
c-0.878-21.097-3.85-41.146-8.662-59.644H418.478z M392.788,163.661c2.803,2.802,5.42,5.783,7.995,8.797h-25.056 c-
3.546-9.219-7.582-17.889-12.098-25.884c-1.739-3.064-3.613-5.943-5.479-
8.814C370.906,144.716,382.565,153.454,392.788,163.661z
M298.844,120.125c1.266,0.11,2.549,0.169,3.816,0.312c4.136,1.706,8.274,3.951,12.384,7.008
c12.343,9.159,24.044,24.769,33.119,45.013h-49.319V120.125z
M298.844,197.944h58.453c5.252,17.999,8.51,38.184,9.472,59.644 h-67.925V197.944z M341.451,381.552c-7.809,13.837-
16.952,24.626-26.407,31.675c-4.11,3.064-8.248,5.318-12.393,7.024 c-1.266,0.144-2.541,0.202-3.807,0.304v-
52.333h49.252C345.993,372.89,343.816,377.399,341.451,381.552z M298.844,342.736v-59.661 h68.111c-0.962,21.477-
4.339,41.662-9.582,59.661H298.844z M392.788,377.01c-10.19,10.198-21.831,18.919-34.562,25.85 c6.728-10.307,12.604-
21.924,17.509-34.638h25.038C398.208,371.236,395.591,374.208,392.788,377.01z M418.46,342.736h-34.705 c4.803-
18.472,7.623-38.581,8.501-59.661h44.068C434.518,304.568,428.322,324.77,418.46,342.736z"
                        fill="var(--primary-color)" />
                    <path class="st0"
                        d="M401.652,459.001c-36.09,22.101-75.928,32.604-115.328,32.604c-74.477-0.008-147.146-37.567-
188.884-105.721 c-22.102-36.09-32.604-75.929-32.604-115.329c0.009-74.485,37.568-147.129,105.73-
188.884L159.92,64.288 C85.503,109.868,44.432,189.274,44.449,270.555c-0.009,43.022,11.506,86.626,35.608,125.974
C125.628,470.93,205.043,512.009,286.324,512c43.021,0,86.625-11.506,125.982-35.608l-10.654-17.382V459.001z"
                        fill="var(--primary-color)" />
                    <path class="st0"
                        d="M244.587,69.818h95.387c9.818,0,17.787-3.317,17.787-13.152c0-9.819-18.8-19.333-28.618-
19.333h-19.704 l-40.1-36.099c-2.009-2.22-9.97-0.768-12.384-0.759c-2.98,0.017-2.33,4.778-1.182,7.547l11.718,29.311h-
28.062l-15.39-24.448 c-2.702-2.946-6.932-3.942-10.654-2.499c-3.732,1.444-6.188,5.032-6.188,9.025l-0.253,17.922v5.293
C206.943,57.644,224.993,69.818,244.587,69.818z"
                        fill="var(--primary-color)" />
                </g>
            </svg>
            <div class="text-surface-900 dark:text-surface-0 text-3xl font-medium mb-4">Welcome to TravelTales!</div>
            <span class="text-muted-color font-medium">{{ isSignup ? 'Create an account to continue' : 'Sign in to
continue' }}</span>
        </div>

        <div v-if="errorMsg" class="mb-4 text-red-500 font-bold text-center">
            {{ errorMsg }}
        </div>
```

```
            <div>
                <label v-if="isSignup" for="username1" class="block text-surface-900 dark:text-surface-0 text-xl font-medium mb-2">Username</label>
                <InputText v-if="isSignup" id="username1" type="text" placeholder="Username" class="w-full md:w-[30rem] mb-4" v-model="username" />

                <label for="email1" class="block text-surface-900 dark:text-surface-0 text-xl font-medium mb-2">Email</label>
                <InputText id="email1" type="text" placeholder="Email address" class="w-full md:w-[30rem] mb-4" v-model="email" />

                <label for="password1" class="block text-surface-900 dark:text-surface-0 font-medium text-xl mb-2">Password</label>
                <Password id="password1" v-model="password" placeholder="Password" :toggleMask="true" class="mb-4" fluid :feedback="false" @keyup.enter="submitAuth"></Password>

                <div class="flex items-center justify-between mt-2 mb-8 gap-8">
                  <div class="flex items-center">
                    <Checkbox v-model="checked" id="rememberme1" binary class="mr-2"></Checkbox>
                    <label for="rememberme1">Remember me</label>
                  </div>
                  <span class="font-medium no-underline ml-2 text-right cursor-pointer text-primary" @click="toggleMode">
                    {{ isSignup ? 'Already have an account?' : 'Need an account? Sign Up' }}
                  </span>
                </div>

                <Button :label="isSignup ? 'Sign Up' : 'Sign In'" :disabled="loading" class="w-full" @click="submitAuth"></Button>

                <div class="flex flex-col items-center justify-between mt-4 mb-8 gap-4">
                  <Button @click="redirectToGoogle" label="Sign In with Google" icon="pi pi-google"
                    class="w-full p-button-outlined" />
                  <Button @click="redirectToGithub" label="Sign In with Github" icon="pi pi-github"
                    class="w-full p-button-outlined" />
                </div>
              </div>
            </div>
          </div>
```

```
    </div>
</template>

<style scoped>
.pi-eye {
    transform: scale(1.6);
    margin-right: 1rem;
}

.pi-eye-slash {
    transform: scale(1.6);
    margin-right: 1rem;
}
</style>
```

ΠΑΡΑΡΤΗΜΑ D Κώδικας android

MainViewModel

```
package com.traveltales

import androidx.lifecycle.viewModelScope
import com.traveltales.domain.auth.GetUserRoleUseCase
import com.traveltales.domain.auth.GetUsernameUseCase
import com.traveltales.domain.auth.loggedin.IsUserLoggedInUseCase
import com.traveltales.domain.auth.model.UserRole
import com.traveltales.domain.profile.view.GetMyProfileUseCase
import com.traveltales.domain.session.AuthSessionManager
import com.traveltales.domain.session.SessionEvent
import com.traveltales.domain.util.DataError
import com.traveltales.domain.util.Result
import com.traveltales.ui.core.BaseViewModel
import com.traveltales.ui.navigation.NavigationRoutes
import com.traveltales.ui.navigation.Navigator
import com.traveltales.util.infoLog
import kotlinx.coroutines.delay
import kotlinx.coroutines.flow.MutableStateFlow
import kotlinx.coroutines.flow.StateFlow
import kotlinx.coroutines.flow.asStateFlow
import kotlinx.coroutines.flow.distinctUntilChanged
import kotlinx.coroutines.flow.launchIn
import kotlinx.coroutines.flow.onEach
import kotlinx.coroutines.flow.update

internal class MainViewModel(
    private val authSessionManager: AuthSessionManager,
    private val navigator: Navigator,
    private val isUserLoggedInUseCase: IsUserLoggedInUseCase,
    private val getUsernameUseCase: GetUsernameUseCase,
    private val getUserRoleUseCase: GetUserRoleUseCase,
    private val getMyProfileUseCase: GetMyProfileUseCase
) : BaseViewModel() {

    private val _isLoggedIn = MutableStateFlow(false)
```

```
val isLoggedIn: StateFlow<Boolean> = _isLoggedIn.asStateFlow()

private val _isSessionChecked = MutableStateFlow(false)
val isSessionChecked: StateFlow<Boolean> = _isSessionChecked.asStateFlow()

private val _username = MutableStateFlow("")
val username: StateFlow<String> = _username.asStateFlow()

private val _userRole = MutableStateFlow<UserRole?>(null)
val userRole: StateFlow<UserRole?> = _userRole.asStateFlow()

init {
    observeSessionEvents()
    checkInitialSession()
    startPeriodicRefresh()
}

private fun startPeriodicRefresh() {
    launch {
        while (true) {
            delay(4 * 60 * 1000L)
            if (_isLoggedIn.value) {
                infoLog("Session", "Performing periodic session refresh...")
                getMyProfileUseCase()
            }
        }
    }
}

private fun checkInitialSession() {
    launch {
        if (isUserLoggedInUseCase()) {
            val result = getMyProfileUseCase()
            if (result is Result.Success) {
                val username = getUsernameUseCase() ?: ""
                val role = getUserRoleUseCase()
                _isLoggedIn.update { true }
                _username.update { username }
```

```
                _userRole.update { role }
            } else {
                val error = (result as? Result.Error)?.error
                if (error == DataError.Network.UNAUTHORIZED || error == DataError.Network.FORBIDDEN) {
                    infoLog("MainViewModel", "Session invalid (401/403). Logging out.")
                    _isLoggedIn.update { false }
                    _username.update { "" }
                    _userRole.update { null }
                    authSessionManager.emit(SessionEvent.Logout)
                } else {
                    infoLog("MainViewModel", "Initial profile fetch failed with error: $error. Keeping session.")
                }
            }
        } else {
            _isLoggedIn.update { false }
            _username.update { "" }
            _userRole.update { null }
            authSessionManager.emit(SessionEvent.Logout)
        }
        _isSessionChecked.update { true }
    }
}

private fun loadProfileAndSyncId() {
    launch {
        getMyProfileUseCase()
    }
}

private fun observeSessionEvents() {
    authSessionManager.events
        .distinctUntilChanged()
        .onEach { event ->
            when (event) {
                SessionEvent.Logout -> {
                    _isLoggedIn.update { false }
                    _username.update { "" }
                    _userRole.update { null }
```

```
                navigator.clearToRoot()
                navigator.replaceTop(NavigationRoutes.Lobby)
            }
            SessionEvent.Login -> {
                val username = getUsernameUseCase() ?: ""
                val role = getUserRoleUseCase()
                _isLoggedIn.update { true }
                _username.update { username }
                _userRole.update { role }
                loadProfileAndSyncId()
            }
        }
    }
    .launchIn(viewModelScope)
  }
}
```

Composable SignInScreen

```
package com.traveltales.ui.auth.signin.composable

import androidx.compose.foundation.Image
import androidx.compose.foundation.clickable
import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.Row
import androidx.compose.foundation.layout.Spacer
import androidx.compose.foundation.layout.WindowInsets
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.height
import androidx.compose.foundation.layout.ime
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.layout.size
import androidx.compose.foundation.rememberScrollState
import androidx.compose.foundation.text.KeyboardOptions
import androidx.compose.foundation.verticalScroll
import androidx.compose.material3.Button
```

```
import androidx.compose.material3.Checkbox
import androidx.compose.material3.ExperimentalMaterial3Api
import androidx.compose.material3.Icon
import androidx.compose.material3.IconButton
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.OutlinedTextField
import androidx.compose.material3.Text
import androidx.compose.material3.TextButton
import androidx.compose.runtime.Composable
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.text.input.ImeAction
import androidx.compose.ui.text.input.KeyboardType
import androidx.compose.ui.text.input.PasswordVisualTransformation
import androidx.compose.ui.text.input.VisualTransformation
import com.traveltales.generated.resources.Res
import com.traveltales.generated.resources.auth_dont_have_account
import com.traveltales.generated.resources.auth_email
import com.traveltales.generated.resources.auth_password
import com.traveltales.generated.resources.auth_remember_me
import com.traveltales.generated.resources.auth_sign_in
import com.traveltales.generated.resources.auth_sign_up
import com.traveltales.generated.resources.auth_signin_subtitle
import com.traveltales.generated.resources.auth_signin_title
import com.traveltales.generated.resources.cd_app_logo
import com.traveltales.generated.resources.cd_hide_password
import com.traveltales.generated.resources.cd_show_password
import com.traveltales.generated.resources.common_error
import com.traveltales.generated.resources.ic_close
import com.traveltales.generated.resources.ic_visibility
import com.traveltales.generated.resources.ic_visibility_off
import com.traveltales.generated.resources.logo
import com.traveltales.ui.auth.signin.model.SignInAction
import com.traveltales.ui.auth.signin.model.SignInUiState
import com.traveltales.ui.core.composable.BaseScreen
import com.traveltales.ui.core.composable.MessageDialog
import com.traveltales.ui.theme.Dimens
```

```
import org.jetbrains.compose.resources.painterResource
import org.jetbrains.compose.resources.stringResource

@OptIn(ExperimentalMaterial3Api::class)
@Composable
internal fun SignInScreen(
    state: SignInUiState,
    onAction: (SignInAction) -> Unit,
    onBackClick: () -> Unit,
    modifier: Modifier = Modifier
) {
    BaseScreen(
        uiState = state,
        modifier = modifier,
        contentWindowInsets = WindowInsets.ime
    ) {
        Box(modifier = Modifier.fillMaxSize()) {
            SignInScreenContent(
                state = state,
                onAction = onAction
            )

            IconButton(
                onClick = onBackClick,
                modifier = Modifier
                    .align(Alignment.TopEnd)
                    .padding(Dimens.Spacing.medium)
                    .size(Dimens.Icon.xxLarge)
            ) {
                Icon(
                    painter = painterResource(Res.drawable.ic_close),
                    contentDescription = null,
                    tint = MaterialTheme.colorScheme.onSurface
                )
            }
        }

        state.error?.let { error ->
```

```
            MessageDialog(
                title = stringResource(Res.string.common_error),
                message = error.asString(),
                onDismiss = { onAction(SignInAction.OnDismissError) },
                titleColor = MaterialTheme.colorScheme.error
            )
        }
    }
}

@Composable
private fun SignInScreenContent(
    state: SignInUiState,
    onAction: (SignInAction) -> Unit
) {
    Column(
        modifier = Modifier
            .fillMaxSize()
            .padding(Dimens.Spacing.medium)
            .verticalScroll(rememberScrollState()),
        horizontalAlignment = Alignment.CenterHorizontally,
        verticalArrangement = Arrangement.Center
    ) {
        Image(
            painter = painterResource(Res.drawable.logo),
            contentDescription = stringResource(Res.string.cd_app_logo),
            modifier = Modifier.size(Dimens.Component.logo)
        )

        Spacer(modifier = Modifier.height(Dimens.Spacing.large))

        Text(
            text = stringResource(Res.string.auth_signin_title),
            style = MaterialTheme.typography.headlineMedium,
            fontWeight = FontWeight.Bold
        )

        Text(
```

```
        text = stringResource(Res.string.auth_signin_subtitle),
        style = MaterialTheme.typography.bodyMedium,
        color = MaterialTheme.colorScheme.onSurfaceVariant
    )

    Spacer(modifier = Modifier.height(Dimens.Spacing.xxxLarge))

    OutlinedTextField(
        value = state.email,
        onValueChange = { onAction(SignInAction.OnEmailChange(it)) },
        label = { Text(text = stringResource(Res.string.auth_email)) },
        keyboardOptions = KeyboardOptions(
            keyboardType = KeyboardType.Email,
            imeAction = ImeAction.Next
        ),
        modifier = Modifier.fillMaxWidth(),
        shape = MaterialTheme.shapes.medium,
        isError = state.emailError != null,
        supportingText = state.emailError?.let { { Text(text = it.asString()) } },
        singleLine = true,
        maxLines = 1
    )

    Spacer(modifier = Modifier.height(Dimens.Spacing.medium))

    OutlinedTextField(
        value = state.password,
        onValueChange = { onAction(SignInAction.OnPasswordChange(it)) },
        label = { Text(text = stringResource(Res.string.auth_password)) },
        visualTransformation = if (state.isPasswordVisible) VisualTransformation.None else
PasswordVisualTransformation(),
        trailingIcon = {
            IconButton(onClick = { onAction(SignInAction.OnTogglePasswordVisibility) }) {
                Icon(
                    painter = painterResource(
                        if (state.isPasswordVisible) Res.drawable.ic_visibility_off else Res.drawable.ic_visibility
                    ),
                    contentDescription = stringResource(if (state.isPasswordVisible) Res.string.cd_hide_password else
Res.string.cd_show_password)
```

```
            )
        }
    },
    keyboardOptions = KeyboardOptions(
        keyboardType = KeyboardType.Password,
        imeAction = ImeAction.Done
    ),
    keyboardActions = androidx.compose.foundation.text.KeyboardActions(
        onDone = { onAction(SignInAction.OnSignIn) }
    ),
    modifier = Modifier.fillMaxWidth(),
    shape = MaterialTheme.shapes.medium,
    isError = state.passwordError != null,
    supportingText = state.passwordError?.let { { Text(text = it.asString()) } },
    singleLine = true,
    maxLines = 1
)

Spacer(modifier = Modifier.height(Dimens.Spacing.small))

Row(
    modifier = Modifier
        .fillMaxWidth()
        .clickable { onAction(SignInAction.OnRememberMeToggle(!state.isRememberMeChecked)) },
    verticalAlignment = Alignment.CenterVertically
) {
    Checkbox(
        checked = state.isRememberMeChecked,
        onCheckedChange = { onAction(SignInAction.OnRememberMeToggle(it)) }
    )
    Text(
        text = stringResource(Res.string.auth_remember_me),
        style = MaterialTheme.typography.bodyMedium
    )
}

Spacer(modifier = Modifier.height(Dimens.Spacing.xLarge))
```

```
        Button(
            onClick = { onAction(SignInAction.OnSignIn) },
            modifier = Modifier
                .fillMaxWidth()
                .height(Dimens.TouchTarget.medium),
            enabled = !state.isLoading,
            shape = MaterialTheme.shapes.medium
        ) {
            Text(
                text = stringResource(Res.string.auth_sign_in),
                style = MaterialTheme.typography.titleMedium,
                fontWeight = FontWeight.Bold
            )
        }

        Spacer(modifier = Modifier.height(Dimens.Spacing.medium))

        Row(
            verticalAlignment = Alignment.CenterVertically
        ) {
            Text(
                text = stringResource(Res.string.auth_dont_have_account),
                style = MaterialTheme.typography.bodyMedium,
                color = MaterialTheme.colorScheme.onSurfaceVariant
            )
            TextButton(
                onClick = { onAction(SignInAction.OnSignUpClick) },
                enabled = !state.isLoading
            ) {
                Text(
                    text = stringResource(Res.string.auth_sign_up),
                    fontWeight = FontWeight.Bold
                )
            }
        }
    }
}
```