



ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ & ΤΕΧΝΟΛΟΓΙΑΣ (ΣΘΕΤ)

Μεταπτυχιακή Εξειδίκευση στα Πληροφοριακά Συστήματα
(MSc)

Μεταπτυχιακή Διπλωματική Εργασία
Σύγχρονες Τεχνολογίες Λογισμικού για την ανάπτυξη
Cross - Platform Εφαρμογής Φυσικής Κατάστασης

Νικόλαος Γκαραγκούνης

Επιβλέπων καθηγητής: Γεώργιος Μαυρομμάτης

Πάτρα, Ιούνιος 2026

Η παρούσα εργασία αποτελεί πνευματική ιδιοκτησία του φοιτητή Γκαραγκούνη Νικόλαου που την εκπόνησε. Στο πλαίσιο της πολιτικής ανοικτής πρόσβασης ο συγγραφέας/δημιουργός εκχωρεί στο ΕΑΠ, μη αποκλειστική άδεια χρήσης του δικαιώματος αναπαραγωγής, προσαρμογής, δημόσιου δανεισμού, παρουσίασης στο κοινό και ψηφιακής διάχυσής τους διεθνώς, σε ηλεκτρονική μορφή και σε οποιοδήποτε μέσο, για διδακτικούς και ερευνητικούς σκοπούς, άνευ ανταλλάγματος και για όλο το χρόνο διάρκειας των δικαιωμάτων πνευματικής ιδιοκτησίας. Η ανοικτή πρόσβαση στο πλήρες κείμενο για μελέτη και ανάγνωση δεν σημαίνει καθ' οιονδήποτε τρόπο παραχώρηση δικαιωμάτων διανοητικής ιδιοκτησίας του συγγραφέα/δημιουργού ούτε επιτρέπει την αναπαραγωγή, αναδημοσίευση, αντιγραφή, αποθήκευση, πώληση, εμπορική χρήση, μετάδοση, διανομή, έκδοση, εκτέλεση, «μεταφόρτωση» (downloading), «ανάρτηση» (uploading), μετάφραση, τροποποίηση με οποιονδήποτε τρόπο, τμηματικά ή περιληπτικά της εργασίας, χωρίς τη ρητή προηγούμενη έγγραφη συναίνεση του/της συγγραφέα/δημιουργού. Ο συγγραφέας/δημιουργός διατηρεί το σύνολο των ηθικών και περιουσιακών του δικαιωμάτων.



Σύγχρονες Τεχνολογίες Λογισμικού για την ανάπτυξη
Cross - Platform Εφαρμογής Φυσικής Κατάστασης

Νικόλαος Γκαραγκούνης

Επιτροπή Κρίσης

Επιβλέπων Καθηγητής:
Γεώργιος Μαυρομμάτης
Μέλος ΣΕΠ ΕΑΠ

Συν-Επιβλέπων Καθηγητής:
Ανδρέας Καναβός
Αν. Καθηγητής Ιονίου Πανεπιστημίου

Πάτρα, Ιούνιος 2026

Περίληψη

Αντικείμενο της παρούσας μεταπτυχιακής διπλωματικής εργασίας αποτελεί η μελέτη, ο σχεδιασμός και η υλοποίηση μιας προηγμένης, offline-first cross-platform εφαρμογής φυσικής κατάστασης και υγείας με την ονομασία "FitnessApp". Για την ανάπτυξη του συστήματος επιστρατεύτηκαν οι πλέον σύγχρονες τεχνολογίες λογισμικού, με κεντρικό πυλώνα το πλαίσιο Kotlin Multiplatform (KMP) και το Compose Multiplatform, επιτρέποντας τη δημιουργία μιας ενιαίας βάσης κώδικα (Shared Codebase) για τη διαχείριση της επιχειρηματικής λογικής και της γραφικής διεπαφής χρήστη (Shared UI). Η εφαρμογή σχεδιάστηκε με γνώμονα τη σχεδιαστική ομοιομορφία και τη βέλτιστη δυνατή ακρίβεια (pixel-perfect) σε περιβάλλοντα φορητών συσκευών Android και iOS, καθώς και σε επιτραπέζιους υπολογιστές (Desktop). Αρχιτεκτονικά, το σύστημα ακολουθεί τις αρχές της Καθαρής Αρχιτεκτονικής (Clean Architecture) και του μοτίβου MVVM/MVI, διασφαλίζοντας τον πλήρη διαχωρισμό των ευθυνών. Για την επίτευξη της offline-first λειτουργικότητας, ενσωματώθηκε η τοπική βάση δεδομένων SQLDelight, η οποία, μέσω ενός custom μηχανισμού αμφίδρομου συγχρονισμού (SyncManager), επικοινωνεί με την cloud πλατφόρμα Supabase (PostgreSQL backend). Επιπλέον, ένα από τα πιο καινοτόμα χαρακτηριστικά της εφαρμογής είναι η ενσωμάτωση του Gemini API, το οποίο τροφοδοτεί τον ψηφιακό βοηθό τεχνητής νοημοσύνης "PocketCoach AI", προσφέροντας διαδραστική καθοδήγηση φυσικής κατάστασης και πρόταση αρχικά περιγραφής και τίτλου άσκησης εκγύμνασης και στην συνέχεια αποθήκευση. Τα πειραματικά αποτελέσματα επιβεβαίωσαν την επιτυχία του εγχειρήματος, αναδεικνύοντας τη σταθερότητα και την υψηλή απόδοση του συστήματος. Η σχεδιαστική σύγκλιση επαληθεύτηκε μέσω δοκιμών σε Android και Desktop συσκευές, όπου τα στοιχεία του Material Design 3 προσαρμόστηκαν δυναμικά σε Portrait και Landscape διατάξεις. Παράλληλα, η cross-platform ορθότητα και η συμβατότητα του κώδικα για το περιβάλλον του iOS διασφαλίστηκαν πλήρως μέσω αυτοματοποιημένων cloud-based μεταγλωττίσεων (Continuous Integration) στους native runners του GitHub Actions. Συμπερασματικά, η εργασία αποδεικνύει ότι το οικοσύστημα της Kotlin Multiplatform είναι πλήρως ώριμο για την παραγωγή σύνθετων, αποδοτικών και εύκολα συντηρήσιμων εφαρμογών, γεφυρώνοντας το χάσμα μεταξύ διαφορετικών λειτουργικών συστημάτων και ενσωματώνοντας με επιτυχία λύσεις τεχνητής νοημοσύνης στην καθημερινή άσκηση.

Λέξεις – Κλειδιά

Kotlin Multiplatform, Compose Multiplatform, Cross-Platform Ανάπτυξη, Τεχνητή
Νοημοσύνη, Offline-First Αρχιτεκτονική, Εφαρμογή Φυσικής Κατάστασης

Modern Software Technologies for Developing a Cross-Platform Fitness Application

Nikolaos Gkaragkounis

Abstract

The subject of this postgraduate master's thesis is the study, design, and implementation of an advanced, offline-first cross-platform mobile and desktop application for fitness and health, titled "FitnessApp". The system was developed using cutting-edge software engineering technologies, with Kotlin Multiplatform (KMP) and Compose Multiplatform serving as its core pillars. This approach enabled the creation of a single shared codebase for both the business logic and the graphical user interface (Shared UI). The application was engineered with a focus on design consistency and pixel-perfect rendering across Android and iOS mobile environments, as well as desktop environments. Architecturally, the system strictly adheres to the principles of Clean Architecture and the MVVM/MVI design patterns, ensuring a thorough separation of concerns. To achieve robust offline-first functionality, the SQLDelight local database was integrated, interacting seamlessly with the cloud-based Supabase platform (PostgreSQL backend) through a custom bidirectional synchronization mechanism (SyncManager). Furthermore, a highly innovative feature of the application is the integration of the Gemini API, which powers "PocketCoach AI", an interactive artificial intelligence virtual assistant that provides personalized fitness guidance and provides the user new exercise to save for workout plan. The experimental results confirmed the success of the project, demonstrating high performance, responsiveness, and structural stability. Design convergence was verified through testing on Android and Desktop devices, where Material Design 3 elements adapted dynamically to Portrait and Landscape layouts. Simultaneously, the cross-platform validity and compilation compatibility for the iOS environment were fully validated via cloud-based automated workflows (Continuous Integration) on native Apple runners within GitHub Actions. In conclusion, this thesis demonstrates that the Kotlin Multiplatform ecosystem is

fully mature for producing complex, scalable, and highly maintainable software, bridging the gap between heterogeneous operating systems while successfully embedding AI solutions into daily fitness routines.

Keywords

Kotlin Multiplatform, Compose Multiplatform, Cross-Platform Development, Artificial Intelligence, Offline-First Architecture, Fitness Application

Περιεχόμενα

Περίληψη.....	iv
Abstract	vi
Περιεχόμενα	viii
Κατάλογος Εικόνων / Σχημάτων	xi
Συντομογραφίες & Ακρωνύμια.....	xii
1. Εισαγωγή.....	1
1.1 Ανάπτυξη εφαρμογών για πολλαπλές πλατφόρμες.....	1
1.1.1 Ξεχωριστές native εφαρμογές για κάθε λειτουργικό σύστημα	1
1.1.2 Progressive Web App (PWA)	2
1.1.3 Εφαρμογές πολλαπλών πλατφορμών (Multiplatform Apps ή Cross – Platform apps)	2
1.1.4 Υβριδικές εφαρμογές	3
1.2 Οφέλη της ανάπτυξης cross – platform εφαρμογών	3
1.2.1 Επαναχρησιμοποιήσιμος κώδικας.....	3
1.2.2 Εξοικονόμηση χρόνου.....	4
1.2.3 Αποτελεσματική διαχείριση πόρων	4
1.2.4 Ευκαιρία προσέγγισης ευρύτερου κοινού	4
1.3 Cross - Platform επιλογές	4
1.3.1 Flutter	4
1.3.2 React Native	5
1.3.3 Kotlin Multiplatform.....	6
1.3.4 Ionic.....	6
1.3.5 .NET MAUI	7
1.3.6 Uno Platform.....	8
1.3.7 NativeScript.....	8
1.4 Επιλογή Cross Platform framework	9
2. Kotlin Multiplatform.....	10
2.1 Γιατί KMP	10
2.2 Η γλώσσα προγραμματισμού Kotlin	13
2.2.1 Ειδικά χαρακτηριστικά της γλώσσας προγραμματισμού Kotlin	13
2.3 Jetpack Compose	14
2.3.1 Συναρτήσεις	14
2.3.2 Material 3 Design	16
2.3.3 Annotations	16
2.3.4 Navigation	17
2.3.5 State.....	18
2.3.6 Lifecycle.....	18
2.3.7 ViewModel.....	19
2.4 Android Studio	20
2.4.1 Project structure	20
2.4.2 Gradle build system.....	21
2.4.3 Build variants	21
2.4.4 Multiple APK support	21

2.4.5	Resource shrinking.....	21
2.4.6	Manage dependencies	22
2.4.7	Annotations	22
2.4.8	Gemini.....	22
3.	Cross - Platform Εφαρμογή Φυσικής Κατάστασης	23
3.1	Λειτουργίες της εφαρμογής.....	23
3.1.1	Σύστημα Αυθεντικοποίησης και Ασφάλειας (User Authentication)	23
3.1.2	Ψηφιακός Βοηθός Τεχνητής Νοημοσύνης (PocketCoach AI).....	23
3.1.3	Δυναμική Βιβλιοθήκη Ασκήσεων (Exercises Library).....	23
3.1.4	Υπολογισμός και Καταγραφή Δείκτη Μάζας Σώματος (BMI Calculator & History) 24	
3.1.5	Διαχείριση Προσωπικού Προφίλ (Profile Management).....	24
3.1.6	Offline-First Λειτουργία και Αυτόματος Συγχρονισμός (Data Synchronization)	24
3.2	Υλοποίηση της εφαρμογής	25
3.2.1	Domain Layer.....	25
3.2.2	Τοπική βάση δεδομένων (Data Layer).....	25
3.2.3	Χρήση Backend-as-a-Service (BaaS) και Απομακρυσμένων Βάσεων Δεδομένων (Network Layer).....	32
3.2.4	Μηχανισμός Ταυτοποίησης (Authentication).....	45
3.2.5	Ενσωμάτωση Μηχανισμών Τεχνητής Νοημοσύνης (Network Layer - Gemini API)	47
3.2.6	Presentation Layer (MVI/MVVM Αρχιτεκτονική & Jetpack Compose)	53
3.2.7	Υλοποίηση Κεντρικής Διεπαφής Χρήστη (Presentation Layer – HomeScreen).....	60
3.2.8	Αρχιτεκτονική Πλοήγησης και Διασφάλιση Τύπων (Navigation Layer, Type-Safe Routes & TopAppBar)	62
3.2.9	Υλοποίηση Κεντρικής Συνάρτησης App και Ενορχήστρωση Επιπέδου Παρουσίασης.....	65
4.	Πειραματικά Αποτελέσματα και Διεπαφή Χρήστη (UI)	68
4.1	Οθόνη Υποδοχής και Εκκίνησης (Splash/Welcome Screen) σε Περιβάλλον Desktop	68
4.2	Οθόνη Υποδοχής και Εκκίνησης (Splash/Welcome Screen) σε Περιβάλλον Android	70
4.3	Οθόνη Αυθεντικοποίησης και Εγγραφής (Sign In/Sign Up) σε Περιβάλλον Desktop	71
4.4	Οθόνη Αυθεντικοποίησης και Εγγραφής (Sign In/Sign Up) σε Περιβάλλον Android	73
4.5	Κεντρική Οθόνη Διαχείρισης (Fitness Dashboard) σε Περιβάλλον Desktop	74
4.6	Κεντρική Οθόνη Διαχείρισης (Fitness Dashboard) σε Περιβάλλον Android	76
4.7	Οθόνη Ψηφιακού Βοηθού Τεχνητής Νοημοσύνης (PocketCoach AI) σε Περιβάλλον Desktop	78
4.8	Οθόνη Ψηφιακού Βοηθού Τεχνητής Νοημοσύνης (PocketCoach AI) σε Περιβάλλον Android	80
4.9	Οθόνη Διαχείρισης Προφίλ Χρήστη (Edit Profile) σε Περιβάλλον Desktop	82
4.10	Οθόνη Διαχείρισης Προφίλ Χρήστη (Edit Profile) σε Περιβάλλον Android...84	

4.11	Οθόνη Υπολογισμού Δείκτη Μάζας Σώματος (BMI Calculator) σε Περιβάλλον Desktop	85
4.12	Οθόνη Υπολογισμού Δείκτη Μάζας Σώματος (BMI Calculator) σε Περιβάλλον Android	87
4.13	Οθόνη Βιβλιοθήκης και Επιλεγμένων Ασκήσεων (Exercises Library) σε Περιβάλλον Desktop	89
4.14	Οθόνη Βιβλιοθήκης και Επιλεγμένων Ασκήσεων (Exercises Library) σε Περιβάλλον Android	91
4.15	Οθόνη Δημιουργίας Προσαρμοσμένης Άσκησης με Τεχνητή Νοημοσύνη σε περιβάλλον Desktop	93
4.16	Οθόνη Δημιουργίας Προσαρμοσμένης Άσκησης με Τεχνητή Νοημοσύνη σε περιβάλλον Android	94
4.17	Σχεδιαστική και Λειτουργική Σύγκλιση των Υπόλοιπων Ενοτήτων της Εφαρμογής	98
4.18	Διασφάλιση Ποιότητας και Μεταγλώττιση για το Λειτουργικό Σύστημα iOS μέσω GitHub Actions	99
5.	Συμπεράσματα, Δυσχέρειες και Μελλοντικές Επεκτάσεις	102
5.1	Δυσχέρειες που Αντιμετωπίστηκαν	102
5.2	Μελλοντικές Επεκτάσεις	104
5.3	Συμπεράσματα	104
	Βιβλιογραφία	106
	Παράρτημα Α: «Παραδείγματα κώδικα της γλώσσας προγραμματισμού Kotlin»	107
	Παράρτημα Β: Υποδείγματα της εφαρμογής φυσικής κατάστασης	113

Κατάλογος Εικόνων / Σχημάτων

Εικόνα 2.1. Παράδειγμα κοινής τεκμηρίωσης μεταξύ λογισμικών (kotlinlang.org, 2026).	12
Εικόνα 2.2. Επιλογή του επιπέδου κοινής χρήσης κώδικα (kotlinlang.org, 2026).	12
Εικόνα 3.1 Η ιδιότητα canNavigateBack	64
Εικόνα 4.1 Η οθόνη υποδοχής της εφαρμογής σε περιβάλλον Desktop	69
Εικόνα 4.2 Η οθόνη υποδοχής της εφαρμογής σε περιβάλλον Android	70
Εικόνα 4.3 Η οθόνη αυθεντικοποίησης της εφαρμογής σε περιβάλλον Desktop	72
Εικόνα 4.4 Η οθόνη αυθεντικοποίησης της εφαρμογής σε περιβάλλον Android	73
Εικόνα 4.5 Η κεντρική οθόνη διαχείρισης (Dashboard) της εφαρμογής σε περιβάλλον Desktop	75
Εικόνα 4.6 Η κεντρική οθόνη διαχείρισης (Dashboard) της εφαρμογής σε περιβάλλον Android	77
Εικόνα 4.7 Η οθόνη του ψηφιακού βοηθού (PocketCoach AI) σε περιβάλλον Desktop	79
Εικόνα 4.8 Η οθόνη του ψηφιακού βοηθού (PocketCoach AI) σε περιβάλλον Android	81
Εικόνα 4.9 Η οθόνη διαχείρισης προφίλ χρήστη (Edit Profile) σε περιβάλλον Desktop	82
Εικόνα 4.10 Η οθόνη διαχείρισης προφίλ χρήστη (Edit Profile) σε περιβάλλον Android	84
Εικόνα 4.11 Η οθόνη υπολογισμού Δείκτη Μάζας Σώματος (BMI Calculator) σε περιβάλλον Desktop	86
Εικόνα 4.12 Η οθόνη υπολογισμού Δείκτη Μάζας Σώματος (BMI Calculator) σε περιβάλλον Android	88
Εικόνα 4.13 Η οθόνη βιβλιοθήκης και επιλεγμένων ασκήσεων (Exercises Library) σε περιβάλλον Desktop	89
Εικόνα 4.14 Εικόνα 4.14: Η οθόνη βιβλιοθήκης και επιλεγμένων ασκήσεων (Exercises Library) σε περιβάλλον Android	91
Εικόνα 4.15 Διεπαφή χρήστη της οθόνης δημιουργίας custom άσκησης με AI (AiExerciseInsertScreen) σε περιβάλλον Desktop, με πεδία εισαγωγής φορμών και κουμπιά ελέγχου του Gemini API	93
Εικόνα 4.16 Οθόνη δημιουργίας custom άσκησης με AI σε περιβάλλον Android (1 ^ο Στάδιο), με εισαγωγή προσαρμοσμένου prompt για τετρακέφαλους και επιλογή παραμέτρων προπόνησης	95
Εικόνα 4.17 Οθόνη δημιουργίας custom άσκησης με AI σε περιβάλλον Android (Στάδιο 2), με αυτόματη συμπλήρωση του ονόματος και των αναλυτικών οδηγιών εκτέλεσης από το Gemini API	97
Εικόνα 4.18 Επιτυχείς αυτοματοποιημένες ροές μεταγλώττισης (CI Build Runs) στο GitHub Actions για την πλατφόρμα iOS	99
Σχήμα 1. Μέρη κοινού κώδικα που χρησιμοποιήθηκαν από τους χρήστες του KMP	3
Σχήμα 2. Η ραγδαία αύξηση των βιβλιοθηκών του KMP μετά το 2022 (kotlinlang.org, 2024)	10
Σχήμα 3. Η αύξηση χρήσης του KMP μεταξύ 2024 και 2025 (kotlinlang.org, 2026)	10
Σχήμα 4. Σύγκριση απόδοσης KMP σε σχέση με native προγραμματισμό (kotlinlang.org, 2024)	11
Σχήμα 5. Το Lifecycle ενός composable σε ένα Composition	19

Συντομογραφίες & Ακρωνύμια

ΜΔΕ	Μεταπτυχιακή Διπλωματική Εργασία
ΕΑΠ	Ελληνικό Ανοικτό Πανεπιστήμιο
ΘΕ	Θεματική Ενότητα
ΜΠΣ	Μεταπτυχιακό Πρόγραμμα Σπουδών
ΣΥΝ	Συντονιστής/Συντονίστρια
OS	Operating System
PWA	Progressive Web App
API	Application Programming Interface
MVP	Minimum Viable Product
UI	User Interface
UX	User Experience
JVM	Java Virtual Machine
IDE	Integrated Development Environment
NDK	Native Development Kit
ABI	Application Binary Interface
APK	Android Package Kit
hdpi	High Density Pixel Info
mdpi	Medium Density Pixel Info
SDK	Software Development Kit
BMI	Body Mass Index
RDBMS	Relational Database Management System
GDPR	General Data Protection Regulation
RLS	Row Level Security
JSON	JavaScript Object Notation
MVI	Model - View – Intent
MVVM	Model - View - ViewModel

1. Εισαγωγή

Στις μέρες μας, πολλές εταιρείες αντιμετωπίζουν την πρόκληση της δημιουργίας εφαρμογών για κινητά για πολλαπλές πλατφόρμες, ειδικά για Android και iOS. Αυτός είναι ο λόγος για τον οποίο οι λύσεις ανάπτυξης εφαρμογών για κινητά σε πολλαπλές πλατφόρμες έχουν αναδειχθεί ως μία από τις πιο δημοφιλείς τάσεις ανάπτυξης λογισμικού.

Σύμφωνα με την Statista, υπήρχαν 1,66 εκατομμύρια εφαρμογές για κινητά διαθέσιμες στο Google Play Store και 2,4 εκατομμύρια εφαρμογές στο App Store κατά το πρώτο τρίμηνο του 2026, με το Android και το iOS μαζί να αντιπροσωπεύουν πλέον το 97% της παγκόσμιας αγοράς λειτουργικών συστημάτων για κινητά (statista.com, 2026).

Τα παραπάνω στατιστικά αντιστοιχούν στις αγορές του δυτικού κόσμου όπως, *Ευρώπη, Αμερική, Αυστραλία* κτλ, καθώς στην Κίνα κυρίως και γενικότερα σε χώρες της Ασίας έχουν αναπτυχθεί λογισμικά τελείως ανεξάρτητα από τα λειτουργικά συστήματα (OS) Android και iOS.

Ειδικότερα στην Κίνα οι υπηρεσίες της Google είναι αποκλεισμένες οπότε τα smartphone χρησιμοποιούν ανεξάρτητα καταστήματα εφαρμογών και το Apple App Store ελέγχει περίπου το 20% του μεριδίου της αγοράς.

1.1 Ανάπτυξη εφαρμογών για πολλαπλές πλατφόρμες

Η ανάπτυξη εφαρμογών σε πολλαπλές πλατφόρμες είναι μια προσέγγιση η οποία επιτρέπει να δημιουργηθεί μια ενιαία εφαρμογή που λειτουργεί ομαλά σε πολλά λειτουργικά συστήματα. Σε εφαρμογές που λειτουργούν σε πολλαπλές πλατφόρμες, μπορεί να γίνει κοινή χρήση μέρους ή ακόμη και ολόκληρου του πηγαίου κώδικα. Αυτό σημαίνει ότι οι προγραμματιστές μπορούν να δημιουργήσουν και να αναπτύξουν στοιχεία που λειτουργούν τόσο σε Android όσο και σε iOS, desktop ή και στο διαδίκτυο χωρίς να χρειάζεται να τα επανακωδικοποιήσουν για κάθε μεμονωμένη πλατφόρμα.

Υπάρχουν τέσσερις βασικοί τρόποι για τη δημιουργία μιας εφαρμογής:

1.1.1 Ξεχωριστές native εφαρμογές για κάθε λειτουργικό σύστημα

Κατά τη δημιουργία native εφαρμογών, οι προγραμματιστές δημιουργούν μια εφαρμογή για ένα συγκεκριμένο OS και βασίζονται σε εργαλεία και γλώσσες προγραμματισμού που

έχουν σχεδιαστεί ειδικά για μία πλατφόρμα: Kotlin ή Java για Android, Objective-C ή Swift για iOS, native για desktop ή Javascript για web.

Αυτά τα εργαλεία και οι γλώσσες προγραμματισμού δίνουν πρόσβαση στις λειτουργίες και τις δυνατότητες ενός δεδομένου OS και επιτρέπουν την δημιουργία εφαρμογών που ανταποκρίνονται στις ανάγκες ενός χρήστη με εύχρηστα περιβάλλοντα εργασίας. Αλλά αν απαιτείται η προσέγγιση κοινού τόσο σε Android όσο και σε άλλα OS, θα πρέπει να δημιουργηθούν ξεχωριστές εφαρμογές, κάτι που απαιτεί πολύ χρόνο και προσπάθεια.

1.1.2 Progressive Web App (PWA)

Οι PWA συνδυάζουν τα χαρακτηριστικά των εφαρμογών για κινητά με λύσεις που χρησιμοποιούνται στην ανάπτυξη ιστοσελίδων. Σε γενικές γραμμές, προσφέρουν έναν συνδυασμό ιστότοπου και εφαρμογής για κινητά. Οι προγραμματιστές δημιουργούν PWA χρησιμοποιώντας τεχνολογίες ιστού, όπως JavaScript, HTML, CSS και WebAssembly.

Οι εφαρμογές ιστού δεν απαιτούν ξεχωριστή ομαδοποίηση ή διανομή και μπορούν να δημοσιευτούν στο διαδίκτυο. Είναι προσβάσιμες μέσω του προγράμματος περιήγησης στον υπολογιστή, το smartphone και το tablet και δεν χρειάζεται να εγκατασταθούν μέσω του Google Play ή του App Store.

Το μειονέκτημα εδώ είναι ότι οι χρήστες δεν μπορούν να αξιοποιήσουν όλες τις λειτουργίες της συσκευής τους ενώ χρησιμοποιούν την εφαρμογή, όπως επαφές, ημερολόγια, τηλέφωνο και άλλα στοιχεία, γεγονός που έχει ως αποτέλεσμα περιορισμένη εμπειρία χρήστη. Όσον αφορά την απόδοση της εφαρμογής, οι native εφαρμογές έχουν το προβάδισμα.

1.1.3 Εφαρμογές πολλαπλών πλατφορμών (Multiplatform Apps ή Cross – Platform apps)

Όπως αναφέρθηκε προηγουμένως, οι εφαρμογές πολλαπλών πλατφορμών έχουν σχεδιαστεί για να εκτελούνται με τον ίδιο τρόπο σε διαφορετικές πλατφόρμες. Τα cross-platform frameworks επιτρέπουν να γράφεται κώδικας που μπορεί να κοινοποιηθεί και να επαναχρησιμοποιηθεί με σκοπό την ανάπτυξη τέτοιων εφαρμογών.

Αυτή η προσέγγιση έχει πολλά πλεονεκτήματα, όπως η αποτελεσματικότητα σε σχέση τόσο με τον χρόνο όσο και με το κόστος.

1.1.4 Υβριδικές εφαρμογές

Η ανάπτυξη υβριδικών εφαρμογών είναι μια προσέγγιση που συνδυάζει native και διαδικτυακές τεχνολογίες. Απαιτεί την ενσωμάτωση κώδικα γραμμένου σε μια γλώσσα ανάπτυξης ιστοσελίδων, όπως HTML, CSS ή JavaScript, σε μια native εφαρμογή. Αυτό γίνεται με τη βοήθεια frameworks, όπως το Ionic Capacitor και το Apache Cordova, χρησιμοποιώντας πρόσθετα plugins για πρόσβαση στις native λειτουργίες των πλατφορμών.

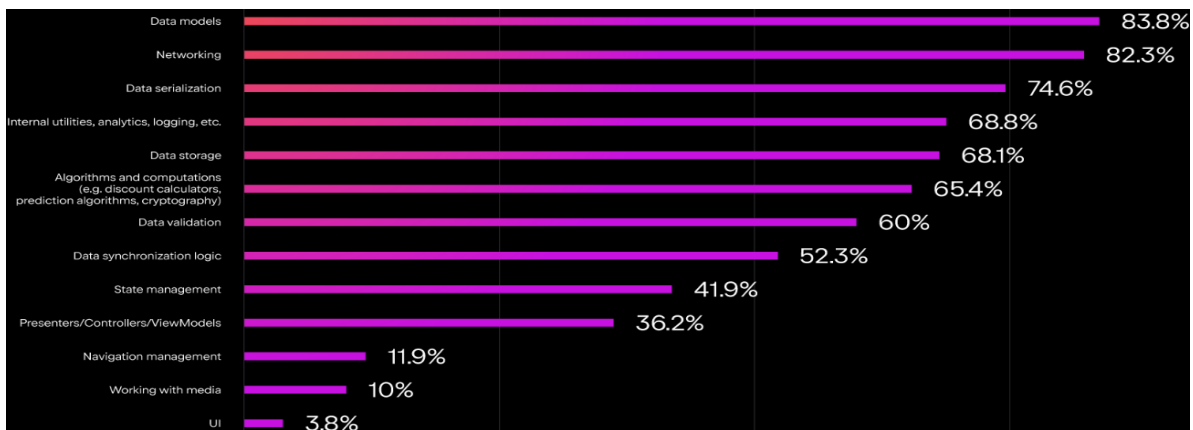
Η μόνη ομοιότητα μεταξύ της ανάπτυξης σε πολλαπλές πλατφόρμες και της υβριδικής ανάπτυξης είναι η δυνατότητα κοινής χρήσης κώδικα. Όσον αφορά την απόδοση, οι υβριδικές εφαρμογές δεν είναι εφάμιλλες με τις native εφαρμογές. Επειδή οι υβριδικές εφαρμογές αναπτύσσουν μία μόνο βάση κώδικα, ορισμένες λειτουργίες είναι συγκεκριμένες για ένα συγκεκριμένο OS και δεν λειτουργούν καλά σε άλλα.

1.2 Οφέλη της ανάπτυξης cross – platform εφαρμογών

Υπάρχουν πολλοί λόγοι για τους οποίους οι επιχειρήσεις επιλέγουν αυτήν την προσέγγιση έναντι άλλων επιλογών.

1.2.1 Επαναχρησιμοποιήσιμος κώδικας

Με τον cross - platform προγραμματισμό, δεν χρειάζεται να γράφεται νέος κώδικας για κάθε λειτουργικό σύστημα. Η χρήση μιας ενιαίας βάσης κώδικα επιτρέπει στους προγραμματιστές να μειώσουν τον χρόνο που αφιερώνουν στην εκτέλεση επαναλαμβανόμενων εργασιών, όπως κλήσεις API, αποθήκευση δεδομένων, serialization και υλοποίηση αναλυτικών στοιχείων.



Σχήμα 1. Μέρη κοινού κώδικα που χρησιμοποιήθηκαν από τους χρήστες του KMP.

1.2.2 Εξοικονόμηση χρόνου

Λόγω της δυνατότητας επαναχρησιμοποίησης κώδικα, οι cross – platform εφαρμογές απαιτούν λιγότερο κώδικα και όσον αφορά τον προγραμματισμό, λιγότερος κώδικας σημαίνει περισσότερος χρόνος. Εξοικονομείται χρόνος επειδή δεν απαιτείται η συγγραφή εκτενούς κώδικα. Επιπλέον, με λιγότερες γραμμές κώδικα, μειώνεται το περιθώριο εμφάνισης σφαλμάτων, με αποτέλεσμα να αφιερώνεται λιγότερος χρόνος στη δοκιμή και τη συντήρησή του.

1.2.3 Αποτελεσματική διαχείριση πόρων

Η δημιουργία ξεχωριστών εφαρμογών είναι δαπανηρή. Η ύπαρξη μιας ενιαίας βάσης κώδικα βοηθά στην αποτελεσματική διαχείριση των οικονομικών πόρων. Τόσο οι ομάδες ανάπτυξης Android όσο και iOS για παράδειγμα, μπορούν να μάθουν πώς να γράφουν και να χρησιμοποιούν κοινόχρηστο κώδικα.

1.2.4 Ευκαιρία προσέγγισης ευρύτερου κοινού

Δεν απαιτείται η επιλογή ανάμεσα σε διαφορετικά OS. Δεδομένου ότι η εφαρμογή είναι συμβατή με πολλά λειτουργικά συστήματα, καθίσταται δυνατή η ικανοποίηση των αναγκών τόσο του κοινού των Android όσο και των υπόλοιπων OS και έτσι μεγιστοποιείται η εμβέλεια των χρηστών στους οποίους απευθύνεται.

1.3 Cross - Platform επιλογές

Τα πιο δημοφιλή cross-platform frameworks περιλαμβάνουν τα Flutter, React Native και Kotlin Multiplatform. Κάθε ένα από αυτά τα framework έχει τις δικές του δυνατότητες και πλεονεκτήματα. Ανάλογα με το εργαλείο που χρησιμοποιείται, η διαδικασία ανάπτυξης και το αποτέλεσμα μπορεί να διαφέρουν.

1.3.1 Flutter

Δημιουργημένο από την Google, το Flutter είναι ένα πλαίσιο ανάπτυξης πολλαπλών πλατφορμών που χρησιμοποιεί τη γλώσσα προγραμματισμού Dart. Το Flutter υποστηρίζει native λειτουργίες, όπως υπηρεσίες τοποθεσίας, λειτουργικότητα κάμερας και πρόσβαση στον σκληρό δίσκο. Εάν χρειάζεται να δημιουργηθεί μια συγκεκριμένη λειτουργία εφαρμογής που δεν υποστηρίζεται στο Flutter, μπορεί να γραφτεί κώδικας ειδικά για την πλατφόρμα χρησιμοποιώντας την τεχνολογία Platform Channel.

Επίσης οι εφαρμογές που έχουν κατασκευαστεί με Flutter πρέπει να μοιράζονται όλα τα επίπεδα UX και UI. Ένα από τα καλύτερα χαρακτηριστικά αυτού του framework είναι η λειτουργία Hot Reload, η οποία επιτρέπει στους προγραμματιστές να κάνουν αλλαγές και να τις βλέπουν άμεσα.

Αυτό το framework μπορεί να είναι η καλύτερη επιλογή στις ακόλουθες περιπτώσεις:

- Εάν απαιτείται κοινή χρήση στοιχείων UI μεταξύ των εφαρμογών, αλλά οι εφαρμογές πρέπει να μοιάζουν πολύ με τις native.
- Η εφαρμογή αναμένεται να επιβαρύνει σημαντικά την CPU/GPU και η απόδοση είναι πιθανό να απαιτεί βελτιστοποίηση.
- Πρέπει να αναπτυχθεί ένα MVP (ελάχιστο βιώσιμο προϊόν).

Μεταξύ των πιο δημοφιλών εφαρμογών που έχουν κατασκευαστεί με το Flutter είναι οι Google Ads, Xianyu by Alibaba, eBay Motors και Hamilton.

1.3.2 React Native

Το Facebook παρουσίασε το React Native το 2015 ως ένα open source framework που έχει σχεδιαστεί για να βοηθά τους μηχανικούς υπολογιστών να δημιουργούν υβριδικές εφαρμογές native/cross-platform. Βασίζεται στο ReactJS, μια βιβλιοθήκη JavaScript για τη δημιουργία διεπαφών χρήστη. Με άλλα λόγια, χρησιμοποιεί JavaScript για τη δημιουργία εφαρμογών για κινητά με λειτουργικά συστήματα Android και iOS.

Το React Native παρέχει πρόσβαση σε διάφορες βιβλιοθήκες UI τρίτων με έτοιμα προς χρήση στοιχεία, βοηθώντας τους προγραμματιστές να εξοικονομήσουν χρόνο κατά τη διαδικασία ανάπτυξης. Όπως και το Flutter, επιτρέπει την παρακολούθηση όλων των αλλαγών αμέσως, χάρη στη λειτουργία Γρήγορης Ανανέωσης.

Η ανάπτυξη εφαρμογής με το React Native αποτελεί βέλτιστη πρακτική στις ακόλουθες περιπτώσεις:

- Η εφαρμογή είναι σχετικά απλή και αναμένεται να μην επιβαρύνει τη συσκευή .
- Η ομάδα ανάπτυξης χρησιμοποιεί σε υψηλό επίπεδο JavaScript ή React.

Οι εφαρμογές που έχουν κατασκευαστεί με το React Native περιλαμβάνουν το Facebook, το Instagram, το Skype και το Uber Eats.

1.3.3 Kotlin Multiplatform

Η Kotlin Multiplatform είναι μια τεχνολογία ανοιχτού κώδικα που αναπτύχθηκε από την JetBrains και επιτρέπει στους προγραμματιστές να μοιράζονται κώδικα σε διάφορες πλατφόρμες, διατηρώντας παράλληλα τα οφέλη του native προγραμματισμού. Τα βασικά της πλεονεκτήματα περιλαμβάνουν (Kotlin Multiplatform, 2026):

- Η δυνατότητα επαναχρησιμοποίησης κώδικα σε Android, iOS, web, desktop και server side, διατηρώντας παράλληλα τον native κώδικα, εάν χρειάζεται.
- Ομαλή ενσωμάτωση με υπάρχοντα έργα. Μπορεί να χρησιμοποιηθεί κώδικας ειδικά για κάθε πλατφόρμα, αξιοποιώντας παράλληλα στο έπακρο τόσο την native όσο και την πολλαπλή πλατφόρμα ανάπτυξης.
- Πλήρης ευελιξία κοινής χρήσης κώδικα και δυνατότητα κοινής χρήσης τόσο της λογικής όσο και του UI, χάρη στο Compose Multiplatform, ένα σύγχρονο δηλωτικό πλαίσιο UI πολλαπλών πλατφορμών που δημιουργήθηκε από την JetBrains.
- Δεν απαιτείται η εισαγωγή νέας γλώσσας στον κώδικα όταν χρησιμοποιείται ήδη το Kotlin για Android. Μπορεί να επαναχρησιμοποιηθεί ο κώδικας και η εμπειρία ενός προγραμματιστή στην γλώσσα προγραμματισμού Kotlin, γεγονός που καθιστά τη μετάβαση στο Kotlin Multiplatform λιγότερο επικίνδυνη σε σύγκριση με άλλες τεχνολογίες.

Στις εφαρμογές που έχουν κατασκευαστεί με την Kotlin Multiplatform περιλαμβάνονται οι McDonald's, Netflix, 9GAG, VMware, Cash App, Philips και πολλές άλλες εταιρείες που ήδη αξιοποιούν την ικανότητα σταδιακής ενσωμάτωσης του Kotlin Multiplatform, καθώς και τον χαμηλό κίνδυνο υιοθέτησής της.

1.3.4 Ionic

Το Ionic είναι ένα κιτ εργαλείων UI ανοιχτού κώδικα για κινητά, το οποίο κυκλοφόρησε το 2013. Βοηθά τους προγραμματιστές να δημιουργούν cross platform εφαρμογές για κινητά από μία μόνο βάση κώδικα χρησιμοποιώντας τεχνολογίες ιστού, όπως HTML, CSS και JavaScript, με ενσωματώσεις για τα frameworks Angular, React και Vue με τη γλώσσα προγραμματισμού JavaScript

Μερικά παραδείγματα εφαρμογών για κινητά που αναπτύχθηκαν με το Ionic είναι: T-Mobile, BBC (εφαρμογές για παιδιά και εκπαίδευση), EA Games.

Ορισμένα βασικά χαρακτηριστικά του είναι:

- Βασίζεται σε ένα πλαίσιο χρήστη SaaS που έχει σχεδιαστεί ειδικά για λειτουργικά συστήματα για κινητά και παρέχει πολλαπλά στοιχεία χρήστη για την κατασκευή εφαρμογών.
- Χρησιμοποιεί τα πρόσθετα Cordova και Capacitor για να παρέχει πρόσβαση στις ενσωματωμένες λειτουργίες της συσκευής, όπως η κάμερα, ο φακός, το GPS και η συσκευή εγγραφής ήχου.
- Έχει τη δική του διεπαφή γραμμής εντολών, το Ionic CLI, το οποίο χρησιμεύει ως το βασικό εργαλείο για την κατασκευή εφαρμογών Ionic.
- Υπάρχει συνεχής δραστηριότητα στο Ionic Framework Forum, όπου τα μέλη της κοινότητας ανταλλάσσουν γνώσεις και βοηθούν ο ένας τον άλλον να ξεπεράσουν τις προκλήσεις κατά την ανάπτυξη εφαρμογών.

1.3.5 .NET MAUI

Το .NET Multi-platform App UI (.NET MAUI) είναι ένα cross-platform framework που κυκλοφόρησε τον Μάιο του 2022 και ανήκει στη Microsoft. Επιτρέπει στους προγραμματιστές να δημιουργούν native εφαρμογές για κινητά και υπολογιστές με C# και XAML. Το .NET MAUI είναι μια εξέλιξη του Xamarin.Forms, μιας από τις λειτουργίες του Xamarin, η οποία παρέχει native στοιχεία ελέγχου για τις πλατφόρμες που υποστηρίζονται από το Xamarin.

Μερικά παραδείγματα εφαρμογών για κινητά που αναπτύχθηκαν με το .NET MAUI είναι: NBC Sports Next, Escola Agil, Irth Solutions.

Βασικά χαρακτηριστικά:

- Το .NET MAUI παρέχει cross-platform APIs για την πρόσβαση σε native λειτουργίες συσκευών, όπως το GPS, το επιταχυνσιόμετρο και οι καταστάσεις μπαταρίας και δικτύου.
- Παρέχεται ένα ενιαίο σύστημα έργων το οποίο ενεργοποιείται χρησιμοποιώντας πολλαπλή στόχευση για Android, iOS, macOS και Windows.
- Με την υποστήριξη για hot reload του .NET, οι προγραμματιστές μπορούν να τροποποιήσουν τον διαχειριζόμενο πηγαίο κώδικα τους ενώ εκτελείται η εφαρμογή.

1.3.6 Uno Platform

Η Uno Platform είναι μια ευέλικτη τεχνολογία ανοιχτού κώδικα για τη δημιουργία σύγχρονων εφαρμογών .NET σε πολλαπλές πλατφόρμες από μία μόνο κοινόχρηστη βάση κώδικα. Με σχεδιασμό εταιρικού επιπέδου και εργαλεία τεχνητής νοημοσύνης με βάση τα συμφραζόμενα, η πλατφόρμα Uno αυξάνει την παραγωγικότητα των προγραμματιστών για τη δημιουργία native εφαρμογών για κινητά, επιτραπέζιους υπολογιστές, ενσωματωμένων εφαρμογών OS και εφαρμογών WebAssembly χρησιμοποιώντας C# και XAML. Η Uno Platform είναι περισσότερο γνωστή για την ενσωμάτωση των μοντέλων προγραμματισμού WinUI/UWP σε πολλαπλές πλατφόρμες πέρα από τα Windows, επιτρέποντας στους προγραμματιστές .NET να επαναχρησιμοποιούν δεξιότητες και κώδικα σε ένα ευρύ φάσμα στόχων.

Μερικά παραδείγματα εφαρμογών για κινητά που αναπτύχθηκαν με το Uno Platform είναι: Μετεγκαταστημένες εφαρμογές από Toyota & Kahua, TradeZero, εταιρικές εφαρμογές που βασίζονται στο SkiaSharp.

Βασικά χαρακτηριστικά:

- Η Uno Platform επιτρέπει στους προγραμματιστές να έχουν πρόσβαση σε δυνατότητες native πλατφορμών, ενώ παράλληλα μοιράζονται ένα ενιαίο επίπεδο UI και επιχειρησιακής λογικής σε όλες τις πλατφόρμες, συμπεριλαμβανομένων των Android, iOS, WebAssembly (WASM), macOS, Linux και Windows. Υποστηρίζει μια ενιαία βάση κώδικα και δομή έργου, με πολλαπλή στόχευση που χρησιμοποιείται για την εκτέλεση της ίδιας εφαρμογής σε πολλαπλές πλατφόρμες χρησιμοποιώντας API συμβατά με WinUI.
- Σε συνδυασμό με το Uno Platform Studio, οι προγραμματιστές .NET μπορούν να επιτύχουν σημαντική αύξηση στην παραγωγικότητα με το Hot Design visual designer μέσω του Hot Reload για ταχύτερο βρόχο ανάπτυξης C#/XAML, με αξιόπιστα εργαλεία AI Agents/MCP για νοημοσύνη AI με βάση τα συμφραζόμενα και ευελιξία στην τεχνολογική στοίβα – όλα για τη δημιουργία εφαρμογών σε πολλαπλές πλατφόρμες από οποιοδήποτε OS/IDE/AI Agent.

1.3.7 NativeScript

Αυτό το framework ανάπτυξης εφαρμογών για κινητά είναι ανοιχτού κώδικα και κυκλοφόρησε αρχικά το 2014. Το NativeScript επιτρέπει τη δημιουργία εφαρμογών για

κινητά Android και iOS χρησιμοποιώντας JavaScript ή γλώσσες που μετατρέπονται σε JavaScript, όπως το TypeScript και frameworks όπως το Angular και το Vue.js.

Μερικά παραδείγματα εφαρμογών για κινητά που αναπτύχθηκαν με το NativeScript είναι: Daily Nanny, Strudel και Breethe.

Βασικά χαρακτηριστικά:

- Το NativeScript επιτρέπει στους προγραμματιστές να έχουν εύκολη πρόσβαση σε native API για Android και iOS.
- Το framework αποδίδει UIs βασισμένα σε πλατφόρμα. Οι εφαρμογές που έχουν κατασκευαστεί με NativeScript εκτελούνται απευθείας σε μια εγγενή συσκευή χωρίς να βασίζονται στο WebViews, ένα στοιχείο συστήματος για το λειτουργικό σύστημα Android που επιτρέπει στις εφαρμογές Android να εμφανίζουν περιεχόμενο από τον ιστό μέσα σε μια εφαρμογή.
- Προσφέρει διάφορα πρόσθετα και προκατασκευασμένα πρότυπα εφαρμογών, εξαλείφοντας την ανάγκη για λύσεις τρίτων.

1.4 Επιλογή Cross Platform framework

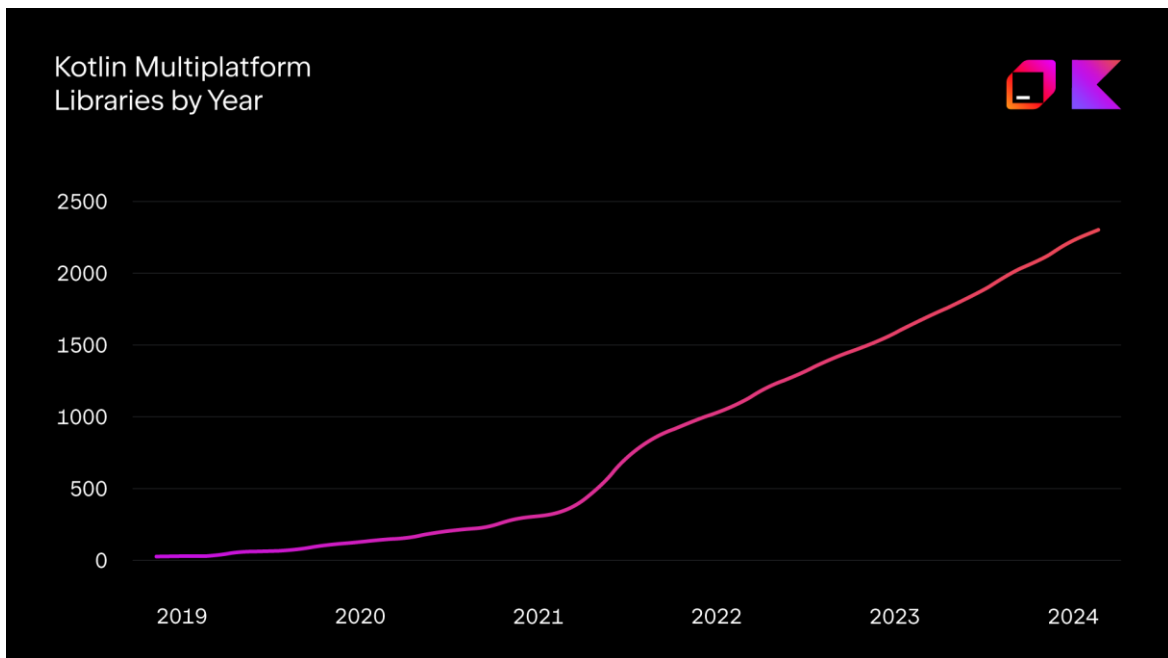
Η παρούσα Μεταπτυχιακή Διπλωματική Εργασία παρουσιάζει μία εφαρμογή παρακολούθησης της φυσικής κατάστασης του χρήστη προκειμένου να αναδείξει την δυνατότητα ανάπτυξης εφαρμογής με cross platform framework.

Για την ανάπτυξη της παραπάνω εφαρμογής επιλέχθηκε το KMP μέσω του Android Studio.

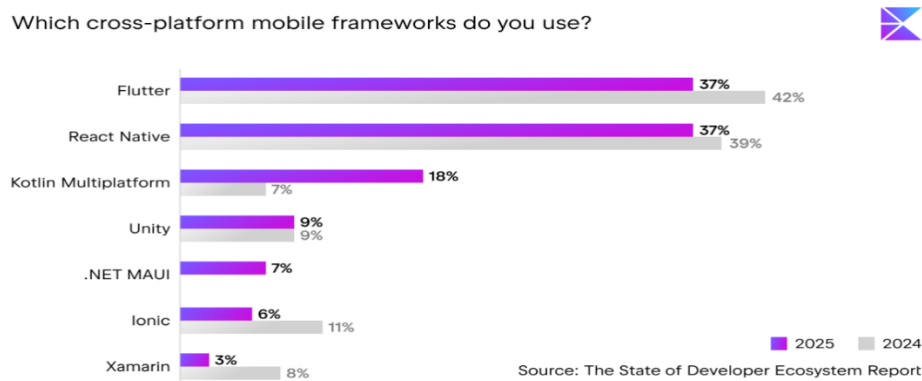
2. Kotlin Multiplatform

2.1 Γιατί KMP

Το KMP επιλέχθηκε διότι η γλώσσα προγραμματισμού Kotlin η οποία δημιουργήθηκε από την JetBrains έχει πλέον υιοθετηθεί από την εταιρεία Google και αποτελεί μια ταχύτατα αναπτυσσόμενη γλώσσα προγραμματισμού η οποία χρησιμοποιείται από χιλιάδες προγραμματιστές οι οποίοι δημιουργούν εφαρμογές που τρέχουν σε λογισμικό Android.



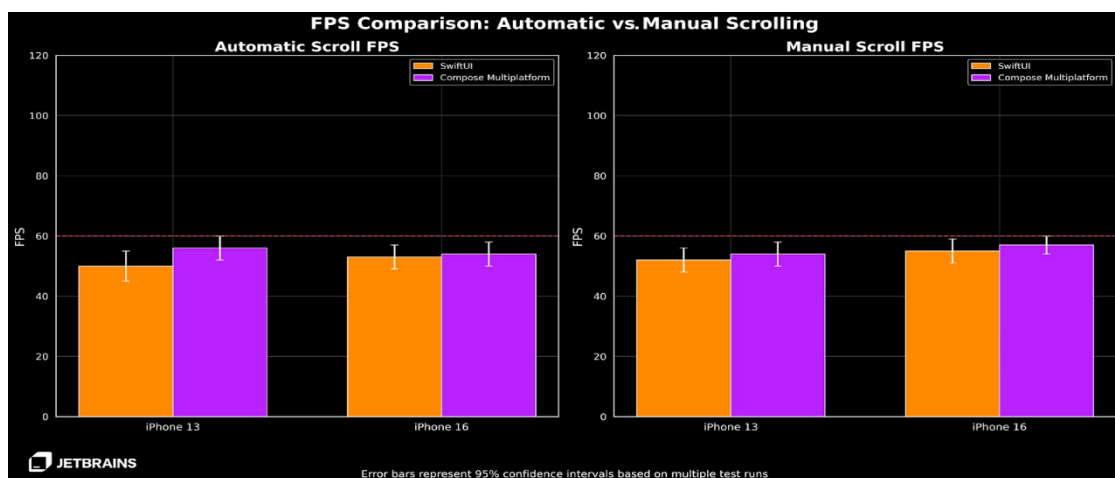
Σχήμα 2. Η ραγδαία αύξηση των βιβλιοθηκών του KMP μετά το 2022 (kotlinlang.org, 2024)



Σχήμα 3. Η αύξηση χρήσης του KMP μεταξύ 2024 και 2025 (kotlinlang.org, 2026).

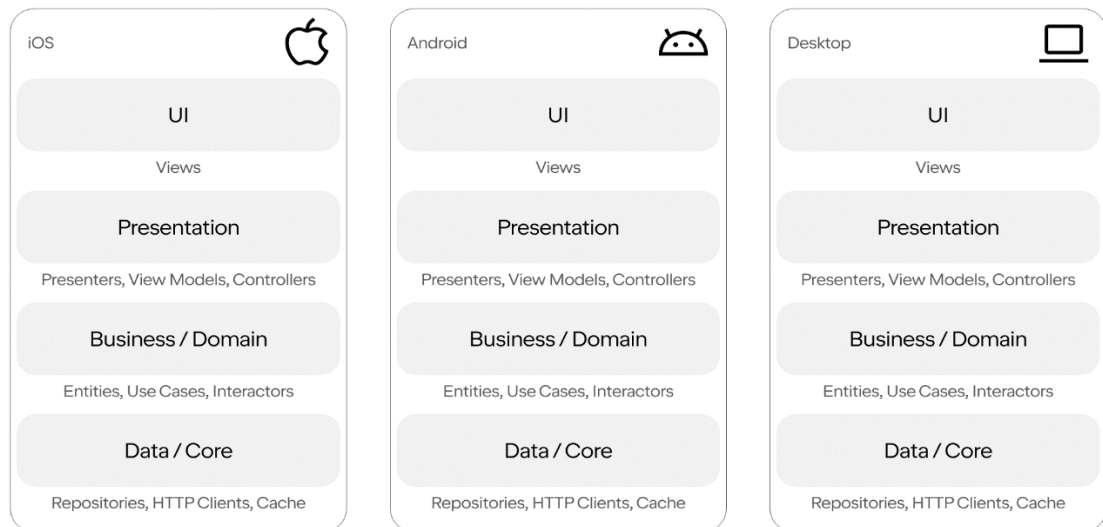
Ακόμη:

- Ο τρόπος που έχει δημιουργηθεί η γλώσσα προγραμματισμού Kotlin, επιτρέπει την χρησιμοποίησή της σε πολλαπλές πλατφόρμες διατηρώντας τις δυνατότητες που διαθέτει ο native προγραμματισμός.
- Είναι ευέλικτο και μπορεί να γίνει κοινή χρήση διακριτών ενοτήτων όπως δικτύωση ή αποθήκευση και στη συνέχεια να επεκταθεί σταδιακά ο κοινόχρηστος κώδικας με την πάροδο του χρόνου. Μπορεί επίσης να γίνει κοινή χρήση όλης της επιχειρηματικής λογικής διατηρώντας παράλληλα το περιβάλλον χρήστη native ή να μετατραπεί σταδιακά το περιβάλλον χρήστη σε Compose Multiplatform χωρίς να παραχωρηθεί πρόσβαση σε native στοιχεία περιβάλλοντος χρήστη, συμπεριλαμβανομένων πολύπλοκων στοιχείων όπως προγράμματα αναπαραγωγής βίντεο ή χάρτες.
- Συνοδεύεται από καλά εργαλεία όπως το IntelliJ IDEA και το Android Studio τα οποία προσφέρουν έξυπνη υποστήριξη IDE για KMP μέσω του πρόσθετου Kotlin Multiplatform IDE , το οποίο περιλαμβάνει κοινές προεπισκοπήσεις UI, hot reload για Compose Multiplatform, διαγλωσσική πλοήγηση, αναδιαμορφώσεις και εργαλεία για εντοπισμό σφαλμάτων σε κώδικα Kotlin και Swift.
- Προσφέρει native απόδοση καθώς το KMP χρησιμοποιεί το Kotlin/Native για τη δημιουργία native δυαδικών αρχείων και την άμεση πρόσβαση σε API πλατφόρμας σε περιπτώσεις όπου οι εικονικές μηχανές είναι ανεπιθύμητες ή μη διαθέσιμες, όπως στο iOS. Αυτό επιτρέπει σχεδόν native απόδοση, δημιουργώντας παράλληλα κώδικα που δεν εξαρτάται από την πλατφόρμα:



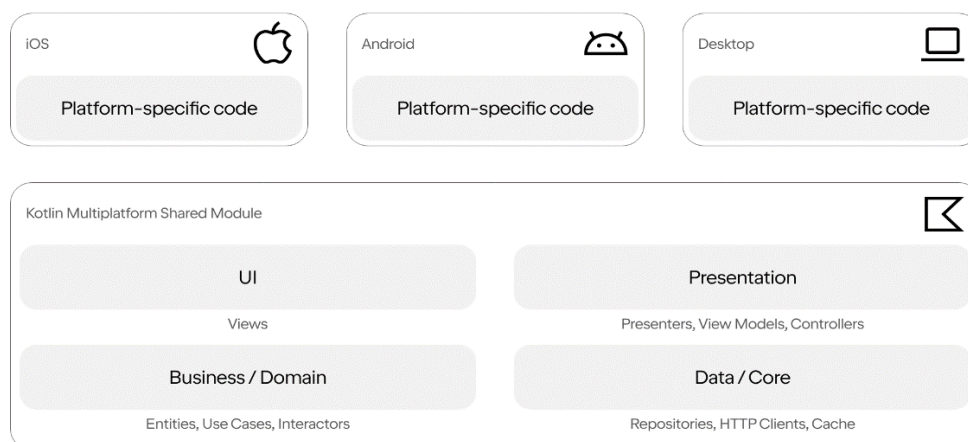
Σχήμα 4. Σύγκριση απόδοσης KMP σε σχέση με native προγραμματισμό (kotlinlang.org, 2024).

- Ένα από τα βασικά πλεονεκτήματα του Kotlin Multiplatform είναι η εκτεταμένη υποστήριξή του σε διάφορες πλατφόρμες, καθιστώντας το μια ευέλικτη επιλογή για προγραμματιστές. Αυτές οι πλατφόρμες περιλαμβάνουν Android, iOS, υπολογιστές, web (JavaScript και WebAssembly) και server (Java Virtual Machine).



Εικόνα 2.1. Παράδειγμα κοινής τεκμηρίωσης μεταξύ λογισμικών (kotlinlang.org, 2026).

- Το Kotlin Multiplatform επιτρέπει την επιλογή του επιπέδου κοινής χρήσης κώδικα που λειτουργεί για το έργο.



Εικόνα 2.2. Επιλογή του επιπέδου κοινής χρήσης κώδικα (kotlinlang.org, 2026).

2.2 Η γλώσσα προγραμματισμού Kotlin

Η Kotlin είναι μια αντικειμενοστραφής γλώσσα προγραμματισμού που σχεδιάστηκε από την ομάδα JetBrains και τρέχει πάνω στην Εικονική Μηχανή της Java (JVM). Έχει επηρεαστεί από τις γλώσσες Java, Scala, C#, Groovy, είναι συμβατή με την Java TM και Android TM και έχει σχεδιαστεί για να τρέχει σαν native κώδικας στα iOS και macOS.

Από τον Φεβρουάριο του 2012 η Kotlin αναπτύσσεται ως γλώσσα προγραμματισμού ανοικτού κώδικα. Η πρώτη επίσημη έκδοση 1.0 δημοσιεύτηκε τον Φεβρουάριο του 2016 και τον Μάιο του 2017 η Google ανακοίνωσε ότι η Kotlin θα είναι επίσημη γλώσσα του Android.

Η γλώσσα Kotlin αξιοποιεί τις υπάρχουσες βιβλιοθήκες για την εικονική μηχανή της Java (JVM), για Android και web browser και ένα πρόγραμμα Kotlin μπορεί να αναπτυχθεί σε ολοκληρωμένο περιβάλλον ανάπτυξης IntelliJ IDEA, Android Studio, Eclipse, NetBeans, Visual Studio Code. Ο κώδικας Kotlin μπορεί να περιέχει κώδικα Java και ο κώδικας Java να περιέχει Kotlin καθώς ο κώδικας Java μετατρέπεται σε κώδικα Kotlin.

2.2.1 Ειδικά χαρακτηριστικά της γλώσσας προγραμματισμού Kotlin

Null safety

Σε ορισμένες γλώσσες, μια μεταβλητή τύπου αναφοράς (reference type variable) μπορεί να δηλωθεί χωρίς να παρέχεται μια αρχική ρητή τιμή. Σε αυτές τις περιπτώσεις, οι μεταβλητές συνήθως περιέχουν μια τιμή null. Οι μεταβλητές στην γλώσσα προγραμματισμού Kotlin δεν μπορούν να διατηρήσουν τιμές null από προεπιλογή. Για να μπορεί μια μεταβλητή να κρατήσει μια τιμή null, πρέπει να είναι τύπου nullable.

Η Kotlin εφαρμόζει σκόπιμα συντακτικούς κανόνες, ώστε να μπορεί να επιτύχει ασφάλεια null, η οποία αναφέρεται στην εγγύηση ότι δεν γίνονται τυχαίες κλήσεις σε πιθανώς null μεταβλητές. Αυτό δεν σημαίνει ότι οι μεταβλητές δεν μπορούν να είναι null. Σημαίνει ότι εάν γίνει πρόσβαση σε ένα μέλος μιας μεταβλητής, η μεταβλητή δεν μπορεί να είναι null.

Αυτό είναι κρίσιμο επειδή εάν υπάρξει προσπάθεια πρόσβασης σε ένα μέλος μιας μεταβλητής που είναι null «γνωστό ως null reference» κατά την εκτέλεση μιας εφαρμογής, η εφαρμογή καταρρέει επειδή η μεταβλητή null δεν περιέχει καμία ιδιότητα ή μέθοδο. Αυτός ο τύπος κατάρρευσης είναι γνωστός ως σφάλμα χρόνου εκτέλεσης, στο οποίο το σφάλμα συμβαίνει μετά τη μεταγλώττιση και την εκτέλεση του κώδικα.

Λόγω της φύσης του null safety της Kotlin, τέτοια σφάλματα χρόνου εκτέλεσης αποτρέπονται επειδή ο μεταγλωττιστής της επιβάλλει έναν έλεγχο για τύπους null. Ο έλεγχος αναφέρεται στο εάν μια μεταβλητή θα μπορούσε να είναι null πριν από την πρόσβαση και την αντιμετώπισή της ως μη null τύπος. (developer.android.com, 2026)

Conditionals

Η Kotlin διαθέτει διάφορους μηχανισμούς για την υλοποίηση λογικών πράξεων. Ο πιο συνηθισμένος από αυτούς είναι μια εντολή if-else. Η υλοποίηση λογικών πράξεων της Kotlin αποτελούν ένα από τα πιο ισχυρά χαρακτηριστικά του, την έξυπνη μετατροπή. Αντί να χρησιμοποιείται ο τελεστής safe call (?) ή ο τελεστής not-null (!) για να υλοποιηθεί μια εργασία με τιμές null, είναι δυνατό να ελεγχθεί αν μια μεταβλητή περιέχει μια αναφορά σε μια τιμή null χρησιμοποιώντας ένα conditional statement, όπως φαίνεται στο [παράδειγμα 1](#) στο Παράρτημα «Α» (developer.android.com, 2026).

Interoperability

Ένα από τα πιο σημαντικά χαρακτηριστικά της Kotlin είναι η διαλειτουργικότητά της με την Java. Επειδή ο κώδικας της Kotlin μεταγλωττίζεται σε bytecode JVM, ο κώδικας Kotlin μπορεί να καλέσει απευθείας κώδικα Java και αντίστροφα. Αυτό σημαίνει ότι δύναται να αξιοποιηθούν υπάρχουσες βιβλιοθήκες Java απευθείας από την Kotlin. Επιπλέον, η πλειονότητα των API Android είναι γραμμένα σε Java και έτσι καθίσταται δυνατή η απευθείας κλήση από την Kotlin (developer.android.com, 2026).

2.3 Jetpack Compose

Το Jetpack Compose είναι το συνιστώμενο σύγχρονο κιτ εργαλείων του Android για τη δημιουργία native UI. Απλοποιεί και επιταχύνει την ανάπτυξη UI σε Android χρησιμοποιώντας λιγότερο κώδικα. Παρακάτω παρουσιάζονται μερικά εργαλεία του Jetpack Compose.

2.3.1 Συναρτήσεις

Text

Το κείμενο είναι ένα κεντρικό κομμάτι κάθε UI και το Jetpack Compose διευκολύνει την εμφάνιση ή τη σύνταξη κειμένου. Το Compose αξιοποιεί τη σύνθεση των δομικών του στοιχείων, πράγμα που σημαίνει ότι δεν χρειάζεται να αντικατασταθούν ιδιότητες και

μέθοδοι ή να επεκταθούν μεγάλες κλάσεις για να υπάρξει ένας συγκεκριμένος σχεδιασμός και λογική σύνθεσης που να λειτουργεί όπως την θέλει ο προγραμματιστής.

Ως βάση του, το Compose παρέχει ένα `BasicText` και ένα `BasicTextField`, τα οποία είναι τα βασικά στοιχεία για την εμφάνιση κειμένου και τη διαχείριση της εισόδου του χρήστη. Σε υψηλότερο επίπεδο, το Compose παρέχει `Text` και `TextField`, τα οποία είναι `composables` που ακολουθούν τις οδηγίες Material Design. Συνιστάται η χρήση τους, καθώς έχουν τη σωστή εμφάνιση και αίσθηση για τους χρήστες στο Android και περιλαμβάνουν επιπλέον επιλογές για την απλοποίηση της προσαρμογής τους χωρίς να χρειάζεται να γραφτεί πολύ κώδικας ([Παράδειγμα 2](#))

Column

`Column` είναι μια συνάρτηση με δυνατότητα σύνθεσης στο Jetpack Compose που τακτοποιεί τα εξαρτώμενα στοιχεία της σε μια κατακόρυφη ακολουθία ([Παράδειγμα 3](#)). Είναι ένα θεμελιώδες δομικό στοιχείο για τη δημιουργία UI με κατακόρυφα στοιβαγμένα στοιχεία (Jaykishan Sewak, 2024).

Card

Οι `Card` είναι απαραίτητα στοιχεία UI στο Material Design που οργανώνουν το περιεχόμενο σε διαχειρίσιμα κοντέινερ ([Παράδειγμα 4](#)). Το Jetpack Compose Material 3 προσφέρει έξι τύπους καρτών, ο καθένας με συγκεκριμένες παραμέτρους και περιπτώσεις χρήσης (Ramadan Sayed, 2025).

Image

Η συνάρτηση `Image` στο Jetpack Compose χρησιμοποιείται για την εμφάνιση εικόνων στο UI της εφαρμογής ([Παράδειγμα 5](#)). Μπορεί να φορτώσει εικόνες από διάφορες πηγές, συμπεριλαμβανομένων τοπικών πόρων, URL δικτύου και πόρων με δυνατότητα σχεδίασης. Η συνάρτηση `Image` παρέχει επίσης διάφορες ιδιότητες και συναρτήσεις που μπορούν να χρησιμοποιηθούν για την προσαρμογή της εμφάνισης και της συμπεριφοράς της εικόνας.

AnimatedVisibility

Το `composable AnimatedVisibility` ενεργοποιεί την εμφάνιση και την εξαφάνιση του περιεχομένου του, καθώς αλλάζει η ορατή τιμή ([Παράδειγμα 6](#)).

2.3.2 Material 3 Design

Το Material Design είναι ένα σύστημα σχεδιασμού που έχει κατασκευαστεί και υποστηρίζεται από σχεδιαστές και προγραμματιστές της Google. Το Material.io περιλαμβάνει εις βάθος καθοδήγηση UX και υλοποιήσεις στοιχείων UI για Android, Flutter και Web.

Η τελευταία έκδοση, Material 3, επιτρέπει προσωπικές, προσαρμοστικές και εκφραστικές εμπειρίες, από δυναμικά χρώματα και βελτιωμένη προσβασιμότητα, έως βάσεις για διατάξεις μεγάλης οθόνης και διακριτικά σχεδίασης. Το M3 Expressive πηγαίνει ένα βήμα παραπέρα προσθέτοντας πιο ευέλικτα στοιχεία, ζωντανά στυλ και πλήρως ενσωματωμένη κίνηση (m3.material.io, 2026).

2.3.3 Annotations

Στο Jetpack Compose, τα annotations (σχολιασμοί) παίζουν κρίσιμο ρόλο στον ορισμό των συναρτήσεων που μπορούν να συντεθούν και της συμπεριφοράς τους. Παρακάτω αναλύονται ορισμένα annotations.

@Composable

Αυτό είναι το βασικό annotation που χρησιμοποιείται με συναρτήσεις που αναπαριστούν στοιχεία UI που μπορούν να συντεθούν. Ενημερώνει τον μεταγλωττιστή Jetpack Compose ότι η συνάρτηση είναι υπεύθυνη για τη δημιουργία του UI και θα πρέπει να υποβληθεί σε επεξεργασία ανάλογα.

@Preview:

Αυτό το annotation χρησιμοποιείται για την προεπισκόπηση συναρτήσεων που μπορούν να συντεθούν απευθείας μέσα στο Android Studio IDE. Επιτρέπει γρήγορη επανάληψη και οπτική ανατροφοδότηση σχετικά με τις αλλαγές στο περιβάλλον χρήστη χωρίς να χρειάζεται να δημιουργηθεί και να αναπτυχθεί η εφαρμογή σε μια συσκευή.

Για να λειτουργήσει σωστά απαιτείται η συνάρτηση που θα γίνει προεπισκόπηση να έχει το annotation @Composable και επίσης δεν δέχεται παραμέτρους ([Παράδειγμα 7](#)).

@OptIn(ExperimentalMaterialApi::class)

Στο Jetpack Compose, το annotation @OptIn(ExperimentalMaterialApi::class) χρησιμεύει ως μέτρο προειδοποίησης κατά τη χρήση λειτουργιών που έχουν χαρακτηριστεί ως πειραματικές στη βιβλιοθήκη Material Design.

2.3.4 Navigation

Το navigation αναφέρεται στις αλληλεπιδράσεις που επιτρέπουν στους χρήστες να πλοηγούνται σε διαφορετικά τμήματα περιεχομένου μέσα στην εφαρμογή καθώς και να εισέρχονται και να εξέρχονται από αυτά.

Το στοιχείο navigation του Android Jetpack περιλαμβάνει τη βιβλιοθήκη navigation, το πρόσθετο Safe Args Gradle και εργαλεία που βοηθούν στην πλοήγηση σε εφαρμογές. Το στοιχείο navigation χειρίζεται ποικίλες περιπτώσεις χρήσης πλοήγησης, από απλά κλικ σε κουμπιά έως πιο σύνθετα μοτίβα, όπως γραμμές εφαρμογών και το Navigation Drawer.

Πιο αναλυτικά η βιβλιοθήκη του navigation χρησιμοποιεί:

- Ένα γράφημα πλοήγησης (navigation graph) το οποίο περιγράφει όλους τους πιθανούς προορισμούς εντός της εφαρμογής και τις συνδέσεις μεταξύ τους. Τα γραφήματα πλοήγησης μπορούν να ενσωματωθούν για να φιλοξενήσουν υποροές στην εφαρμογή.
- Ένας προορισμός (destination) είναι ένας κόμβος στο γράφημα πλοήγησης στον οποίο γίνεται πλοήγηση. Αυτό μπορεί να είναι ένα στοιχείο composable, ένα ένθετο γράφημα πλοήγησης (nested navigation graph) ή ένα παράθυρο διαλόγου. Όταν ο χρήστης πλοηγείται στον προορισμό, η εφαρμογή εμφανίζει το περιεχόμενό του.
- Μια διαδρομή (route) προσδιορίζει έναν προορισμό και καθορίζει τα ορίσματα που απαιτούνται για την πλοήγηση σε αυτόν, αλλά δεν κάνει τίποτα για να περιγράψει το περιβάλλον χρήστη. Με αυτόν τον τρόπο, τα δεδομένα είναι ξεχωριστά από την αναπαράσταση, γεγονός που επιτρέπει να διατηρείται κάθε κομμάτι της υλοποίησης του περιβάλλοντος χρήστη ανεξάρτητο από τη συνολική δομή της εφαρμογής. Αυτό διευκολύνει, για παράδειγμα, τη δοκιμή και την αναδιάταξη των στοιχείων σύνθεσης στο έργο. Ένα βασικό παράδειγμα navigation φαίνεται στο [παράδειγμα 8](#) του Παραρτήματος «Α».

2.3.5 State

Το State σε μια εφαρμογή είναι οποιαδήποτε τιμή που μπορεί να αλλάξει με την πάροδο του χρόνου. Αυτός είναι ένας πολύ ευρύς ορισμός και περιλαμβάνει τα πάντα, από μια βάση δεδομένων Room έως μια μεταβλητή σε μια κλάση.

Το Compose είναι δηλωτικό (declarative) και ως εκ τούτου, ο μόνος τρόπος για να ενημερωθεί είναι καλώντας το ίδιο composable με νέα ορίσματα. Αυτά τα ορίσματα είναι αναπαραστάσεις της κατάστασης του UI. Κάθε φορά που ενημερώνεται μια κατάσταση, λαμβάνει χώρα ένα recomposition. Ως αποτέλεσμα, αντικείμενα όπως το TextField δεν ενημερώνονται αυτόματα όπως συμβαίνει στις προβολές που βασίζονται σε imperative XML. Σε ένα composable απαιτείται ο ρητός ορισμός της νέας κατάστασης, ώστε να επιτευχθεί η αντίστοιχη ενημέρωση του UI.

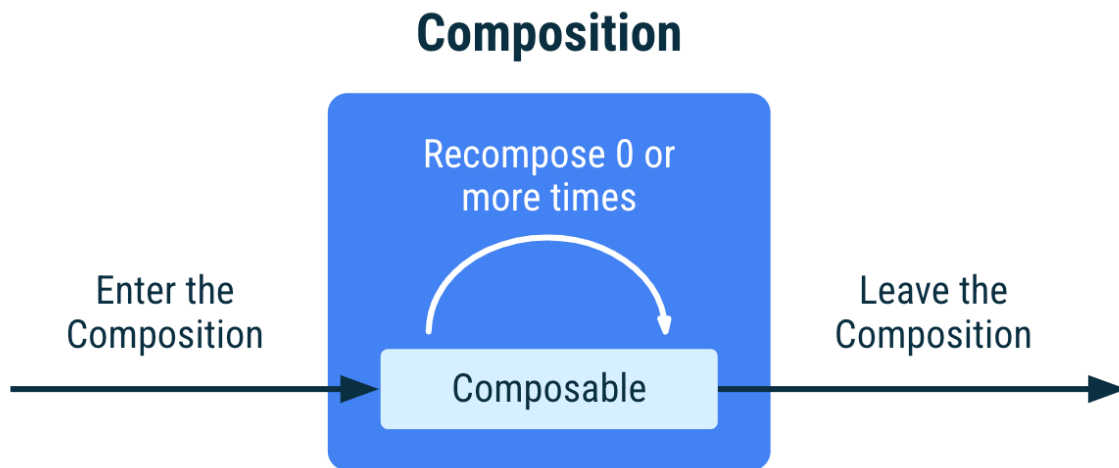
Οι composable συναρτήσεις μπορούν να χρησιμοποιήσουν την συνάρτηση remember για να αποθηκεύσουν ένα αντικείμενο στη μνήμη. Μια τιμή που υπολογίζεται από την remember αποθηκεύεται κατά την αρχική σύνθεση (composition) και η αποθηκευμένη τιμή επιστρέφεται κατά την ανασύνθεση (recomposition). Το remember μπορεί να χρησιμοποιηθεί για την αποθήκευση τόσο μεταβλητών όσο και αμετάβλητων αντικειμένων.

2.3.6 Lifecycle

Ένα composition περιγράφει το περιβάλλον χρήστη της εφαρμογής και παράγεται εκτελώντας compositions. Ένα composition είναι μια δένδροειδής δομή των composables που περιγράφουν το περιβάλλον χρήστη.

Όταν το Jetpack Compose εκτελεί τα composables για πρώτη φορά, κατά το αρχικό composition, θα παρακολουθεί τα composables που καλούνται για να περιγράψουν το περιβάλλον χρήστη σε ένα composition. Στη συνέχεια, όταν αλλάξει η κατάσταση της εφαρμογής, το Jetpack Compose προγραμματίζει ένα recomposition. Το recomposition συμβαίνει όταν το Jetpack Compose εκτελεί ξανά τα composables που μπορεί να έχουν αλλάξει ως απόκριση στις αλλαγές κατάστασης και στην συνέχεια, ενημερώνει το composition ώστε να αντικατοπτρίζει τυχόν αλλαγές ([Παράδειγμα 9](#)).

Ένα composition μπορεί να παραχθεί μόνο από ένα αρχικό composition και να ενημερωθεί μέσω recomposition. Ο μόνος τρόπος για να τροποποιηθεί ένα composition είναι μέσω recomposition.



Σχήμα 5. Το Lifecycle ενός composable σε ένα Composition.

2.3.7 ViewModel

Το ViewModel εκθέτει την κατάσταση στο UI και ενσωματώνει τη σχετική επιχειρηματική λογική. Το κύριο πλεονέκτημά του είναι ότι αποθηκεύει προσωρινά την κατάσταση και τη διατηρεί κατά τη διάρκεια αλλαγών στη διαμόρφωση. Αυτό σημαίνει ότι το UI δεν χρειάζεται να ανακτά δεδομένα ξανά κατά την πλοήγηση μεταξύ δραστηριοτήτων ή μετά από αλλαγές στη διαμόρφωση, όπως κατά την περιστροφή της οθόνης.

Η εναλλακτική λύση σε ένα ViewModel είναι μια απλή κλάση που περιέχει τα δεδομένα που εμφανίζονται στο UI. Αυτό μπορεί να δημιουργήσει πρόβλημα κατά την πλοήγηση μεταξύ δραστηριοτήτων ή προορισμών πλοήγησης. Με αυτόν τον τρόπο καταστρέφονται αυτά τα δεδομένα εάν δεν αποθηκευτούν χρησιμοποιώντας τον μηχανισμό κατάστασης αποθηκευμένης παρουσίας (saved instance state). Το ViewModel παρέχει ένα βολικό API για διατήρηση δεδομένων που επιλύει αυτό το πρόβλημα.

Εναλλακτικά, το Compose προσφέρει δυνατότητες διατήρησης που επιτρέπουν στις απλές κλάσεις να επιβιώνουν από αλλαγές διαμόρφωσης χωρίς την πλήρη υποδομή ενός ViewModel. Ενώ και οι δύο μηχανισμοί βοηθούν στη διατήρηση κατάστασης, είναι

γενικά ασφαλέστερο να παρέχεται ένα ViewModel σε ένα διατηρημένο instance παρά το αντίστροφο, καθώς το Lifecycle τους και οι συμπεριφορές καθαρισμού τους διαφέρουν.

Τα βασικά οφέλη του ViewModel είναι ουσιαστικά δύο:

- Επιτρέπει την διατήρηση της κατάστασης του UI.
- Παρέχει δυνατότητες στις εφαρμογές για να αξιοποιηθούν επιχειρηματικά (business logic).

2.4 Android Studio

Το Android Studio είναι το επίσημο ολοκληρωμένο περιβάλλον ανάπτυξης (IDE) για την ανάπτυξη εφαρμογών Android. Βασισμένο στα ισχυρά εργαλεία επεξεργασίας κώδικα και προγραμματισμού από την IntelliJ IDEA, το Android Studio προσφέρει ακόμη περισσότερες δυνατότητες που βελτιώνουν την παραγωγικότητά κατά τη δημιουργία εφαρμογών Android, όπως:

- Ένα ευέλικτο σύστημα δημιουργίας βασισμένο στο Gradle.
- Έναν γρήγορο και πλούσιο σε δυνατότητες εξομοιωτή.
- Ένα ενοποιημένο περιβάλλον όπου μπορεί να γίνει ανάπτυξη εφαρμογών για όλες τις συσκευές Android.
- Ζωντανή επεξεργασία για ενημέρωση συνθέσιμων στοιχείων σε εξομοιωτές και φυσικές συσκευές σε πραγματικό χρόνο.
- Πρότυπα κώδικα και ενσωμάτωση GitHub για να βοηθήσουν στην δημιουργία κοινών δυνατοτήτων εφαρμογών και την εισαγωγή δειγμάτων κώδικα.
- Εκτεταμένα εργαλεία και πλαίσια δοκιμών.
- Εργαλεία Lint για τον εντοπισμό απόδοσης, χρηστικότητας, συμβατότητας εκδόσεων και άλλων προβλημάτων.
- Υποστήριξη C++ και NDK.

2.4.1 Project structure

Κάθε project στο Android Studio περιέχει μία ή περισσότερες ενότητες με αρχεία πηγαίου κώδικα και αρχεία πόρων. Οι τύποι ενότητων περιλαμβάνουν:

- Ενότητες εφαρμογών Android

- Ενότητες βιβλιοθήκης
- Ενότητες Google App Engine

2.4.2 Gradle build system

Το Android Studio χρησιμοποιεί το Gradle ως βάση του συστήματος δημιουργίας, με περισσότερες δυνατότητες ειδικά για Android που παρέχονται από το πρόσθετο Android Gradle. Αυτό το σύστημα δημιουργίας λειτουργεί ως ενσωματωμένο εργαλείο από το μενού του Android Studio και ανεξάρτητα από τη γραμμή εντολών. Μπορούν να χρησιμοποιηθούν οι λειτουργίες του συστήματος δημιουργίας για να υλοποιηθούν τα εξής:

- Προσαρμογή, διαμόρφωση και επέκταση της διαδικασίας δημιουργίας.
- Δημιουργία πολλών APK για την εφαρμογή με διαφορετικές λειτουργίες, χρησιμοποιώντας το ίδιο έργο και ενότητες.
- Επαναχρησιμοποίηση κώδικα και πόρους σε διάφορα σύνολα πηγαίου κώδικα.

Αξιοποιώντας την ευελιξία του Gradle, μπορούν να επιτευχθούν όλα αυτά χωρίς να τροποποιηθούν τα βασικά αρχεία πηγαίου κώδικα της εφαρμογής.

2.4.3 Build variants

Το σύστημα δημιουργίας μπορεί να βοηθήσει στη δημιουργία διαφορετικών εκδόσεων της ίδιας εφαρμογής από ένα μόνο έργο. Αυτό είναι χρήσιμο όταν υπάρχει μια δωρεάν και μια πληρωμένη έκδοση της εφαρμογής ή εάν απαιτείται η διανομή πολλών APK για διαφορετικές διαμορφώσεις συσκευών στο Google Play.

2.4.4 Multiple APK support

Η υποστήριξη πολλαπλών APK επιτρέπει την δημιουργία αποτελεσματικά πολλών APK με βάση την πυκνότητα οθόνης ή το ABI (Application Binary Interface) . Για παράδειγμα, μπορούν να δημιουργηθούν ξεχωριστά APK μιας εφαρμογής για τις πυκνότητες οθόνης hdpi και mdpi, θεωρώντας τα παράλληλα ως μία παραλλαγή και επιτρέποντάς τους να μοιράζονται τις ρυθμίσεις APK δοκιμής, javac, dx και ProGuard.

2.4.5 Resource shrinking

Η συρρίκνωση πόρων (resource shrinking) στο Android Studio καταργεί αυτόματα τους αχρησιμοποίητους πόρους από τις εξαρτήσεις της εφαρμογής και της βιβλιοθήκης. Για

παράδειγμα, εάν η εφαρμογή χρησιμοποιεί τις υπηρεσίες Google Play για πρόσβαση στις λειτουργίες του Google Drive και δεν χρησιμοποιείται αυτήν τη στιγμή το Google Sign-In, τότε η συρρίκνωση πόρων μπορεί να καταργήσει τα διάφορα στοιχεία σχεδίασης για τα κουμπιά SignInButton.

2.4.6 Manage dependencies

Οι εξαρτήσεις για το project καθορίζονται ονομαστικά στο σενάριο δημιουργίας σε επίπεδο ενότητας. Το Gradle βρίσκει εξαρτήσεις και τις καθιστά διαθέσιμες στην κατασκευή (build). Μπορεί να δηλωθούν εξαρτήσεις ενότητας, απομακρυσμένες δυαδικές εξαρτήσεις και τοπικές δυαδικές εξαρτήσεις στο αρχείο build.gradle.kts.

2.4.7 Annotations

Το Android Studio υποστηρίζει annotations για μεταβλητές, παραμέτρους και τιμές επιστροφής για να βοηθήσει στον εντοπισμό σφαλμάτων, όπως εξαιρέσεις null pointer και διενέξεις τύπων πόρων.

Ο Διαχειριστής SDK του Android συσκευάζει τη βιβλιοθήκη annotation Jetpack στο Android Support Repository για χρήση με το Android Studio. Το Android Studio επικυρώνει τα διαμορφωμένα annotation κατά την επιθεώρηση κώδικα.

2.4.8 Gemini

Το Gemini στο Android Studio είναι ο βοηθός στον προγραμματισμό με τεχνητή νοημοσύνη για την ανάπτυξη Android. Βοηθά τον προγραμματιστή να γίνεται πιο παραγωγικός απαντώντας στα ερωτήματά του για ανάπτυξη, δημιουργώντας κώδικα, βρίσκοντας σχετικούς πόρους και ενθαρρύνοντας τις βέλτιστες πρακτικές.

Το πιο σημαντικό στοιχείο αποτελεί το γεγονός ότι, το Gemini στο Android Studio είναι μοναδικά εξοπλισμένο για να επιταχύνει την ανάπτυξη Android και μπορεί να βοηθήσει στην δημιουργία μακετών και να αντιμετωπιστούν προβλήματα με τα περιβάλλοντα χρήστη Compose, να διορθωθούν σφάλματα δημιουργίας Gradle, να αναλυθούν σφάλματα μέσω ενσωματώσεων με το Logcat και το App Quality Insights και πολλά άλλα.

3. Cross - Platform Εφαρμογή Φυσικής Κατάστασης

Με την ανάπτυξη της Cross - Platform Εφαρμογής Φυσικής Κατάστασης επιδιώκεται η παρουσίαση των δυνατοτήτων της KMP. Για να επιτευχθεί ο σκοπός αυτός χρησιμοποιήθηκαν όλα τα εργαλεία που αναφέρθηκαν στο προηγούμενο κεφάλαιο καθώς ορισμένα ακόμη που θα αναλυθούν παρακάτω.

3.1 Λειτουργίες της εφαρμογής

Η αναπτυχθείσα εφαρμογή "FitnessApp" σχεδιάστηκε με γνώμονα την παροχή ενός ολοκληρωμένου, αυτόνομου και έξυπνου περιβάλλοντος διαχείρισης της σωματικής υγείας και της καθημερινής άσκησης. Οι βασικές λειτουργίες που ενσωματώνει το σύστημα και οι οποίες είναι καθολικά προσβάσιμες σε Android, iOS και Desktop, συνοψίζονται στις ακόλουθες ενότητες.

3.1.1 Σύστημα Αυθεντικοποίησης και Ασφάλειας (User Authentication)

Η εφαρμογή παρέχει έναν αυτόνομο μηχανισμό εισόδου και εγγραφής χρηστών (Sign In / Sign Up) μέσω της υπηρεσίας Supabase Auth. Διασφαλίζεται η προστασία των προσωπικών δεδομένων και η δημιουργία ενός μοναδικού αναγνωριστικού (User ID) για κάθε χρήστη, το οποίο αποτελεί τη βάση για την εξατομίκευση της πληροφορίας και τον μετέπειτα συγχρονισμό των δεδομένων στο cloud.

3.1.2 Ψηφιακός Βοηθός Τεχνητής Νοημοσύνης (PocketCoach AI)

Μέσω της απευθείας ενσωμάτωσης του Gemini API, η εφαρμογή προσφέρει μια διαδραστική ροή συνομιλίας (Chat Interface) με έναν εικονικό προπονητή φυσικής κατάστασης. Ο χρήστης μπορεί να υποβάλλει ερωτήσεις σε φυσική γλώσσα, να ζητά εξειδικευμένα προγράμματα εκγύμνασης (π.χ. ασκήσεις για συγκεκριμένες μυϊκές ομάδες) και να λαμβάνει άμεσες, δομημένες και επιστημονικά τεκμηριωμένες απαντήσεις.

3.1.3 Δυναμική Βιβλιοθήκη Ασκήσεων (Exercises Library)

Η εφαρμογή ενσωματώνει μια πλήρη βάση δεδομένων με ασκήσεις γυμναστικής. Παρέχεται η δυνατότητα φιλτραρίσματος των ασκήσεων με βάση τη δυσκολία (Difficulty), την κατηγορία (Category) και τη μυϊκή ομάδα στόχευσης (Muscle Group). Κάθε άσκηση συνοδεύεται από αναλυτικές οδηγίες σωστής τεχνικής (αρχική θέση, φάση

καθόδου και ανόδου) για την αποφυγή τραυματισμών, ενώ υποστηρίζεται και η εισαγωγή νέων custom ασκήσεων από τον χρήστη.

Η λειτουργία αυτή ενισχύεται σημαντικά μέσω της ενσωμάτωσης του ψηφιακού βοηθού PocketCoach AI, ο οποίος επιτρέπει στον χρήστη να παράγει αυτοματοποιημένα νέες, εξειδικευμένες ασκήσεις εισάγοντας απλώς μια γενική ιδέα εκγύμνασης. Το σύστημα, αξιοποιώντας προηγμένες τεχνικές Prompt Engineering, υποχρεώνει το Gemini API να επιστρέψει μια αυστηρά δομημένη απάντηση με έναν πρωτότυπο τίτλο και αναλυτικές step-by-step οδηγίες εκτέλεσης, τις οποίες ο χρήστης μπορεί να ελέγξει, να εμπλουτίσει με δικές του σημειώσεις και να αποθηκεύσει άμεσα στην προσωπική του συλλογή.

3.1.4 Υπολογισμός και Καταγραφή Δείκτη Μάζας Σώματος (BMI Calculator & History)

Η ενότητα αυτή επιτρέπει τον αυτόματο υπολογισμό του Δείκτη Μάζας Σώματος (BMI) και την κλινική κατηγοριοποίηση του χρήστη (π.χ. φυσιολογικό βάρος, υπέρβαρος). Η εφαρμογή αντλεί αυτόματα το αποθηκευμένο ύψος από το προφίλ για τη διευκόλυνση του χρήστη, ενώ προσφέρει ένα πλήρες ιστορικό προηγούμενων καταγραφών, επιτρέποντας τη διαχρονική παρακολούθηση της εξέλιξης του σωματικού βάρους.

3.1.5 Διαχείριση Προσωπικού Προφίλ (Profile Management)

Μέσα από μια native φόρμα εισαγωγής, ο χρήστης μπορεί να διαχειρίζεται και να επικαιροποιεί ανά πάσα στιγμή τα βασικά βιομετρικά του στοιχεία, όπως το ονοματεπώνυμο, το τρέχον ύψος και το σωματικό του βάρος. Τα στοιχεία αυτά αποθηκεύονται τοπικά και τροφοδοτούν τις υπόλοιπες λειτουργικές ενότητες της εφαρμογής.

3.1.6 Offline-First Λειτουργία και Αυτόματος Συγχρονισμός (Data Synchronization)

Όλες οι παραπάνω λειτουργίες (καταγραφή BMI, τροποποίηση προφίλ, διαχείριση ασκήσεων και πλάνων) εκτελούνται απρόσκοπτα ακόμα και σε πλήρη απουσία σύνδεσης διαδικτύου (Offline Mode). Τα δεδομένα αποθηκεύονται άμεσα στην τοπική βάση SQLDelight, και ο ενσωματωμένος μηχανισμός SyncManager αναλαμβάνει αυτόματα τον αμφίδρομο συγχρονισμό τους με το απομακρυσμένο cloud backend (Supabase) μόλις εντοπιστεί ενεργό δίκτυο.

3.2 Υλοποίηση της εφαρμογής

Για την παρουσίαση της υλοποίησης του κυρίως μέρους της εφαρμογής επιλέχθηκε η λειτουργία υπολογισμού του δείκτη μάζας σώματος (BMI).

3.2.1 Domain Layer

Το επίπεδο του domain (Domain Layer) ([Υπόδειγμα 1](#)) σχεδιάστηκε με γνώμονα την πλήρη απομόνωση της επιχειρηματικής λογικής (business logic) από τα επιμέρους περιβάλλοντα διεπαφής χρήστη. Κεντρικό στοιχείο αυτού του επιπέδου αποτελεί η περίπτωση χρήσης CalculateBMIUseCase, η οποία ενθυλακώνει τους κανόνες υπολογισμού του Δείκτη Μάζας Σώματος (BMI). Η συγκεκριμένη κλάση δέχεται ως παραμέτρους εισόδου το βάρος και το ύψος του χρήστη, εκτελεί τον μαθηματικό προσδιορισμό του δείκτη και μέσω της μεθόδου categorize, κατατάσσει το αποτέλεσμα στις προκαθορισμένες κλινικές κατηγορίες της δομής BmiCategory. Η επιστροφή των δεδομένων δομείται μέσω του αντικειμένου BmiScore, εξασφαλίζοντας τη μορφοποίηση του αποτελέσματος.

Λόγω της φύσης του KMP, ο συγκεκριμένος κώδικας στερείται πλατφορμικών εξαρτήσεων (platform-agnostic), γεγονός που επιτρέπει την αυτούσια επαναχρησιμοποίησή του σε Android, iOS και desktop διασφαλίζοντας την αρχιτεκτονική καθαρότητα και τη διευκόλυνση της διαδικασίας ελέγχου (unit testing).

3.2.2 Τοπική βάση δεδομένων (Data Layer)

Για την υλοποίηση του επιπέδου δεδομένων (Data Layer) της cross-platform εφαρμογής φυσικής κατάστασης, επιλέχθηκε η χρήση της βιβλιοθήκης SQLDelight ([Υπόδειγμα 2](#)). Η SQLDelight αποτελεί μια advanced λύση για τη διαχείριση τοπικών βάσεων δεδομένων SQLite σε περιβάλλον Kotlin Multiplatform (KMP).

Η βασική της ιδιαιτερότητα έγκειται στο ότι παράγει αυτόματα ασφαλείς κλάσεις Kotlin (typesafe Kotlin APIs) απευθείας από αρχεία καθαρής SQL. Με τον τρόπο αυτό, εκμηδενίζονται τα σφάλματα κατά το runtime και διασφαλίζεται η εγκυρότητα των ερωτημάτων (queries) κατά τη διαδικασία της μεταγλώττισης (compile-time). Η βιβλιοθήκη προσφέρει πλήρη διαμοιρασμό της λογικής των δεδομένων, επιτρέποντας στην εφαρμογή να εκτελεί τις ίδιες λειτουργίες αποθήκευσης τόσο στην πλατφόρμα

Android όσο και στα iOS και desktop, διατηρώντας την αρχιτεκτονική συνοχή του συστήματος.

Στο πλαίσιο της αποθήκευσης των δεδομένων φυσικής κατάστασης, δημιουργήθηκε ο πίνακας bmiResults στην τοπική βάση δεδομένων SQLite μέσω του αρχείου ρυθμίσεων της SQLDelight. Ο πίνακας αυτός υποστηρίζει πλήρως τη λειτουργικότητα εκτός σύνδεσης (offline-first) και τον συγχρονισμό δεδομένων.

Η δομή του πίνακα bmiResults περιλαμβάνει τα εξής πεδία:

- id: Ο μοναδικός αναγνωριστικός αριθμός (κλειδί) της εγγραφής (Primary Key), ο οποίος αυξάνεται αυτόματα (Autoincrement).
- user_id: Το αναγνωριστικό του χρήστη, επιτρέποντας τη διαχείριση δεδομένων σε περιβάλλον πολλών χρηστών.
- weight_kg & height_cm: Οι πρωτογενείς μετρήσεις βάρους (σε κιλά) και ύψους (σε εκατοστά).
- bmi_value & bmi_category: Η υπολογισμένη τιμή του Δείκτη Μάζας Σώματος και η αντίστοιχη κλινική κατηγορία.
- created_at: Χρονική σήμανση (Timestamp σε μορφή INTEGER) για την ακριβή καταγραφή της ημερομηνίας και ώρας της μέτρησης.
- is_deleted: Σημαία (flag) λογικής διαγραφής με προκαθορισμένη τιμή το μηδέν (DEFAULT 0).
- sync_state: Κείμενο που ορίζει την κατάσταση συγχρονισμού της εγγραφής με το backend.

Ορίζονται επίσης ερωτήματα SQL (SQL Queries) τα οποία η SQLDelight μετατρέπει απευθείας σε ασφαλείς μεθόδους της Kotlin όπως παρακάτω:

- getAll: Ανακτά το σύνολο των καταγραφών BMI, ταξινομημένο με φθίνουσα σειρά με βάση την ημερομηνία δημιουργίας (ORDER BY created_at DESC), ώστε ο χρήστης να βλέπει πρώτες τις πρόσφατες μετρήσεις του.
- insertResult: Πραγματοποιεί την εισαγωγή των νέων μετρήσεων στον πίνακα, δεχόμενη τις απαραίτητες παραμέτρους.
- selectPending: Αποτελεί βασικό ερώτημα για τον μηχανισμό συγχρονισμού. Φιλτράρει και επιστρέφει όλες τις εγγραφές των οποίων η κατάσταση δεν είναι 'SYNCED', υποδεικνύοντας τις αλλαγές που πρέπει να αποσταλούν στον server.

- `selectById`: Επιτρέπει την απομονωμένη αναζήτηση και ανάκτηση μιας συγκεκριμένης καταγραφής με βάση το μοναδικό της αναγνωριστικό.
- `markSynced`: Ενημερώνει το πεδίο `sync_state` μιας εγγραφής, σηματοδοτώντας την επιτυχή ολοκλήρωση του συγχρονισμού με το απομακρυσμένο API.
- `softDeleteBmi`: Υλοποιεί τη λογική διαγραφής. Αντί να διαγράψει οριστικά την εγγραφή, θέτει τη σημαία `is_deleted = 1` και αλλάζει το `sync_state`. Έτσι, η εγγραφή αποκρύπτεται από τον χρήστη αλλά παραμένει στη βάση μέχρι να ενημερωθεί ο server για τη διαγραφή.
- `deleteBmi`: Εκτελεί τη φυσική και οριστική διαγραφή (hard delete) της εγγραφής από την τοπική βάση δεδομένων.

Για τη διαχείριση των δεδομένων του Δείκτη Μάζας Σώματος (BMI) και την αποσύνδεση της επιχειρηματικής λογικής από τις πηγές δεδομένων, εφαρμόστηκε το αρχιτεκτονικό πρότυπο Repository Pattern. Η προσέγγιση αυτή υλοποιείται μέσω της διεπαφής (interface) `BmiRepository` στο επίπεδο του domain και της συγκεκριμένης κλάσης υλοποίησης `RealBmiRepository` στο επίπεδο των δεδομένων (Data Layer).

Διεπαφή BmiRepository

Η διεπαφή `BmiRepository` ορίζει το συμβόλαιο (contract) για όλες τις απαιτούμενες λειτουργίες διαχείρισης των αποτελεσμάτων BMI. Περιλαμβάνει μεθόδους για:

- Ασύγχρονη ανάκτηση όλων των καταγραφών.
- Εισαγωγή νέων αποτελεσμάτων στη βάση.
- Συγχρονισμό δεδομένων με απομακρυσμένο διακομιστή.
- Λογική και φυσική διαγραφή εγγραφών.

Κλάση Υλοποίησης RealBmiRepository

Η κλάση `RealBmiRepository` αποτελεί την πραγματική υλοποίηση της διεπαφής, χρησιμοποιώντας τη βάση δεδομένων `FitnessDatabase` που δημιουργήθηκε μέσω της βιβλιοθήκης `SQLDelight`. Η επικοινωνία με τη βάση γίνεται μέσω του αντικειμένου `queries (bmiResultsQueries)`, το οποίο παράγεται αυτόματα από την `SQLDelight`. Υπόδειγμα δημιουργίας της βάσης δεδομένων `FitnessDatabase` που δημιουργήθηκε στο `build.gradle.kts (:composeApp)` παρουσιάζεται στο [Υπόδειγμα 3](#) του Παραρτήματος «B».

Στην κλάση αυτή ενσωματώνονται προηγμένες τεχνολογίες της Kotlin όπως παρακάτω:

- Kotlin Coroutines & Dispatchers: Οι λειτουργίες εγγραφής, τροποποίησης και διαγραφής εκτελούνται ασύγχρονα, χωρίς να μπλοκάρουν το κύριο νήμα (main thread) της εφαρμογής.
- Kotlin Flows: Η μέθοδος `getAll()` επιστρέφει μια ροή δεδομένων (`Flow<List<BmiResult>>`). Χρησιμοποιώντας τις επεκτάσεις `asFlow()` και `mapToList(Dispatchers.Default)` της `SQLDelight`, η εφαρμογή αποκτά reactive ιδιότητες. Κάθε αλλαγή στη βάση δεδομένων πυροδοτεί αυτόματα την ανανέωση της διεπαφής χρήστη (UI) σε πραγματικό χρόνο.
- Data Mapping: Χρησιμοποιούνται mappers (`toDomain()`) για τη μετατροπή των οντοτήτων της βάσης δεδομένων σε καθαρά μοντέλα του domain (`BmiResult`), διατηρώντας τα επίπεδα της αρχιτεκτονικής απομονωμένα.

Υλοποίηση του κώδικα της διεπαφής `BmiRepository` και της κλάσης υλοποίησης `RealBmiRepository` φαίνονται στα [Υποδείγματα 5 έως 10](#) του Παραρτήματος «B».

Μηχανισμός Συγχρονισμού και Διαχείρισης Κατάστασης (Sync State)

Ένα κρίσιμο χαρακτηριστικό της cross-platform εφαρμογής είναι η υποστήριξη λειτουργίας εκτός σύνδεσης (offline support) και ο μετέπειτα συγχρονισμός με το backend. Για τον σκοπό αυτό, χρησιμοποιείται η δομή `SyncState` με την παρακάτω λογική:

- `markSynced`: Ενημερώνει τη βάση ότι η εγγραφή ανέβηκε επιτυχώς στο cloud (`SyncState.SYNCED`).
- `softDeleteBmi` (Λογική Διαγραφή): Αντί για την άμεση διαγραφή μιας εγγραφής όταν ο χρήστης είναι offline, η εγγραφή σημαίνεται ως προς διαγραφή (`SyncState.PENDING_DELETE`). Με αυτόν τον τρόπο, το σύστημα γνωρίζει ότι πρέπει να διαγράψει την εγγραφή και από τον server μόλις αποκατασταθεί η σύνδεση.
- `deleteBmi` (Φυσική Διαγραφή): Εκτελεί την οριστική διαγραφή της εγγραφής από την τοπική βάση δεδομένων `SQLite`.

Διαχείριση Οδηγών Βάσης Δεδομένων (SQL Drivers)

Η βιβλιοθήκη `SQLDelight` βασίζεται σε μια split-architecture προσέγγιση: διαχωρίζει τη λογική ορισμού της βάσης δεδομένων από την πλατφορμική υλοποίηση του οδηγού (driver) που εκτελεί τα queries. Για την επίτευξη αυτού του στόχου σε περιβάλλον Kotlin

Multiplatform (KMP) με υποστήριξη Android, iOS και Desktop εφαρμογών, χρησιμοποιήθηκε το σχεδιαστικό πρότυπο Factory Pattern σε συνδυασμό με τον μηχανισμό Expect/Actual της Kotlin.

Ο Μηχανισμός expect / actual

Στο κοινόχρηστο επίπεδο κώδικα (Common SourceSet), ορίζεται η κλάση expect class DatabaseDriverFactory. Αυτή λειτουργεί ως ένα συμβόλαιο, δηλώνοντας ότι κάθε πλατφόρμα υποχρεούται να επιστρέψει έναν συμβατό SqlDriver μέσω της μεθόδου createDriver(). Η εξειδίκευση (implementation) γίνεται ξεχωριστά για κάθε λειτουργικό σύστημα όπως παρακάτω:

- Android (AndroidSqliteDriver): Απαιτεί το Context της εφαρμογής για να αρχικοποιήσει το αρχείο fitness.db μέσω του εγγενούς Android SQLite framework.
- iOS (NativeSqliteDriver): Χρησιμοποιεί την native υλοποίηση της SQLite για Apple πλατφόρμες. Δεν απαιτεί επιπλέον παραμέτρους (όπως context), καθώς αποκτά άμεση πρόσβαση στον τοπικό κατάλογο εγγράφων του app sandbox.
- Desktop/JVM (JdbcSqliteDriver): Χρησιμοποιεί τον οδηγό JDBC (sqlite-jdbc). Στη συγκεκριμένη πλατφόρμα, η βάση αποθηκεύεται στον αρχικό κατάλογο του χρήστη (user.home) ως FitnessDatabase.db.

Έλεγχος Ύπαρξης και Δημιουργία Σχήματος στο Desktop

Μια κρίσιμη ιδιαιτερότητα του Desktop driver είναι η χειροκίνητη διαχείριση του κύκλου ζωής του σχήματος της βάσης (Schema Lifecycle). Ενώ στο Android και το iOS η SQLDelight αναλαμβάνει αυτόματα τη δημιουργία των πινάκων, στον JDBC driver του Desktop αυτό πρέπει να γίνει ρητά.

Όπως φαίνεται στον κώδικα, ελέγχεται αν το αρχείο δεν υπήρχε πριν από τη σύνδεση (!databasePath.exists()). Αν η βάση δεδομένων είναι νέα (isNewDatabase), εκτελείται η μέθοδος FitnessDatabase.Schema.create(driver). Αυτή η διαδικασία διασφαλίζει ότι οι πίνακες SQL θα δημιουργηθούν μόνο κατά την πρώτη εκκίνηση της εφαρμογής, αποφεύγοντας σφάλματα διπλότυπης δημιουργίας σε επόμενες εκτελέσεις.

Κλάση Ενοποίησης DatabaseFactory

Η κλάση DatabaseFactory λειτουργεί ως ο κεντρικός ενορχηστρωτής (orchestrator). Δέχεται ως εξάρτηση (dependency injection) την κατάλληλη πλατφορμική υλοποίηση του

DatabaseDriverFactory. Μέσω της μεθόδου `createDatabase()`, ανακτά τον αντίστοιχο `SqlDriver` και τον μεταβιβάζει στον κατασκευαστή της `FitnessDatabase` ([Υπόδειγμα 4](#)).

Με αυτόν τον τρόπο, ολόκληρο το υπόλοιπο `Data Layer` της εφαρμογής (όπως τα `Repositories`) αλληλεπιδρά με μια ενιαία, αφηρημένη οντότητα βάσης δεδομένων, αγνοώντας πλήρως αν η εφαρμογή εκτελείται σε κινητό τηλέφωνο `Android`, σε `iPhone` ή σε υπολογιστή `Desktop`.

Επίσημη βιβλιοθήκη της εταιρείας `Google` για τη διαχείριση της εσωτερικής μνήμης της συσκευής σε `android` μέσω της γλώσσας προγραμματισμού `Kotlin` είναι η βιβλιοθήκη `Room`. Παρακάτω αναλύεται ο λόγος για τον οποίο επιλέχθηκε η `SQLDelight`.

Δυνατότητες και Αρχιτεκτονική της SQLDelight

Η βιβλιοθήκη `SQLDelight` (αναπτύσσεται από την εταιρεία `Cash App`) και προσεγγίζει τη διαχείριση δεδομένων με τη φιλοσοφία "`SQL-First`". Αντί να προσπαθεί να κρύψει τη βάση δεδομένων πίσω από κλάσεις, θέτει την ίδια την `SQL` στο επίκεντρο. Έχει επίσης τα παρακάτω χαρακτηριστικά:

- **Μεταγλώττιση `SQL` σε Κώδικα (`Compile-time Code Generation`):** Ο προγραμματιστής γράφει αρχεία με καθαρό κώδικα `SQL` (`.sq`). Κατά τη διαδικασία του `build`, η `SQLDelight` αναλύει τις εντολές και παράγει αυτόματα `typesafe` κλάσεις και μεθόδους σε `Kotlin`.
- **Ασφάλεια Τύπων (`Type Safety`):** Οποιοδήποτε λάθος σε όνομα πίνακα, στήλης ή σύνταξης (π.χ. ένα ξεχασμένο κόμμα) εντοπίζεται αμέσως κατά το `compile-time`. Αυτό εκμηδενίζει τα `runtime` κρυσμαρίσματα λόγω κακών ερωτημάτων.
- **Πλήρης Υποστήριξη `Multiplatform` (`KMP`):** Είναι σχεδιασμένη εξ αρχής για να διαμοιράζεται. Λειτουργεί ως κοινός κώδικας (`Common SourceSet`) και συνδέεται με διαφορετικούς οδηγούς (`Android SQLite`, `Native CoreData/SQLite`, `JDBC` για `Desktop/JVM`).
- **`Reactive Streams`:** Ενσωματώνει επεκτάσεις για `Kotlin Flows`, επιτρέποντας την αυτόματη ενημέρωση του `UI` μόλις τροποποιηθούν τα δεδομένα στη βάση.

Δυνατότητες και Αρχιτεκτονική της βιβλιοθήκης Room

Το Room αποτελεί την επίσημη βιβλιοθήκη αντιστοίχισης αντικειμένων (ORM - Object-Relational Mapping) της Google για το Android, ως μέρος των Android Jetpack αρχιτεκτονικών στοιχείων.

- Προσέγγιση «Annotation-First»: Η λειτουργία του βασίζεται σε annotations (όπως @Entity, @Dao, @Query, @Insert). Ο προγραμματιστής ορίζει κλάσεις της Kotlin και το Room αναλαμβάνει να τις χαρτογραφήσει σε πίνακες της SQLite.
- Αφηρημένο Επίπεδο (Abstraction Layer): Κρύβει το μεγαλύτερο μέρος της πολυπλοκότητας της SQLite. Για απλές λειτουργίες (εισαγωγή, διαγραφή, ενημέρωση), δεν απαιτείται η συγγραφή κώδικα SQL, καθώς παράγεται αυτόματα από το Room.
- Ενσωμάτωση στο Οικοσύστημα της Google: Συνεργάζεται άψογα με άλλα Jetpack components (όπως Paging Library, LiveData, ViewModel), καθιστώντας το την standard επιλογή για native Android εφαρμογές.
- Πρόσφατη Υποστήριξη KMP: Παρόλο που το Room ξεκίνησε ως αποκλειστικά Android βιβλιοθήκη, οι πρόσφατες εκδόσεις του έχουν επεκταθεί για να υποστηρίξουν Kotlin Multiplatform, ωστόσο η μετάβαση αυτή κουβαλάει ακόμα την κληρονομιά του Android σχεδιασμού του.

Επισημαίνεται ότι κατά τη έναρξη ανάπτυξης της εφαρμογής φυσικής κατάστασης το Room δεν ήταν διαθέσιμο για χρήση σε KMP.

Γιατί επιλέχθηκε SQLDelight

Για τις ανάγκες μιας cross-platform εφαρμογής φυσικής κατάστασης που αναπτύσσεται μέσω Kotlin Multiplatform (KMP) και στοχεύει σε Android, iOS και Desktop, η επιλογή του SQLDelight έναντι του Room κρίνεται ως η πλέον κατάλληλη για τους εξής επιστημονικούς και αρχιτεκτονικούς λόγους:

- Αρχιτεκτονική ωριμότητα στο KMP: Το SQLDelight δημιουργήθηκε με σκοπό την πολυ-πλατφορμική ανάπτυξη. Η υποστήριξη του Room για iOS και Desktop είναι μεταγενέστερη και συχνά παρουσιάζει περιορισμούς, ενώ το SQLDelight διαθέτει σταθερούς και δοκιμασμένους drivers για Native SQLite (iOS) και JDBC (Desktop).

- **Αρχιτεκτονική Καθαρότητα (Clean Architecture):** Το SQLDelight επιτρέπει στο Domain Layer να παραμένει εντελώς καθαρό από annotations τρίτων βιβλιοθηκών. Στο Room, οι οντότητες γεμίζουν με annotations (@Entity, @PrimaryKey), γεγονός που "μολύνει" τα μοντέλα με εξαρτήσεις του framework. Με το SQLDelight, οι πίνακες ορίζονται σε ξεχωριστά αρχεία SQL, διατηρώντας τον κώδικα της Kotlin απομονωμένο.
- **Πλήρης Έλεγχος του Μηχανισμού Συγχρονισμού (Sync State):** Όπως αναλύθηκε στην ενότητα της υλοποίησης, η εφαρμογή απαιτεί σύνθετη διαχείριση για offline-first λειτουργία (soft deletes, έλεγχος μη συγχρονισμένων εγγραφών με selectPending). Η συγγραφή αυτών των εξειδικευμένων queries σε καθαρή SQL προσφέρει απόλυτο έλεγχο στην απόδοση και τη συμπεριφορά της βάσης, κάτι που στα ORM frameworks όπως το Room μπορεί να καταλήξει σε πολύπλοκο κώδικα.
- **Μηδενικό Runtime Overhead:** Επειδή η SQLDelight μετατρέπει την SQL σε Kotlin κατά το compile-time, δεν υπάρχει καμία ανάκτηση (reflection) ή επεξεργασία annotations κατά την εκτέλεση της εφαρμογής (runtime). Αυτό μεταφράζεται σε ταχύτερη εκτέλεση των queries και χαμηλότερη κατανάλωση πόρων, στοιχείο κρίσιμο για mobile και desktop συσκευές.

3.2.3 Χρήση Backend-as-a-Service (BaaS) και Απομακρυσμένων Βάσεων Δεδομένων (Network Layer)

Στο πλαίσιο της παρούσας διπλωματικής εργασίας, για την υποστήριξη του οπίσθιου συστήματος (backend) και τη διαχείριση της απομακρυσμένης βάσης δεδομένων της εφαρμογής φυσικής κατάστασης, επιλέχθηκε η πλατφόρμα Supabase.

Η Πλατφόρμα Supabase

Το Supabase αποτελεί μια σύγχρονη open-source (ανοιχτού κώδικα) σουίτα εργαλείων Backend-as-a-Service (BaaS), η οποία προτείνεται στη διεθνή βιβλιογραφία ως η κυριότερη εναλλακτική λύση έναντι ιδιόκτητων (proprietary) πλατφορμών, όπως το Google Firebase. Η αρχιτεκτονική του Supabase εδράζεται στην τεχνολογία του cloud computing, επιτρέποντας την ταχεία ανάπτυξη εφαρμογών πολλαπλών πλατφορμών (Cross-Platform) χωρίς την ανάγκη χειροκίνητης παραμετροποίησης και συντήρησης ιδιόκτητων διακομιστών (serverless architecture).

Αρχιτεκτονική και Σχεσιακό Μοντέλο Δεδομένων (PostgreSQL)

Στον πυρήνα του Supabase βρίσκεται το Σύστημα Διαχείρισης Σχεσιακών Βάσεων Δεδομένων (RDBMS) PostgreSQL. Η επιλογή αυτή προσφέρει σημαντικά πλεονεκτήματα στην παρούσα εφαρμογή fitness:

- Ακεραιότητα Δεδομένων (Data Integrity): Σε αντίθεση με τις NoSQL βάσεις δεδομένων, η PostgreSQL επιβάλλει αυστηρούς κανόνες και περιορισμούς (constraints). Αυτό εξασφαλίζει ότι τα δεδομένα των χρηστών (π.χ. μετρήσεις σωματικού βάρους, ιστορικό προπονήσεων) παραμένουν έγκυρα και συνεπή.
- Σύνθετες Ερωτήσεις (Complex Queries): Επιτρέπει την εκτέλεση πολύπλοκων συσχετίσεων (JOINS) μεταξύ πινάκων, διευκολύνοντας την εξαγωγή στατιστικών στοιχείων.

Δυνατότητες και Τεχνολογικά Χαρακτηριστικά του Supabase

Η πλατφόρμα ενσωματώνει μια σειρά από κρίσιμες λειτουργίες για τη λειτουργία της εφαρμογής, οι οποίες προσφέρονται μέσω ενοποιημένων APIs:

1. Σύστημα Ταυτοποίησης και Αυθεντικοποίησης (Supabase Auth): Παρέχει έναν πλήρη μηχανισμό διαχείρισης χρηστών. Υποστηρίζει την ασφαλή εγγραφή και σύνδεση μέσω παραδοσιακών μεθόδων (Email/Password) καθώς και τη δυνατότητα ταυτοποίησης μέσω τρίτων παρόχων (OAuth), διασφαλίζοντας τα προσωπικά δεδομένα με σύγχρονα πρωτόκολλα κρυπτογράφησης.
2. Συγχρονισμός σε Πραγματικό Χρόνο (Realtime Engine): Μέσω της τεχνολογίας WebSockets, το Supabase επιτρέπει την ακρόαση (listening) αλλαγών στη βάση δεδομένων. Οποιοδήποτε update ή insert ανιχνευθεί, προωθείται άμεσα από και προς την cross-platform εφαρμογή, επιτρέποντας τη δυναμική ανανέωση της διεπαφής (UI) χωρίς την ανάγκη χειροκίνητης επαναφόρτωσης (polling).
3. Αποθήκευση Αντικειμένων (Supabase Storage): Παρέχει τη δυνατότητα διαχείρισης και αποθήκευσης μη δομημένων δεδομένων μεγάλου όγκου (Binary Large Objects - BLOBs).
4. Υπολογιστικό Νέφος (Edge Functions): Επιτρέπει την εκτέλεση κώδικα TypeScript σε περιβάλλον serverless, εγγύτερα στον τελικό χρήστη. Με τον τρόπο αυτό, μπορούν να υλοποιηθούν επιχειρηματική λογική (business logic) και

εξωτερικές ενσωματώσεις (integrations) που απαιτούν αυξημένη ασφάλεια, απομονωμένες από τη συσκευή του πελάτη (client).

Στο πλαίσιο του Kotlin Multiplatform (KMP), η επικοινωνία με το Supabase επιτυγχάνεται μέσω των επίσημων κοινόχρηστων βιβλιοθηκών (KMP SDKs), διασφαλίζοντας ότι η λογική πρόσβασης στα δεδομένα γράφεται μία φορά και εκτελείται απρόσκοπτα τόσο στο περιβάλλον Android όσο και στην Desktop έκδοση της εφαρμογής.

Πολιτικές Ασφάλειας Επιπέδου Γραμμής (Row Level Security - RLS Policies)

Ένα από τα σημαντικότερα χαρακτηριστικά του Supabase, το οποίο διασφαλίζει τη συμμόρφωση της εφαρμογής με τους σύγχρονους κανονισμούς προστασίας προσωπικών δεδομένων (όπως ο GDPR), είναι η ενσωμάτωση του μηχανισμού Row Level Security (RLS) της PostgreSQL. Σε παραδοσιακές αρχιτεκτονικές, ο έλεγχος πρόσβασης στα δεδομένα (authorization) υλοποιείται σε επίπεδο κώδικα στο backend API. Αντίθετα, με τη χρήση του RLS, οι κανόνες ασφαλείας μεταφέρονται απευθείας στο επίπεδο της βάσης δεδομένων. Αυτό σημαίνει ότι κάθε πίνακας (table) μπορεί να θωρακιστεί με συγκεκριμένες πολιτικές (policies), οι οποίες καθορίζουν με ακρίβεια ποιος χρήστης έχει δικαίωμα να εκτελέσει λειτουργίες CRUD (Create, Read, Update, Delete) σε κάθε μεμονωμένη γραμμή (row) του πίνακα.

Στην αναπτυχθείσα εφαρμογή fitness, ο μηχανισμός RLS αξιοποιείται ως εξής:

- Απομόνωση Δεδομένων Χρήστη (User Isolation): Εφαρμόζονται RLS policies που επιτρέπουν την ανάγνωση ή τροποποίηση δεδομένων (π.χ. οι καταγραφές των ασκήσεων) αποκλειστικά και μόνο από τον αυθεντικοποιημένο χρήστη στον οποίο ανήκουν. Αυτό επιτυγχάνεται με τη χρήση της ενσωματωμένης συνάρτησης `auth.uid()`, η οποία διασταυρώνει το αναγνωριστικό του χρήστη που κάνει το αίτημα με την αντίστοιχη ξένη κλειδα (foreign key) του πίνακα.
- Πρόσβαση σε Δημόσια Δεδομένα (Public/Global Data): Για πίνακες που περιέχουν γενικές πληροφορίες, όπως ο κατάλογος με τις διαθέσιμες ασκήσεις, οι πολιτικές RLS διαμορφώνονται έτσι ώστε να επιτρέπουν την καθολική ανάγνωση (SELECT) από όλους τους εγγεγραμμένους χρήστες, απαγορεύοντας παράλληλα οποιαδήποτε δικαίωμα εγγραφής ή τροποποίησης (INSERT, UPDATE, DELETE) από την πλευρά του client.

- Προστασία από Παρακάμψεις (Client-Side Tampering): Επειδή ο cross-platform client (Android, iOS, Desktop) επικοινωνεί απευθείας με το API του Supabase, η ενεργοποίηση του RLS διασφαλίζει ότι ακόμα και αν ένας κακόβουλος χρήστης προσπαθήσει να τροποποιήσει τον κώδικα της εφαρμογής ή να υποκλέψει τα API keys, η βάση δεδομένων θα απορρίψει αυτόματα οποιοδήποτε αίτημα δεν πληροί τις προϋποθέσεις του εκάστοτε policy.

Συγκριτική Αξιολόγηση: Supabase έναντι Google Firebase

Κατά τη φάση του αρχιτεκτονικού σχεδιασμού της εφαρμογής, πραγματοποιήθηκε συγκριτική μελέτη μεταξύ των δύο επικρατέστερων λύσεων Backend-as-a-Service (BaaS): του Google Firebase και του Supabase. Παρόλο που το Firebase αποτελεί μια ώριμη και ευρέως διαδεδομένη πλατφόρμα, επιλέχθηκε τελικά το Supabase για την υλοποίηση της παρούσας διπλωματικής εργασίας. Η απόφαση αυτή βασίστηκε σε συγκεκριμένα τεχνικά, αρχιτεκτονικά και οικονομικά κριτήρια που αναλύονται παρακάτω:

Σχεσιακό Μοντέλο (SQL) έναντι Μη Σχεσιακού (NoSQL)

Η θεμελιώδης διαφορά μεταξύ των δύο πλατφορμών έγκειται στο μοντέλο διαχείρισης των δεδομένων:

- Firebase (Firestore / Realtime Database): Βασίζεται σε αρχιτεκτονική NoSQL (Document-based ή JSON tree). Αν και προσφέρει υψηλή οριζόντια κλιμάκωση (horizontal scaling), παρουσιάζει σημαντικούς περιορισμούς στην υλοποίηση σύνθετων επιχειρηματικών κανόνων, καθώς δεν υποστηρίζει native συσχετίσεις (JOINS) μεταξύ διαφορετικών συλλογών δεδομένων (collections). Αυτό οδηγεί συχνά σε πλεονασμό δεδομένων (data denormalization) και αυξάνει την πολυπλοκότητα του κώδικα στον client.
- Supabase (PostgreSQL): Ενσωματώνει μια πλήρη σχεσιακή βάση δεδομένων (RDBMS). Για μια εφαρμογή φυσικής κατάστασης, όπου τα δεδομένα είναι εξαιρετικά δομημένα και αλληλένδετα (π.χ. ένας χρήστης εκτελεί μια συγκεκριμένη άσκηση, η οποία ανήκει σε ένα πρόγραμμα εκγύμνασης, το οποίο με τη σειρά του περιλαμβάνει συγκεκριμένα σετ και επαναλήψεις), η χρήση της SQL είναι αρχιτεκτονικά ορθότερη. Διασφαλίζει την απόλυτη ακεραιότητα των δεδομένων μέσω ξένων κλειδιών (foreign keys) και επιτρέπει την εκτέλεση

σύνθετων αναλυτικών ερωτημάτων για την πρόοδο του χρήστη χωρίς να επιβαρύνεται η συσκευή (client).

Ανοιχτό Λογισμικό (Open-Source) και Αποφυγή Εγκλωβισμού (Vendor Lock-in)

- **Firestore:** Αποτελεί ένα ιδιόκτητο (proprietary) προϊόν της Google. Η χρήση του συνεπάγεται τον πλήρη εγκλωβισμό του δημιουργού στο οικοσύστημα της συγκεκριμένης εταιρείας (vendor lock-in). Η μεταφορά της εφαρμογής σε άλλη υποδομή (migration) σε μεταγενέστερο στάδιο είναι εξαιρετικά δύσκολη και δαπανηρή, καθώς ο κώδικας και η δομή των δεδομένων είναι άρρηκτα συνδεδεμένα με τα API της Google.
- **Supabase:** Είναι μια open-source πλατφόρμα που βασίζεται σε ευρέως αποδεκτά πρότυπα του κλάδου (PostgreSQL, GoTrue, PostgREST). Αυτό προσφέρει μέγιστη ευελιξία, καθώς η εφαρμογή μπορεί ανά πάσα στιγμή να μεταφερθεί εκτός των διακομιστών του Supabase. Η βάση δεδομένων μπορεί να γίνει self-hosted σε οποιονδήποτε ιδιόκτητο server ή cloud πάροχο (π.χ. AWS, DigitalOcean) χωρίς να απαιτείται η επανασυγγραφή του κώδικα της εφαρμογής.

Μηχανισμός Ασφάλειας και Διαχείρισης API

- **Firestore:** Η ασφάλεια των δεδομένων βασίζεται στους κανόνες Firestore Security Rules. Πρόκειται για μια ιδιοταγή γλώσσα κειμένου που απαιτεί εξειδικευμένη εκμάθηση, ενώ η διαχείριση της ασφάλειας γίνεται αποκομμένη από το επίπεδο της ίδιας της βάσης.
- **Supabase:** Αξιοποιεί το πρότυπο Row Level Security (RLS) της PostgreSQL. Η ασφάλεια ορίζεται με τη χρήση καθαρά SQL εκφράσεων, ενσωματώνεται άμεσα στους πίνακες και εκτελείται στο χαμηλότερο δυνατό επίπεδο. Επιπλέον, το Supabase μετατρέπει αυτόματα το σχήμα της βάσης δεδομένων σε ένα πλήρες RESTful API μέσω του PostgREST, προσφέροντας ταχύτερη και πιο τυποποιημένη επικοινωνία με τον Kotlin Multiplatform client.

Προβλεψιμότητα Κόστους (Pricing Model)

- **Firestore:** Το μοντέλο τιμολόγησης βασίζεται αποκλειστικά στον όγκο των λειτουργιών ανάγνωσης, εγγραφής και διαγραφής (pay-per-operation). Σε εφαρμογές με πολλούς χρήστες ή σε περιπτώσεις κακού σχεδιασμού των

ερωτημάτων από τον client, το κόστος μπορεί να αυξηθεί απρόβλεπτα και εκθετικά.

- Supabase: Η τιμολόγηση βασίζεται κυρίως στον συνολικό όγκο αποθήκευσης των δεδομένων (database size) και όχι στον αριθμό των μεμονωμένων ερωτημάτων API (API calls). Αυτό καθιστά το κόστος της υποδομής εξαιρετικά προβλέψιμο, στοιχείο κρίσιμο για τη βιωσιμότητα και την κλιμάκωση (scalability) του project.

Περιορισμοί Υποστήριξης Πλατφορμών (Platform Targets) στο KMP

Ένας καθοριστικός παράγοντας για την απόρριψη του Google Firebase στην παρούσα διατριβή ήταν η έλλειψη επίσημης και πλήρους υποστήριξης για περιβάλλοντα Desktop (JVM) εντός του πλαισίου Kotlin Multiplatform.

Αντίθετα, το Supabase προσεγγίζει την αρχιτεκτονική cross-platform με διαφορετική φιλοσοφία. Επειδή βασίζεται σε καθολικά HTTP/REST APIs και WebSockets (μέσω του PostgREST και των GoTrue/Realtime servers), οι επίσημες βιβλιοθήκες του για το Kotlin Multiplatform (Supabase KMP SDKs) λειτουργούν 100% ισοδύναμα (feature-parity) σε όλες τις πλατφόρμες-στόχους.

Ακολουθεί η αναλυτική παρουσίαση της δομής δεδομένων για τον υπολογισμό και την καταγραφή του Δείκτη Μάζας Σώματος (BMI). Το κείμενο είναι δομημένο με τρόπο που αναδεικνύει την αρχιτεκτονική προσέγγιση του Offline-First, τη διαχείριση της ασφάλειας μέσω RLS, καθώς και τον μηχανισμό συγχρονισμού μεταξύ της τοπικής βάσης (SQLDelight) και της απομακρυσμένης βάσης (Supabase/PostgreSQL).

Μοντελοποίηση Δεδομένων και Διαχείριση Ιστορικού BMI

Για την παρακολούθηση της φυσικής κατάστασης των χρηστών, η εφαρμογή απαιτεί την καταγραφή και αποθήκευση των μετρήσεων του BMI. Για την υλοποίηση αυτής της λειτουργίας, υιοθετήθηκε μια Offline-First αρχιτεκτονική. Η προσέγγιση αυτή διασφαλίζει ότι ο χρήστης μπορεί να χρησιμοποιεί την εφαρμογή και να καταγράφει τις μετρήσεις του απρόσκοπτα, ακόμη και σε συνθήκες απουσίας δικτύου, ενώ τα δεδομένα συγχρονίζονται αυτόματα στο cloud μόλις αποκατασταθεί η σύνδεση.

Για τον σκοπό αυτό, σχεδιάστηκαν δύο επίπεδα αποθήκευσης:

- Το Απομακρυσμένο Επίπεδο (Remote Layer - Supabase/PostgreSQL): Λειτουργεί ως η κεντρική πηγή αλήθειας (Central Source of Truth) και διασφαλίζει την επιμονή των δεδομένων σε όλες τις συσκευές του χρήστη.
- Το Τοπικό Επίπεδο (Local Layer - SQLDelight/SQLite): Λειτουργεί εντός της cross-platform εφαρμογής (Android, iOS, Desktop), προσφέροντας άμεση απόκριση (low latency) και offline λειτουργικότητα.

Στο περιβάλλον του Supabase, ο πίνακας `bmi_results` ([Υπόδειγμα 11](#)) που δημιουργήθηκε στο πλαίσιο της εφαρμογής ορίζεται με αυστηρούς κανόνες σχεσιακού μοντέλου (SQL), εξασφαλίζοντας την ακεραιότητα και την ασφάλεια των δεδομένων:

- `id` (bigint primary key): Το μοναδικό αναγνωριστικό της κάθε μέτρησης στην απομακρυσμένη βάση.
- `user_id` (uuid): Ξένο κλειδί (Foreign Key) που συνδέεται άμεσα με τον πίνακα αυθεντικοποίησης του Supabase (`auth.users(id)`). Η προσθήκη της ιδιότητας `on delete cascade` διασφαλίζει ότι, αν ένας χρήστης διαγράψει τον λογαριασμό του, θα διαγραφούν αυτόματα και όλες οι προσωπικές του μετρήσεις για λόγους συμμόρφωσης με τα δικαιώματα προστασίας προσωπικών δεδομένων (GDPR).
- `weight_kg`, `height_cm`, `bmi_value` (real): Πεδία κινητής υποδιαστολής που αποθηκεύουν με ακρίβεια το βάρος, το ύψος και τον τελικό υπολογισμένο δείκτη.
- `bmi_category` (text): Αλφαριθμητικό πεδίο που αποθηκεύει την κατηγοριοποίηση της μέτρησης (π.χ. "Φυσιολογικό Βάρος", "Υπέρβαρος").
- `created_at` (bigint): Αποθήκευση της χρονικής σήμανσης (Timestamp) σε μορφή Unix Epoch. Επιλέχθηκε η χρήση `bigint` αντί για `timestamp` για την επίτευξη απόλυτης συμβατότητας και ευκολίας στη σύγκριση χρόνου με το τοπικό επίπεδο της SQLite.

Όπως αναλύθηκε στο θεωρητικό υπόβαθρο, ο πίνακας προστατεύεται με την ενεργοποίηση του Row Level Security. Ορίστηκαν τρεις συγκεκριμένες πολιτικές (policies) που ελέγχουν τις λειτουργίες SELECT, INSERT και DELETE:

- Ανάγνωση & Διαγραφή: Η πρόσβαση ή η κατάργηση γραμμών επιτρέπεται μέσω της συνάρτησης `auth.uid() = user_id`, διασφαλίζοντας ότι κανένας χρήστης δεν μπορεί να υποκλέψει ή να διαγράψει μετρήσεις άλλου μέλους.

- Εγγραφή: Κατά την εισαγωγή (INSERT), ο έλεγχος with check (auth.uid() = user_id) επιβάλλει στον client να εισάγει δεδομένα που αντιστοιχούν αποκλειστικά στο δικό του αναγνωριστικό, αποτρέποντας την πλαστογράφηση δεδομένων (identity spoofing)

Υλοποίηση του Μηχανισμού Συγχρονισμού στο Kotlin Multiplatform

Για την πρακτική εφαρμογή της offline-first αρχιτεκτονικής, αναπτύχθηκε ένα εξειδικευμένο υποσύστημα εντός του κοινόχρηστου κώδικα (shared module) της Kotlin Multiplatform εφαρμογής. Το υποσύστημα αυτό αποτελείται από τρία βασικά δομικά στοιχεία: το αντικείμενο μεταφοράς δεδομένων (DTO), την απομακρυσμένη πηγή δεδομένων (Remote Data Source) που επικοινωνεί με το Supabase και τον διαχειριστή συγχρονισμού (Sync Manager), ο οποίος ενορχηστρώνει τη ροή των δεδομένων μεταξύ του τοπικού και του απομακρυσμένου επιπέδου.

Αντικείμενο Μεταφοράς Δεδομένων (Data Transfer Object - DTO)

Η επικοινωνία μέσω του δικτύου απαιτεί τη σειριοποίηση (serialization) των δεδομένων σε μια τυποποιημένη μορφή ανταλλαγής, όπως το JSON και για τον σκοπό αυτό, ορίστηκε η κλάση δεδομένων BmiResultDto όπως παρουσιάζεται στο [Υπόδειγμα 12](#) του Παραρτήματος «B».

Η χρήση του annotation @Serializable (από τη βιβλιοθήκη kotlinx.serialization) επιτρέπει την αυτόματη και αποδοτική μετατροπή της κλάσης σε JSON κατά την αποστολή στο Supabase, καθώς και την αποκωδικοποίησή της κατά τη λήψη. Η δομή του DTO αντιστοιχεί επακριβώς στο σχήμα (schema) του απομακρυσμένου πίνακα bmi_results της PostgreSQL, διασφαλίζοντας την ομαλή χαρτογράφηση (mapping) των πεδίων.

Απομακρυσμένη Πηγή Δεδομένων

Η κλάση SupabaseBmiRemoteDataSource ([Υπόδειγμα 13](#)) αναλαμβάνει την άμεση επικοινωνία με το backend API του Supabase, αξιοποιώντας το επίσημο Supabase KMP SDK. Μέσω της έγχυσης εξάρτησης (Dependency Injection) του SupabaseClient, η κλάση υλοποιεί τις απαραίτητες ασύγχρονες συναρτήσεις (suspend functions) για την εκτέλεση λειτουργιών CRUD στην απομακρυσμένη βάση με τις παρακάτω μεθόδους:

1. Εισαγωγή Δεδομένων (insert): Η συνάρτηση προωθεί το αντικείμενο BmiResultDto απευθείας στον πίνακα bmi_results χρησιμοποιώντας τη βιβλιοθήκη postgres.
2. Ανάκτηση Δεδομένων (getAll): Εκτελεί ένα φιλτραρισμένο ερώτημα επιλογής (select) στον server, απομονώνοντας τις καταγραφές με βάση το μοναδικό αναγνωριστικό του χρήστη (eq("user_id", userId)). Το αποτέλεσμα μετατρέπεται αυτόματα σε λίστα αντικειμένων μέσω της συνάρτησης .decodeList<BmiResultDto>().
3. Διαγραφή Δεδομένων (deleteBmi): Στοχεύει και διαγράφει μια συγκεκριμένη εγγραφή από την PostgreSQL, εφαρμόζοντας φίλτρο ισότητας στο πεδίο του αναγνωριστικού (eq("id", id)).

Λόγω της multiplatform φύσης του Supabase SDK, ο συγκεκριμένος κώδικας εκτελείται χωρίς καμία τροποποίηση και με την ίδια ακριβώς συμπεριφορά στα runtime περιβάλλοντα του Android, του iOS και του Desktop.

Ενορχήστρωση Συγχρονισμού

Η κλάση BmiSyncManager αποτελεί τον πυρήνα της offline-first αρχιτεκτονικής, καθώς γεφυρώνει την τοπική βάση δεδομένων (FitnessDatabase μέσω SQLDelight) με την απομακρυσμένη πηγή (SupabaseBmiRemoteDataSource). Η κεντρική συνάρτηση fullSync εκτελεί τη διαδικασία προώθησης των τοπικών αλλαγών στο υπολογιστικό νέφος (Cloud Push), ενώ έχει προβλεφθεί και η δομή για τη μελλοντική ενσωμάτωση της διαδικασίας λήψης των νέων απομακρυσμένων αλλαγών (Cloud Pull).

Η διαδικασία προώθησης των τοπικών αλλαγών εκτελείται ασύγχρονα και ακολουθεί την εξής αυστηρή επιχειρηματική λογική:

1. Ανάκτηση Εκκρεμοτήτων: Καλείται η εντολή selectPending() της SQLDelight, η οποία επιστρέφει σε μορφή λίστας όλες τις τοπικές εγγραφές BMI των οποίων η κατάσταση δεν είναι 'SYNCED'.
2. Διακλαδισμένη Επεξεργασία (State-based Execution): Για κάθε εκκρεμή εγγραφή (entity), εξετάζεται η τιμή του πεδίου sync_state μέσω μιας έκφρασης when:
 - α. Περίπτωση PENDING_INSERT: Σημαίνει ότι ο χρήστης πρόσθεσε μια νέα μέτρηση BMI (είτε online είτε offline). Ο Manager μετατρέπει την τοπική εγγραφή σε DTO (entity.toDto()) και καλεί τη συνάρτηση remote.insert().

Μόλις το Supabase επιβεβαιώσει την εγγραφή, εκτελείται η συνάρτηση `markSynced()`, η οποία ενημερώνει την τοπική βάση ότι η εγγραφή είναι πλέον πλήρως συγχρονισμένη (`SyncState.SYNCED`).

- β. Περίπτωση `PENDING_DELETE`: Σημαίνει ότι ο χρήστης επέλεξε να διαγράψει μια μέτρηση και η εφαρμογή εκτέλεσε "απαλή διαγραφή" (Soft Delete) τοπικά. Ο Manager καλεί τη συνάρτηση `remote.deleteBmi()` για να αφαιρεθεί η εγγραφή από το Supabase, και στη συνέχεια εκτελεί την εντολή `db.bmiResultsQueries.deleteBmi()`, διαγράφοντας πλέον οριστικά (Hard Delete) τη γραμμή από την τοπική SQLite βάση της συσκευής.

Η διαδικασία ανάκτησης απομακρυσμένων αλλαγών εκτελείται ασύγχρονα επίσης και ακολουθεί την εξής αρχιτεκτονική υποδομή για το Pull (`pullRemoteChanges`) επιδεικνύοντας τη στρατηγική επίλυσης διενέξεων (`conflict resolution`):

Η συνάρτηση ανακτά όλες τις εγγραφές από το Supabase (`remote.getAll(id)`) και για κάθε DTO ελέγχει αν υπάρχει ήδη στην τοπική βάση μέσω του `selectById(dto.id)`. Εάν η εγγραφή δεν ανιχνευθεί τοπικά (`local == null`), εισάγεται στην SQLite με κατάσταση `SyncState.SYNCED`, διασφαλίζοντας ότι η συσκευή ενημερώνεται με δεδομένα που ενδέχεται να είχαν εισαχθεί από άλλη πλατφόρμα (π.χ. ο χρήστης εισήγαγε δεδομένα στο Android και στη συνέχεια άνοιξε την Desktop ή την iOS έκδοση της εφαρμογής).

Η υλοποίηση αυτή αποδεικνύει πώς η χρήση της Kotlin Multiplatform, σε συνδυασμό με το Supabase, επιτρέπει τη δημιουργία ενός εύρωστου, ασφαλούς και αποδοτικού συστήματος διαχείρισης δεδομένων, μειώνοντας στο ελάχιστο το boilerplate score και εγγυώμενη την ακεραιότητα των ευαίσθητων δεδομένων φυσικής κατάστασης του χρήστη. Δίνεται υπόδειγμα υλοποίησης της συγχρονισμού στο [Υπόδειγμα 14](#) του Παραρτήματος «B».

Υλοποίηση του SupabaseClientProvider (Singleton & Lazy Initialization)

Η κλάση `SupabaseClientProvider` ([Υπόδειγμα 15](#)), αποτελεί το κεντρικό σημείο εισόδου για την επικοινωνία της εφαρμογής με την cloud πλατφόρμα Supabase. Για την υλοποίησή της επιλέχθηκε το αρχιτεκτονικό πρότυπο Singleton μέσω της χρήσης του Kotlin object. Η προσέγγιση αυτή εξασφαλίζει ότι καθ' όλη τη διάρκεια ζωής της εφαρμογής (Application Lifecycle) θα δημιουργηθεί και θα επαναχρησιμοποιηθεί μόνο ένα στιγμιότυπο (instance)

του client Supabase, εξοικονομώντας πόρους συστήματος και αποφεύγοντας περιττές συνδέσεις δικτύου.

Η αρχικοποίηση του client γίνεται με τη μέθοδο της καθυστερημένης αρχικοποίησης (Lazy Initialization) μέσω της εντολής `by lazy`. Αυτό σημαίνει ότι το αντικείμενο `SupabaseClient` δεν δεσμεύει μνήμη κατά την εκκίνηση της εφαρμογής, αλλά κατασκευάζεται μόνο τη στιγμή που θα ζητηθεί για πρώτη φορά από κάποια υπηρεσία.

Κατά τη διαμόρφωση (configuration) του client εισάγονται οι απαραίτητες παράμετροι σύνδεσης (URL και API Key) και εγκαθίστανται (install) δύο κρίσιμα υποσυστήματα (plugins) της βιβλιοθήκης:

- Υποσύστημα Ταυτοποίησης (Auth): Ενεργοποιεί τη διαχείριση χρηστών. Η ιδιότητα `autoSaveToStorage = true` επιτρέπει την αυτόματη και ασφαλή αποθήκευση των διακριτικών πρόσβασης (tokens) στην τοπική συσκευή, ώστε ο χρήστης να παραμένει συνδεδεμένος (Persistent Login). Παράλληλα, η ιδιότητα `alwaysAutoRefresh = true` αναλαμβάνει την αυτόματη ανανέωση των tokens (access tokens) μέσω των refresh tokens, διασφαλίζοντας την απρόσκοπτη ροή εργασίας χωρίς να απαιτείται εκ νέου αυθεντικοποίηση από τον χρήστη.
- Υποσύστημα Postgrest: Παρέχει το απαραίτητο API για την εκτέλεση λειτουργιών CRUD (Create, Read, Update, Delete) απευθείας στην PostgreSQL βάση δεδομένων του Supabase, χρησιμοποιώντας ασφαλείς και τυποποιημένες HTTP κλήσεις.

Αρχιτεκτονική και Λειτουργία της Κλάσης FullSync

Η κλάση `FullSync` ([Υπόδειγμα 16](#)) αποτελεί τον κεντρικό ενορχηστρωτή (orchestrator) της διαδικασίας συγχρονισμού δεδομένων μεταξύ της τοπικής βάσης δεδομένων (FitnessDatabase) και του απομακρυσμένου backend. Η κλάση αυτή εφαρμόζει τη λογική του καθολικού συγχρονισμού (Full Synchronization), η οποία είναι απαραίτητη για τη διασφάλιση της ακεραιότητας και της συνέπειας των δεδομένων (data consistency) σε περιβάλλοντα multi-platform, όπου ο χρήστης μπορεί να τροποποιήσει τα στοιχεία του από διαφορετικές συσκευές.

Η κλάση δέχεται ως παραμέτρους στον κατασκευαστή της (Constructor Injection) την τοπική βάση, τον πελάτη Supabase και το μοναδικό αναγνωριστικό του χρήστη (userId). Η βασική της συνάρτηση, `sync()`, είναι δηλωμένη ως `suspend`, γεγονός που υποδηλώνει

ότι η εκτέλεσή της βασίζεται στις ρουτίνες της Kotlin (Kotlin Coroutines). Με τον τρόπο αυτό, όλες οι δικτυακές και αποθηκευτικές διεργασίες εκτελούνται ασύγχρονα σε background thread, αποτρέποντας το πάγωμα του περιβάλλοντος εργασίας χρήστη (UI blocking).

Η διαδικασία συγχρονισμού εκτελείται σειριακά και δομείται σε επιμέρους διακριτά επίπεδα (Modular Architecture) μέσω εξειδικευμένων διαχειριστών (Sync Managers) και απομακρυσμένων πηγών δεδομένων (Remote Data Sources). Αυτός ο διαχωρισμός ευθυνών (Separation of Concerns) επιτρέπει την απομόνωση της επιχειρησιακής λογικής κάθε οντότητας:

1. Συγχρονισμός Προφίλ (ProfileSyncManager): Εκκινεί τη διαδικασία για τα προσωπικά στοιχεία του χρήστη, χρησιμοποιώντας την πηγή δεδομένων SupabaseProfileRemoteDataSource. Απαιτεί το userId για να απομονώσει και να επικαιροποιήσει τα δεδομένα του συγκεκριμένου λογαριασμού.
2. Συγχρονισμός Ασκήσεων (ExerciseSyncManager): Ενημερώνει τον κατάλογο των διαθέσιμων ασκήσεων μέσω του SupabaseExerciseRemoteDataSource. Καθώς πρόκειται για καθολικά δεδομένα της εφαρμογής (global data) που δεν σχετίζονται με συγκεκριμένο χρήστη, η μέθοδος fullSync() καλείται χωρίς παραμέτρους.
3. Συγχρονισμός Προπονήσεων (WorkoutSyncManager): Συντονίζει τη μεταφορά δεδομένων για το ιστορικό και τα πλάνα προπονήσεων του χρήστη, γεφυρώνοντας τα τοπικά καταγεγραμμένα δεδομένα με τα απομακρυσμένα μέσω του SupabaseWorkoutRemoteDataSource.
4. Συγχρονισμός Δείκτη Μάζας Σώματος (BmiSyncManager): Ολοκληρώνει τη διαδικασία με τον συγχρονισμό των μετρήσεων του Δείκτη Μάζας Σώματος (BMI) μέσω του SupabaseBmiRemoteDataSource, οι οποίες είναι αυστηρά προσωπικές και απαιτούν το userId.

Αυτοματοποίηση μέσω της Κλάσης SyncManager (Scheduler)

Η αυτοματοποίηση και η περιοδικότητα του συγχρονισμού των δεδομένων υλοποιούνται μέσω της κλάσης SyncManager ([Υπόδειγμα 17](#)). Η κλάση αυτή λειτουργεί ως ένας μηχανισμός χρονοπρογραμματισμού (Scheduler), ο οποίος αναλαμβάνει την εκτέλεση της διαδικασίας συγχρονισμού σε καθορισμένα χρονικά διαστήματα, διασφαλίζοντας ότι η

τοπική βάση δεδομένων παραμένει επικαιροποιημένη με το απομακρυσμένο backend χωρίς την ανάγκη χειροκίνητης παρέμβασης από τον χρήστη.

Αρχιτεκτονική Διαχείρισης Ασύγχρονων Διεργασιών (Concurrency)

Η υλοποίηση του SyncManager βασίζεται εξολοκλήρου στο μοντέλο των ρουτίνων της Kotlin (Kotlin Coroutines), το οποίο προσφέρει μια ελαφριά και αποδοτική λύση για την εκτέλεση background εργασιών σε cross-platform περιβάλλοντα (KMP):

- Καθορισμός Πεδίου (CoroutineScope): Για τη διαχείριση του κύκλου ζωής των ρουτίνων, η κλάση αρχικοποιεί ένα προσαρμοσμένο CoroutineScope.
- Επιλογέας Νημάτων (Dispatchers.Default): Χρησιμοποιείται ο Dispatchers.Default, ο οποίος αναθέτει την εκτέλεση των διεργασιών σε ένα κοινόχρηστο σύνολο νημάτων (background thread pool), βελτιστοποιημένο για εργασίες που απαιτούν υπολογιστική ισχύ ή συνεχή έλεγχο, προστατεύοντας το κύριο νήμα της διεπαφής (Main/UI Thread) από τυχόν καθυστερήσεις.
- Χρήση SupervisorJob: Η ενσωμάτωση του SupervisorJob στον κατασκευαστή του πεδίου είναι καθοριστικής σημασίας για την ανθεκτικότητα του συστήματος (fault tolerance). Σε αντίθεση με ένα κανονικό Job, το SupervisorJob διασφαλίζει ότι αν μια επιμέρους εργασία ή ένας συγχρονισμός αποτύχει με κάποιο σφάλμα (exception), η αποτυχία αυτή δεν θα συμπαρασύρει ολόκληρο το πεδίο (scope), επιτρέποντας στον SyncManager να συνεχίσει τη λειτουργία του στους επόμενους κύκλους.

Η έναρξη της διαδικασίας επιτυγχάνεται μέσω της συνάρτησης start(syncAction: suspend () -> Unit), η οποία δέχεται ως παράμετρο μια αναστελλόμενη συνάρτηση υψηλότερης τάξης (higher-order suspend function). Η προσέγγιση αυτή προσφέρει υψηλό βαθμό ευελιξίας (loose coupling), καθώς ο SyncManager δεν γνωρίζει τις λεπτομέρειες του συγχρονισμού, αλλά απλώς εκτελεί την ενέργεια που του ανατίθεται (στην προκειμένη περίπτωση, τη μέθοδο sync() της κλάσης FullSync). Η ροή εκτέλεσης περιλαμβάνει τα εξής στάδια:

1. Έλεγχος Διπλότυπης Εκτέλεσης: Κατά την κλήση της start, ελέγχεται αν η μεταβλητή job είναι ήδη ενεργή. Αυτό αποτρέπει τη δημιουργία πολλαπλών ταυτόχρονων κύκλων συγχρονισμού (race conditions) σε περίπτωση επαναλαμβανόμενων κλήσεων της μεθόδου.

2. Βρόχος Επανάληψης (Infinite Loop): Μέσα στην ρουτίνα εκτελείται ένας βρόχος `while(isActive)`. Η ιδιότητα `isActive` ελέγχει συνεχώς αν το πεδίο ή η συγκεκριμένη εργασία παραμένει ενεργή, επιτρέποντας τον άμεσο και καθαρό τερματισμό του βρόχου όταν ζητηθεί.
3. Διαχείριση Καθυστερήσεων και Σφαλμάτων (Polling Mechanism)
 - α. Η συνάρτηση εισάγει μια αρχική καθυστέρηση και στη συνέχεια καλεί ασύγχρονα τη συστημική ενέργεια `syncAction()`.
 - β. Ολόκληρος ο κώδικας εκτέλεσης είναι θωρακισμένος μέσα σε ένα μπλοκ `try-catch`. Αν προκύψει οποιοδήποτε σφάλμα δικτύου (π.χ. απώλεια σύνδεσης κατά τη διάρκεια του HTTP αιτήματος), το σφάλμα καταγράφεται (`catch`), αλλά δεν διακόπτει την εφαρμογή.
 - γ. Μετά την ολοκλήρωση ή την αποτυχία της ενέργειας, εφαρμόζεται μια καθυστέρηση ίση με την τιμή `syncIntervalMs` (με προκαθορισμένη τιμή τα 5.000 χιλιοστά του δευτερολέπτου/5 δευτερόλεπτα), πριν ξεκινήσει ο επόμενος κύκλος ελέγχου (Polling).

Τερματισμός Λειτουργίας (Lifecycle Management)

Η συνάρτηση `stop()` αναλαμβάνει τη διαχείριση της αποδέσμευσης των πόρων. Καλώντας τη μέθοδο `job?.cancel()`, ο `SyncManager` ακυρώνει την τρέχουσα ρουτίνα και σταματά αμέσως τον περιοδικό έλεγχο. Στη συνέχεια, η μεταβλητή `job` μηδενίζεται (`null`), καθιστώντας τον μηχανισμό έτοιμο για μια μελλοντική επανεκκίνηση. Η σωστή διαχείριση του τερματισμού είναι απαραίτητη για την αποφυγή διαρροών μνήμης (`memory leaks`), ειδικά κατά την εναλλαγή οθονών ή την έξοδο από την εφαρμογή.

3.2.4 Μηχανισμός Ταυτοποίησης (Authentication)

Η υλοποίηση του Μηχανισμού Ταυτοποίησης (Authentication) στο Network Layer της εφαρμογής βασίζεται στο οικοσύστημα του Supabase και συγκεκριμένα στη βιβλιοθήκη `Supabase-kt` (Jan-ten-Have), η οποία παρέχει πλήρη συμβατότητα με το KMP. Η αρχιτεκτονική προσέγγιση διαχωρίζεται σε τρία βασικά επίπεδα:

- Τη μοντελοποίηση των δεδομένων χρήστη (Data Transfer Objects - DTOs).
- Την υπηρεσία διαχείρισης των συνεδριών (Authentication Service).
- Τον κεντρικό πάροχο παραμετροποίησης του πελάτη Supabase (Client Provider).

Μοντελοποίηση Δεδομένων Χρήστη (UserDto)

Για την ανταλλαγή δεδομένων ταυτοποίησης με το API του Supabase χρησιμοποιείται η κλάση δεδομένων UserDto ([Υπόδειγμα 18](#)). Η κλάση αυτή φέρει τον σχολιασμό @Serializable της βιβλιοθήκης kotlinx.serialization, επιτρέποντας την αυτόματη μετατροπή των δεδομένων από και προς μορφή JSON κατά τις δικτυακές κλήσεις. Η κλάση περιλαμβάνει τα εξής πεδία:

- id: Μοναδικό αναγνωριστικό του χρήστη (UUID σε μορφή String) που παράγεται από το Supabase Auth.
- email: Η ηλεκτρονική διεύθυνση του χρήστη, η οποία λειτουργεί ως κύριο διαπιστευτήριο.
- created_at: Χρονική σήμανση δημιουργίας του λογαριασμού, αποτυπωμένη σε αντικείμενο Instant της βιβλιοθήκης kotlinx.datetime για ορθή διαχείριση του χρόνου ανεξαρτήτως πλατφόρμας (Android, iOS, Desktop).

Υπηρεσία Ταυτοποίησης (SupabaseAuthService)

Η κλάση SupabaseAuthService ([Υπόδειγμα 19](#)) αποτελεί τον κεντρικό μηχανισμό διαχείρισης των δικαιωμάτων πρόσβασης και των συνεδριών των χρηστών. Δέχεται ως εξάρτηση (dependency) ένα αντικείμενο SupabaseClient και υλοποιεί τις ασύγχρονες λειτουργίες μέσω των ανασταλτικών συναρτήσεων (suspend functions) των Kotlin Coroutines. Οι βασικές λειτουργίες που καλύπτει η υπηρεσία είναι:

1. Εγγραφή Χρήστη (signUp): Εκτελεί ασύγχρονη κλήση για τη δημιουργία νέου λογαριασμού χρησιμοποιώντας ως πάροχο (provider) την ενσωματωμένη λειτουργία email (Email).
2. Σύνδεση Χρήστη (signIn): Πραγματοποιεί την αυθεντικοποίηση υφιστάμενου χρήστη με βάση το email και τον κωδικό πρόσβασής του, εκκινώντας μια νέα συνεδρία.
3. Αποσύνδεση Χρήστη (signOut): Τερματίζει την τρέχουσα συνεδρία του χρήστη και καθαρίζει τα τοπικά διαπιστευτήρια.
4. Ανάκτηση Τρέχοντος Χρήστη (currentUser): Property που επιστρέφει τις πληροφορίες του τρέχοντος συνδεδεμένου χρήστη ή null σε περίπτωση απουσίας ενεργής συνεδρίας.

5. Κατάσταση Συνεδρίας (sessionStatus): Παρέχει άμεση πρόσβαση στην τρέχουσα κατάσταση της συνεδρίας (π.χ. Authenticated, NotAuthenticated, Loading) για τη δυναμική ενημέρωση της διεπαφής χρήστη.

Παραμετροποίηση και Προετοιμασία του Supabase Client

Η αρχικοποίηση του client Supabase πραγματοποιείται μέσω του singleton αντικειμένου SupabaseClientProvider. Η προσέγγιση αυτή εγγυάται ότι θα δημιουργηθεί μόνο ένα στιγμιότυπο του πελάτη για ολόκληρη την εφαρμογή, βελτιστοποιώντας τη διαχείριση των πόρων σε όλες τις υποστηριζόμενες πλατφόρμες (Android, iOS, Desktop) όπως αναφέρθηκε και προηγουμένως.

3.2.5 Ενσωμάτωση Μηχανισμών Τεχνητής Νοημοσύνης (Network Layer - Gemini API)

Στο πλαίσιο της παροχής εξατομικευμένης υποστήριξης προς τον χρήστη, η εφαρμογή ενσωματώνει έναν ψηφιακό βοηθό φυσικής κατάστασης (Fitness Support Bot). Η υλοποίηση αυτή βασίζεται στο μοντέλο μεγάλου γλωσσικού μεγέθους Gemini 2.5 Flash της Google, το οποίο προσφέρει χαμηλό χρόνο απόκρισης (latency) και υψηλή ακρίβεια σε εργασίες επεξεργασίας φυσικής γλώσσας. Η επικοινωνία με το API της Google πραγματοποιείται στο Network Layer μέσω του HTTP Client της βιβλιοθήκης Ktor, διασφαλίζοντας cross-platform συμβατότητα.

Βιβλιοθήκη Ktor

Η χρήση του Ktor HTTP Client ως ο βασικός δικτυακός πελάτης (client) της εφαρμογής αποτελεί μια στρατηγική αρχιτεκτονική επιλογή, πλήρως εναρμονισμένη με τις προδιαγραφές του KMP. Το Ktor έχει αναπτυχθεί από την JetBrains και σχεδιάστηκε εξ αρχής για να είναι ασύγχρονο, βασισμένο σε Kotlin Coroutines, και multi-platform. Στην παρούσα υλοποίηση, η χρήση του Ktor αναλύεται στους ακόλουθους βασικούς άξονες:

1. Cross-Platform Αρχιτεκτονική (Multi-platform Engine Support)

Σε αντίθεση με παραδοσιακές βιβλιοθήκες (όπως το Retrofit, το οποίο περιορίζεται αποκλειστικά στο οικοσύστημα του Android/Java), ο Ktor Client διαχωρίζει τη διεπαφή προγραμματισμού (API) από την υποκείμενη μηχανή εκτέλεσης (Engine).

- α. Στον κοινό κώδικα (commonMain) ορίζεται η δομή και η παραμετροποίηση του client.
 - β. Κατά τη μεταγλώττιση (compilation), η Ktor χρησιμοποιεί αυτόματα την κατάλληλη native μηχανή για κάθε πλατφόρμα-στόχο: OkHttp ή AndroidClientEngine για το Android, Darwin (NSURLSession) για το iOS, και CIO (Coroutine-based I/O) ή Apache για desktop εκδόσεις. Η προσέγγιση αυτή εξαλείφει την ανάγκη συγγραφής πλατφορμικά εξαρτώμενου κώδικα δικτύωσης.
2. Σύστημα Επεκτάσεων μέσω Plugins (Feature-Based Architecture)

Ο Ktor Client ακολουθεί μια εξαιρετικά αρθρωτή (modular) φιλοσοφία, όπου κάθε επιπλέον λειτουργικότητα εισάγεται ως ξεχωριστό πρόσθετο (Plugin) μέσω του μπλοκ `install()`. Στον παρεχόμενο κώδικα ([Υπόδειγμα 20](#)) χρησιμοποιείται το εξής κρίσιμο plugin:

- α. ContentNegotiation: Το plugin αυτό αναλαμβάνει τη διαπραγμάτευση περιεχομένου μεταξύ της εφαρμογής και του API του Gemini. Επιτρέπει στον Ktor να εξετάζει την κεφαλίδα Content-Type των αιτημάτων και των αποκρίσεων και να εκτελεί αυτόματα τη μετατροπή των δεδομένων.
- β. Json (kotlinx.serialization): Μέσα στο ContentNegotiation, ορίζεται ως μετατροπέας η επίσημη βιβλιοθήκη σειριοποίησης της Kotlin. Αυτό επιτρέπει στην ανασταλτική συνάρτηση `setBody(requestPayload)` να μετατρέπει αυτόματα την Kotlin Data Class `GeminiRequest` σε JSON String κατά την αποστολή, χωρίς χειροκίνητη παρέμβαση.

3. Παραμετροποίηση του JSON Serializer

Μέσα στη δομή του Ktor Client, η βιβλιοθήκη `kotlinx.serialization` παραμετροποιείται με δύο σημαντικές ιδιότητες:

- α. `ignoreUnknownKeys = true`: Ιδιότητα ζωτικής σημασίας για την ανθεκτικότητα της εφαρμογής. Αν το API του Gemini επιστρέψει επιπλέον πεδία (metadata, νέα χαρακτηριστικά) που δεν έχουν χαρτογραφηθεί στις δικές μας Data Classes, ο serializer τα αγνοεί αντί να προκαλέσει εξαίρεση και κατάρρευση (crash) της εφαρμογής.
- β. `prettyPrint = true`: Μορφοποιεί το παραγόμενο JSON με δομημένο τρόπο (αλλαγές γραμμών, εσοχές). Αν και αυξάνει ελαφρώς το μέγεθος του

payload, διευκολύνει σημαντικά τη διαδικασία αποσφαλμάτωσης (debugging) κατά την ανάπτυξη της διπλωματικής εργασίας.

4. Ασύγχρονη Εκτέλεση και Coroutines Integration

Ο Ktor είναι εγγενώς σχεδιασμένος να συνεργάζεται με τα Kotlin Coroutines. Η συνάρτηση `client.post` είναι μια ανασταλτική συνάρτηση (suspend function). Όταν εκτελείται η δικτυακή κλήση, το νήμα εκτέλεσης (thread) δεν "παγώνει" (non-blocking I/O). Η εκτέλεση αναστέλλεται απελευθερώνοντας τους πόρους της συσκευής, και συνεχίζει μόλις ληφθεί η απάντηση από τους διακομιστές της Google. Αυτό εγγυάται ότι η διεπαφή χρήστη (UI) παραμένει απόλυτα ομαλή (responsive) σε Android, iOS και Desktop, αποφεύγοντας φαινόμενα "παγώματος" της οθόνης.

5. Ευέλικτος Χειρισμός Αιτημάτων και Αποκρίσεων (Type-Safe Request DSL)

Ο Ktor χρησιμοποιεί μια DSL (Domain-Specific Language) βασισμένη σε Kotlin Lambdas για τη διαμόρφωση των HTTP αιτημάτων. Όπως φαίνεται στο block της `client.post`:

- α. Η εισαγωγή παραμέτρων, όπως το API Key, γίνεται με απόλυτα καθαρή σύνταξη μέσω της συνάρτησης `header()`.
- β. Ο τύπος του περιεχομένου ορίζεται ρητά με την ιδιότητα `contentType(ContentTypes.Application.Json)`.
- γ. Η διαχείριση της απόκρισης διευκολύνεται με τη συνάρτηση `response.bodyAsText()`, η οποία επιτρέπει την άμεση λήψη του ακατέργαστου κειμένου (raw string) για περαιτέρω επεξεργασία (parsing) μέσω του `Json.parseToJsonElement()`.

Μοντελοποίηση Δεδομένων και Δομή Payload (Gemini DTOs)

Για τη διασφάλιση της ορθότητας της δομής των δεδομένων (Type Safety) και την αποφυγή σφαλμάτων κατά τη σύνταξη των JSON αντικειμένων, αποφεύγεται η χρήση γενικών δομών όπως το `JsonObject`. Αντίθετα, ορίζεται μια αυστηρή ιεραρχία από Data Classes, οι οποίες φέρουν τον σχολιασμό `@Serializable` της βιβλιοθήκης `kotlinx.serialization` ([Υπόδειγμα 20](#)):

- Part: Η βασικότερη μονάδα πληροφορίας, η οποία περιέχει το πραγματικό κείμενο (text) του μηνύματος.

- **Content:** Αντιπροσωπεύει ένα μεμονωμένο μήνυμα στη συνομιλία. Διαθέτει το πεδίο `role` (το οποίο λαμβάνει τις τιμές "user" ή "model" για τον διαχωρισμό των συμμετεχόντων) και μια λίστα από `Part`.
- **SystemInstruction:** Περιλαμβάνει τις γενικές οδηγίες συμπεριφοράς και τους περιορισμούς που επιβάλλονται στο μοντέλο.
- **GeminiRequest:** Το τελικό αντικείμενο μεταφοράς δεδομένων (Root Payload) που αποστέλλεται στο API, το οποίο συνθέτει τις οδηγίες του συστήματος και το ιστορικό της συνομιλίας.

Αρχιτεκτονική και Υλοποίηση του FitnessSupportBot

Η κλάση `FitnessSupportBot` ([Υπόδειγμα 21](#)) αναλαμβάνει τη διαχείριση του κύκλου ζωής της συνομιλίας, την προετοιμασία των αιτημάτων και την ανάλυση των αποκρίσεων. Κατά την κατασκευή της κλάσης αρχικοποιείται ένα στιγμιότυπο του `HttpClient` του `Ktor`. Μέσω του plugin `ContentNegotiation` και της παραμέτρου `json()`, ορίζεται ότι η ανταλλαγή δεδομένων θα γίνεται σε μορφή `JSON`. Η ρύθμιση `ignoreUnknownKeys = true` διασφαλίζει τη σταθερότητα της εφαρμογής σε περίπτωση που το API επιστρέψει νέα, μη χαρτογραφημένα πεδία στο μέλλον.

Η συμπεριφορά του ψηφιακού βοηθού, ο οποίος ονομάζεται `PocketCoach`, καθορίζεται αυστηρά μέσω ενός κειμένου οδηγιών (`systemInstruction`). Οι κανόνες αυτοί επιβάλλουν:

1. **Συντομία (Be Brief):** Παροχή σύντομων και ενθαρρυντικών απαντήσεων.
2. **Δομή (Use Bullets):** Χρήση λιστών με κουκκίδες για βελτίωση της αναγνωσιμότητας.
3. **Ασφάλεια (Safety First):** Άμεση παραπομπή σε ιατρό σε περίπτωση αναφοράς τραυματισμού ή έντονου πόνου.
4. **Θεματικός Περιορισμός (Stay On-Topic):** Απόρριψη ερωτήσεων που δεν σχετίζονται με τη γυμναστική, την υγεία και τη διατροφή.

Η διατήρηση του κειμένου (`Context`) της συζήτησης επιτυγχάνεται μέσω της μεταβλητής `chatHistory`, η οποία είναι μια δυναμική λίστα τύπου `mutableListOf<Content>`. Η ροή εκτέλεσης της ανασταλτικής συνάρτησης `sendMessage` περιλαμβάνει τα εξής στάδια:

1. **Ενημέρωση Ιστορικού Χρήστη:** Το νέο μήνυμα του χρήστη εισάγεται στη λίστα `chatHistory` με ρόλο "user".

2. Σύνθεση Αιτήματος: Δημιουργείται το αντικείμενο GeminiRequest, το οποίο περιλαμβάνει τις πάγιες οδηγίες συστήματος και το σύνολο του ιστορικού της συνομιλίας έως εκείνη τη στιγμή.
3. Εκτέλεση HTTP POST: Πραγματοποιείται δικτυακή κλήση στην endpoint διεύθυνση του μοντέλου gemini-2.5-flash. Το API Key μεταφέρεται με ασφάλεια στην κεφαλίδα του αιτήματος (x-goog-api-key), ενώ η Ktor αναλαμβάνει την αυτόματη σειριοποίηση (serialization) του requestPayload σε JSON string στο σώμα του αιτήματος (setBody).
4. Ανάλυση Απόκρισης (JSON Parsing): Εάν η κλήση είναι επιτυχής (HttpStatusCode.OK), το JSON string της απάντησης μετατρέπεται σε JsonElement. Μέσω διαδοχικών προσβάσεων στα κλειδιά του δέντρου JSON (candidates → content → parts → text), απομονώνεται η πρωτογενής απάντηση του μοντέλου.
5. Ενημέρωση Ιστορικού Μοντέλου: Η απάντηση του Bot εισάγεται στο chatHistory με ρόλο "model", ώστε να λαμβάνεται υπόψη στις επόμενες ερωτήσεις του χρήστη, και επιστρέφεται στο UI Layer.

Η συνάρτηση περιβάλλεται από ένα μπλοκ try-catch για την ασφαλή διαχείριση εξαιρέσεων δικτύου ή αποτυχίας του διακομιστή, επιστρέφοντας κατάλληλα μηνύματα σφάλματος χωρίς να προκαλείται κατάρρευση (crash) της εφαρμογής. Τέλος, παρέχεται η συνάρτηση clearChat(), η οποία εκκαθαρίζει τη λίστα chatHistory, επιτρέποντας στον χρήστη να ξεκινήσει μια νέα, ανεξάρτητη συνεδρία συνομιλίας.

Μηχανισμός Παραγωγής και Δόμησης Ασκήσεων μέσω Τεχνητής Νοημοσύνης

Η μέθοδος generateAndAddExercise του ChatViewModel ([Υπόδειγμα 22](#)) αποτελεί τον πυρήνα της υλοποίησης για την αυτοματοποιημένη δημιουργία εξειδικευμένου περιεχομένου φυσικής κατάστασης. Ο αρχιτεκτονικός της ρόλος είναι ο ασύγχρονος συντονισμός μεταξύ των επιλογών του χρήστη στη διεπαφή (UI) και των δυνατοτήτων παραγωγικής τεχνητής νοημοσύνης (Generative AI) που προσφέρει το Domain Layer. Η μέθοδος δέχεται ως παραμέτρους την κεντρική ιδέα του χρήστη (userConcept), καθώς και τις επιλεγμένες μεταβλητές κατηγορίας (selectedCategory), δυσκολίας (selectedDifficulty) και μυϊκής ομάδας (selectedMuscle), διασφαλίζοντας την παραγωγή απόλυτα προσαρμοσμένου περιεχομένου.

Κατά την κλήση της μεθόδου, εκτελείται άμεσος έλεγχος εγκυρότητας των δεδομένων εισόδου, όπου αν η μεταβλητή `userConcept` είναι κενή (`isBlank()`), η εκτέλεση διακόπτεται πρόωρα (`early return`), αποτρέποντας την άσκοπη σπατάλη υπολογιστικών πόρων και δικτυακών αιτημάτων. Εφόσον η εισαγωγή είναι έγκυρη, πραγματοποιείται η πρώτη μεταβολή της κατάστασης της διεπαφής μέσω της μεθόδου `_uiState.update`. Η τρέχουσα κατάσταση τροποποιείται αμετάβλητα (`safely copied`) ως εξής:

- Το πεδίο `generatedExercise` μηδενίζεται (`null`), ώστε να καθαριστεί τυχόν προηγούμενος παραχθείσα άσκηση από τη μνήμη.
- Ο δυαδικός δείκτης `isLoading` τίθεται σε κατάσταση αλήθειας (`true`), ενεργοποιώντας το ημιδιαφανές επίπεδο αναμονής (`Loading Overlay`) στη διεπαφή χρήστη.
- Το πεδίο `errorMessage` αρχικοποιείται σε `null`, προετοιμάζοντας το σύστημα για την επερχόμενη ασύγχρονη διεργασία.

Το βασικότερο τεχνικό ζήτημα κατά την ενσωμάτωση Μεγάλων Γλωσσικών Μοντέλων (LLMs) σε δομημένες εφαρμογές είναι η εγγενής τάση τους να επιστρέφουν απαντήσεις σε ελεύθερη μορφή (`unstructured text`). Για την επίλυση αυτού του προβλήματος, αναπτύχθηκε μια εξειδικευμένη τεχνική `prompt engineering` μέσω της μεταβλητής `systemPrompt`. Το `prompt` αυτό λειτουργεί ως ένα αυστηρό πλαίσιο οδηγιών (`system-level instruction`) προς το Gemini API, επιβάλλοντας τους ακόλουθους περιορισμούς:

- Πλαισιοποίηση Ρόλου (`Role Prompting`): Το μοντέλο τοποθετείται στον ρόλο ενός επαγγελματία εκπαιδευτή φυσικής κατάστασης, ο οποίος οφείλει να συνθέσει έναν παρακινητικό τίτλο και αναλυτικές `step-by-step` οδηγίες εκτέλεσης.
- Ντετερμινιστική Μορφοποίηση (`Deterministic Formatting`): Επιβάλλεται η αποκλειστική χρήση του χαρακτήρα της κατακόρυφης γραμμής (`|`) ως μοναδικού διαχωριστικού στοιχείου μεταξύ του τίτλου και της περιγραφής (`Τίτλος|Περιγραφή`). Απαγορεύεται ρητά η εισαγωγή `markdown` στοιχείων, χαιρετισμών ή επιπλέον επεξηγήσεων.
- Γλωσσικός Συγχρονισμός: Το μοντέλο υποχρεούται να απαντήσει στην ίδια γλώσσα που χρησιμοποίησε ο χρήστης, διασφαλίζοντας τη διεθνοποίηση (`localization`) της ροής.

Η επικοινωνία με το API εκτελείται εντός της ρουτίνας `coroutineScope.launch`, διασφαλίζοντας ότι η δικτυακή κλήση δεν θα προκαλέσει πάγωμα (blocking) στο κεντρικό νήμα της διεπαφής (UI thread). Το σύστημα αποστέλλει το διαμορφωμένο prompt μέσω της μεθόδου `fitnessSupportBot.sendMessage(systemPrompt)` και αναμένει την ακατέργαστη απόκριση (`rawResponse`).

Μετά τη λήψη του κειμένου, εκτελείται έλεγχος εγκυρότητας για την επαλήθευση της ύπαρξης του διαχωριστικού χαρακτήρα (`contains("|")`). Εφόσον η συνθήκη ικανοποιείται, το κείμενο διασπάται μέσω της μεθόδου `.split("|")` σε έναν πίνακα συμβολοσειρών. Τα επιμέρους τμήματα απομονώνονται με ασφάλεια, καθαρίζονται από περιττά κενά ή χαρακτήρες αλλαγής γραμμής μέσω της `.trim()`, και χρησιμοποιούνται για την κατασκευή ενός νέου, typesafe αντικειμένου `GeneratedExercise(title, description)`.

Η ροή ολοκληρώνεται με τη δεύτερη ενημέρωση του `_uiState`, όπου το παραχθέν αντικείμενο εκχωρείται στο πεδίο `generatedExercise` και ο δείκτης `isLoading` απενεργοποιείται (`false`), πυροδοτώντας την παρενέργεια `LaunchedEffect` στο `Presentation Layer` για την αυτόματη συμπλήρωση των πεδίων της οθόνης. Σε περίπτωση που η απόκριση δεν τηρεί το προκαθορισμένο `format` ή αποτύχει λόγω σφάλματος δικτύου, ο μηχανισμός `try-catch` παγιδεύει την εξαίρεση, απενεργοποιεί τον δείκτη φόρτωσης και μεταφέρει το μήνυμα σφάλματος στο πεδίο `errorMessage`, διασφαλίζοντας την αδιάλειπτη και ασφαλή λειτουργία της εφαρμογής.

3.2.6 Presentation Layer (MVI/MVVM Αρχιτεκτονική & Jetpack Compose)

Το `Presentation Layer` της εφαρμογής ακολουθεί τα σύγχρονα αρχιτεκτονικά πρότυπα ανάπτυξης για Android, iOS και Desktop, συνδυάζοντας το μοτίβο `Model-View-ViewModel` (MVVM) με τις αρχές του MVI (`Model-View-Intent`) για τη διαχείριση της κατάστασης (`State Management`). Η σχεδίαση της διεπαφής χρήστη (UI) βασίζεται αποκλειστικά στο `Jetpack Compose/Compose Multiplatform`, το οποίο επιτρέπει τη δήλωση της οθόνης με δηλωτικό τρόπο (`Declarative UI`) και την κοινή χρήση του κώδικα διεπαφής μεταξύ των διαφορετικών πλατφορμών.

MVI/MVVM Αρχιτεκτονική

Τα MVVM και MVI είναι δημοφιλή μοτίβα αρχιτεκτονικής λογισμικού (`software architectural patterns`). Χρησιμοποιούνται κυρίως στην ανάπτυξη εφαρμογών (π.χ.

Android, iOS, web) για τον διαχωρισμό του κώδικα, ώστε να είναι πιο ευανάγνωστος, διαχειρίσιμος και εύκολος στον έλεγχο (testable) (Ignasio Cambanas, 2024).

MVVM (Model - View - ViewModel)

Το MVVM διαχωρίζει την εφαρμογή σε τρία βασικά συστατικά:

1. Model: Τα δεδομένα (data) και η επιχειρηματική λογική της εφαρμογής (business logic).
2. View: Το περιβάλλον χρήστη (UI), δηλαδή ό,τι βλέπει και αλληλεπιδρά ο χρήστης.
3. ViewModel: Ο συνδετικός κρίκος. Ανακτά δεδομένα από το Model, τα επεξεργάζεται και τα προετοιμάζει ώστε να εμφανιστούν σωστά στο View (Vaibhav Shakya, 2025).

Η φιλοσοφία του: Το View "παρακολουθεί" (observes) τις αλλαγές στο ViewModel και ενημερώνεται αυτόματα όταν αλλάζουν τα δεδομένα.

MVI (Model - View - Intent)

Το MVI βασίζεται στη μονόδρομη ροή δεδομένων (unidirectional data flow) και αποτελεί μια πιο σύγχρονη εξέλιξη για πολύπλοκες εφαρμογές:

- Model: Σε αντίθεση με το MVVM, εδώ το Model αντιπροσωπεύει την κατάσταση της οθόνης (State). Αποτελεί τη μοναδική πηγή αλήθειας (Single Source of Truth) για το τι πρέπει να δείχνει το UI ανά πάσα στιγμή (Gabriel Ribeiro, 2026).
- View: Εμφανίζει τα δεδομένα της οθόνης που λαμβάνει από το Model.
- Intent: Οι προθέσεις ή ενέργειες του χρήστη (π.χ. πάτημα ενός κουμπιού). Αυτές οι ενέργειες στέλνονται ως γεγονότα (events) που μεταβάλλουν την κατάσταση (Model).

Ποιες είναι οι διαφορές των MVVM και MVI

- Ροή Δεδομένων: Το MVVM επιτρέπει αμφίδρομη ροή (το View ενημερώνει το ViewModel και το ViewModel ενημερώνει το View). Στο MVI η ροή είναι αυστηρά μονόδρομη (Intent → Model → View).
- Κατάσταση: Στο MVVM έχεις λιγότερο έλεγχο στο πώς διαμορφώνεται το UI σε κάθε βήμα. Το MVI είναι ασφαλέστερο σε πολύπλοκες οθόνες γιατί διαχειρίζεται τα πάντα ως αμετάβλητες καταστάσεις (Immutable States).

Το MVVM είναι ιδανικό για απλές και γρήγορες εφαρμογές. Το MVI υπερέχει σε εφαρμογές με πολύπλοκες ροές (complex, multi-step workflows) ή δεδομένα που αλλάζουν συνεχώς.

Μοντελοποίηση της Κατάστασης της Οθόνης (BmiUiState)

Η κλάση δεδομένων BmiUiState ([Υπόδειγμα 23](#)) αποτελεί τη μοναδική πηγή αλήθειας (Single Source of Truth) για την κατάσταση της οθόνης υπολογισμού ανά πάσα χρονική στιγμή. Η προσέγγιση αυτή αποτρέπει αντιφάσεις στη διεπαφή χρήστη (UI inconsistencies):

- **isLoading**: Boolean μεταβλητή που υποδεικνύει αν εκτελείται κάποια ασύγχρονη διεργασία (π.χ. υπολογισμός, αποθήκευση στη βάση ή ενημέρωση προφίλ), ώστε να εμφανίζεται ένας καθολικός δείκτης προόδου (Loading Overlay).
- **bmiResult**: Φέρει το τρέχον αντικείμενο BmiResult που προέκυψε από τον υπολογισμό.
- **errorMessage**: Συγκρατεί τυχόν μηνύματα σφαλμάτων δικτύου ή βάσης, επιτρέποντας την ελεγχόμενη προβολή τους στον χρήστη.
- **isSaved**: Σημαία (Flag) που ενημερώνει το UI για την επιτυχή καταγραφή των δεδομένων.

Αρχιτεκτονική του BmiViewModel και Διαχείριση Ροών Δεδομένων

Το BmiViewModel ([Υπόδειγμα 24](#)) λειτουργεί ως ο ενορχηστρωτής της επιχειρησιακής λογικής (Business Logic) στο Presentation Layer. Δέχεται ως εξαρτήσεις το BmiRepository για την πρόσβαση στα δεδομένα, το CalculateBMIUseCase για την εκτέλεση του μαθηματικού τύπου και ένα CoroutineScope για τη διαχείριση των threads.

Η ανάκτηση του ιστορικού BMI γίνεται με τη χρήση της ροής bmiRepository.getAll(). Η μετατροπή της ψυχρής ροής (Cold Flow) σε θερμή ροή (Hot Flow) κατάστασης (StateFlow) επιτυγχάνεται μέσω του τελεστή stateIn():

- **SharingStarted.WhileSubscribed(5000)**: Μια κρίσιμη ρύθμιση για τη βελτιστοποίηση των πόρων της συσκευής. Εάν η οθόνη πάψει να είναι ορατή (π.χ. αλλαγή οθόνης ή ελαχιστοποίηση της εφαρμογής) για περισσότερα από 5 δευτερόλεπτα, η ροή σταματά να καταναλώνει πόρους και να παρακολουθεί τη βάση δεδομένων.

Ψυχρές (Cold Flows) και Θερμές (Hot Flows) Ροές στην Kotlin

Στο οικοσύστημα των Kotlin Coroutines και του Asynchronous Stream Processing, η διαχείριση των ασύγχρονων ροών δεδομένων πραγματοποιείται μέσω της διεπαφής Flow. Για την ορθή αρχιτεκτονική σχεδίαση της εφαρμογής και τη βελτιστοποίηση των πόρων της συσκευής, είναι απαραίτητος ο διαχωρισμός των ροών σε δύο βασικές κατηγορίες: τις Ψυχρές Ροές (Cold Flows) και τις Θερμές Ροές (Hot Flows).

Ψυχρές Ροές (Cold Flows)

Μια ροή δεδομένων χαρακτηρίζεται ως Ψυχρή (Cold) όταν ο κώδικας παραγωγής των δεδομένων εκτελείται μόνο όταν ένας καταναλωτής ξεκινήσει ρητά να συλλέγει (collect) τα δεδομένα αυτά.

- Μηχανισμός Λειτουργίας (Lazy Execution): Η ψυχρή ροή παραμένει αδρανής (declarative) μέχρι τη στιγμή που θα κληθεί μια τερματική συνάρτηση (terminal operator) όπως η collect().
- Ανεξαρτησία Καταναλωτών: Κάθε καταναλωτής (subscriber) λαμβάνει ένα δικό του, ανεξάρτητο αντίγραφο της ροής των δεδομένων. Η παραγωγή των δεδομένων ξεκινά από την αρχή για κάθε συλλέκτη ξεχωριστά (unicast συμπεριφορά).
- Εφαρμογή στο Data Layer: Η συνάρτηση bmiRepository.getAll() επιστρέφει μια ψυχρή ροή. Όταν η τοπική βάση δεδομένων (SQLDelight) επιστρέφει ένα Flow<List<BmiResult>>, δεν εκτελείται κάποιο ερώτημα (query) στη μνήμη μέχρι το UI Layer να ζητήσει ρητά τα δεδομένα.

Θερμές Ροές (Hot Flows)

Μια ροή δεδομένων χαρακτηρίζεται ως Θερμή (Hot) όταν η παραγωγή και η εκπομπή των δεδομένων πραγματοποιείται ανεξάρτητα από την ύπαρξη ενεργών καταναλωτών.

- Μηχανισμός Λειτουργίας (Eager Execution): Η ροή παράγει και διατηρεί δεδομένα στη μνήμη ακόμη και αν κανείς δεν τα παρακολουθεί τη δεδομένη στιγμή.
- Διαμοιρασμός Δεδομένων: Όλοι οι καταναλωτές μοιράζονται το ίδιο στιγμιότυπο της ροής και λαμβάνουν τις ίδιες εκπομπές δεδομένων ταυτόχρονα (multicast συμπεριφορά).

- Το StateFlow ως Θερμή Ροή: Το StateFlow αποτελεί μια εξειδικευμένη θερμή ροή. Λειτουργεί ως ένας αποθηκευτικός χώρος κατάστασης (state-holding observable flow), ο οποίος διατηρεί πάντα την τελευταία εκδοθείσα τιμή στη μνήμη και την εκπέμπει άμεσα σε οποιονδήποτε νέο καταναλωτή συνδεθεί.

Η Γέφυρα Μετατροπής: Ο Τελεστής stateIn()

Στην αρχιτεκτονική MVVM/MVI, η απευθείας έκθεση μιας ψυχρής ροής από το Data Layer στο UI Layer θεωρείται κακή πρακτική. Αν δύο Composable στοιχεία καλούσαν ταυτόχρονα την collect(), η βάση δεδομένων θα εκτελούσε το ίδιο query δύο φορές άσκοπα. Για τον λόγο αυτό, χρησιμοποιείται ο τελεστής stateIn(), ο οποίος μετατρέπει την ψυχρή ροή σε θερμή (StateFlow).

Η παραμετροποίηση του τελεστή στο BmiViewModel αναλύεται ως εξής:

- scope = coroutineScope: Καθορίζει το πεδίο εκτέλεσης (CoroutineScope) το οποίο κρατά τη θερμή ροή ενεργή στη μνήμη. Όσο το ViewModel είναι ζωντανό, η ροή παραμένει διαθέσιμη.
- started = SharingStarted.WhileSubscribed(5000): Ρυθμίζει τη στρατηγική ενεργοποίησης (Sharing Strategy). Η θερμή ροή θα ξεκινήσει να συλλέγει δεδομένα από την ψυχρή ροή της βάσης μόνο όταν υπάρχει τουλάχιστον ένας καταναλωτής στο UI. Το όριο των 5 δευτερολέπτων (5000ms) εξασφαλίζει ότι αν ο χρήστης περιστρέψει την οθόνη (configuration change) ή αλλάξει στιγμιαία παράθυρο, η ροή δεν θα κλείσει και δεν θα ξανανοίξει άσκοπα, προστατεύοντας τους πόρους του συστήματος.
- initialValue = emptyList(): Επειδή η ανάκτηση από τη βάση δεδομένων είναι ασύγχρονη, το StateFlow απαιτεί μια άμεση, αρχική τιμή για να σχεδιαστεί το UI χωρίς καθυστέρηση. Στην προκειμένη περίπτωση, ορίζεται μια άδεια λίστα.

Διαχείριση Λειτουργιών CRUD μέσω Coroutines

Όλες οι λειτουργίες τροποποίησης δεδομένων εκτελούνται ασύγχρονα μέσα σε Coroutine Scopes μέσω της coroutineScope.launch:

1. Εισαγωγή Δεδομένων (insert):
 - α. Ενεργοποιεί την κατάσταση φόρτωσης (isLoading = true).

- β. Καλεί το Use Case `calculateBMIUseCase.execute()` για τον υπολογισμό του σκορ.
 - γ. Δημιουργεί ένα νέο αντικείμενο `BmiResult` εισάγοντας τη χρονική σήμανση συστήματος μέσω της `multiplatform` βιβλιοθήκης `kotlinx.datetime.Clock`.
 - δ. Θέτει την αρχική κατάσταση χρονισμού ως `SyncState.PENDING_INSERT`, προετοιμάζοντας το `Network Layer` για τον μελλοντικό συγχρονισμό με το `Supabase`.
2. Λογική Διαγραφή (`softDelete`): Αντί για την άμεση διαγραφή από την τοπική βάση, το αντικείμενο ενημερώνεται με τη σημαία `isDeleted = 1` και κατάσταση `PENDING_DELETE`. Αυτή η πρακτική (`Soft Delete`) είναι απαραίτητη σε `cross-platform` εφαρμογές για να γνωρίζει ο μηχανισμός συγχρονισμού ότι πρέπει να διαγράψει την εγγραφή και από τον απομακρυσμένο διακομιστή (`Supabase`) όταν υπάρξει σύνδεση στο διαδίκτυο.
3. Φυσική Διαγραφή (`deleteBmi`): Εκτελεί οριστική διαγραφή της εγγραφής από το τοπικό αποθετήριο.

Ανάπτυξη των Οθονών Διεπαφής Χρήστη (Compose Multiplatform Screens)

Οι `Composable` συναρτήσεις `BmiScreen` ([Υπόδειγμα 26](#)) και `BmiHistoryScreen` ([Υπόδειγμα 25](#)) υλοποιούν τη διεπαφή χρήστη με βάση τις προδιαγραφές του `Material Design 3`, διασφαλίζοντας κοινή εμφάνιση και συμπεριφορά σε `Android`, `iOS` και `Desktop`.

Οθόνη Υπολογισμού BMI (BmiScreen)

Η οθόνη `BmiScreen` αποτελεί το κεντρικό σημείο αλληλεπίδρασης του χρήστη για τον υπολογισμό του δείκτη μάζας σώματος εκτελώντας τις παρακάτω λειτουργίες:

- Κατανάλωση Κατάστασης από Πολλαπλά `ViewModels`: Μέσω ενός κεντρικού `GeneralViewModel`, η οθόνη συνδυάζει και παρακολουθεί με ασφάλεια (`Lifecycle-aware`) τρεις διαφορετικές καταστάσεις: `bmiViewModel.uiState`, `authViewModel.uiState` (για τα στοιχεία του συνδεδεμένου χρήστη) και `profileViewModel.uiState` (για την ανάκτηση του αποθηκευμένου ύψους).
- Διαχείριση Τοπικής Κατάστασης (`Local State`): Χρησιμοποιούνται οι συναρτήσεις `remember { mutableStateOf(...) }` για τη διατήρηση των δεδομένων εισαγωγής

- βάρους (weight) και του τοπικού σφάλματος εγκυρότητας (inputError) κατά τις ανασυνθέσεις (recompositions) της οθόνης.
- Εξυγίανση Δεδομένων Εισαγωγής (Input Sanitization): Κατά την πληκτρολόγηση στο OutlinedTextField, το UI αντικαθιστά αυτόματα το κόμμα (,) με τελεία (.) για την αποφυγή σφαλμάτων μετατροπής, ενώ επιτρέπει μόνο έγκυρους δεκαδικούς αριθμούς.
 - Επικύρωση Δεδομένων και Συγχρονισμός (Validation & Multi-command execution): Κατά το πάτημα του κουμπιού "Calculate BMI", εκτελείται έλεγχος εγκυρότητας (εύρος 1 έως 300 kg). Αν τα δεδομένα είναι έγκυρα, εκτελείται ταυτόχρονα η καταγραφή του νέου BMI και η αυτόματη ενημέρωση του τρέχοντος βάρους στο προφίλ του χρήστη (updateProfile), κρατώντας τα δεδομένα συγχρονισμένα.
 - Δυναμική Εμφάνιση Αποτελέσματος: Εάν υπάρχει υπολογισμένο αποτέλεσμα, εμφανίζεται ένα Material Card σε εμφανές σημείο με έντονη γραμματοσειρά (FontWeight.Black).
 - Καθολικό Loading Overlay: Όταν η ιδιότητα uiState.isLoading είναι αληθής, εμφανίζεται ένα ημιδιαφανές επίπεδο (Box) με ένα CircularProgressIndicator στο κέντρο, εμποδίζοντας την παράλληλη αλληλεπίδραση του χρήστη μέχρι να ολοκληρωθεί η εγγραφή.

Οθόνη Ιστορικού Καταγραφών (BmiHistoryScreen)

Η οθόνη BmiHistoryScreen αναλαμβάνει την αναδρομική προβολή των αποθηκευμένων μετρήσεων όπως αναλύεται παρακάτω:

- Ασφαλής Συλλογή Δεδομένων: Η συλλογή του ιστορικού γίνεται με τη συνάρτηση collectAsStateWithLifecycle(), διασφαλίζοντας ότι όταν η εφαρμογή περνά στο παρασκήνιο (background), η παρατήρηση της βάσης δεδομένων παύει αυτόματα για εξοικονόμηση ενέργειας.
- Λογική Empty State: Υλοποιείται έλεγχος για το αν η λίστα είναι άδεια (results.isEmpty()). Σε περίπτωση απουσίας εγγραφών, το UI εμφανίζει ένα κεντραρισμένο επεξηγηματικό μήνυμα (No BMI records found yet), βελτιώνοντας την εμπειρία χρήστη (User Experience).

- Αποδοτική Απόδοση Λίστας: Σε επόμενο στάδιο, οι εγγραφές αποδίδονται μέσω ενός LazyColumn. Το στοιχείο αυτό ανακυκλώνει τα γραφικά συστατικά (UI components), εξασφαλίζοντας υψηλή απόδοση (rendering performance) και ομαλή ολίσθηση (smooth scrolling) σε όλες τις υποστηριζόμενες πλατφόρμες (Android, iOS, Desktop), ανεξάρτητα από το μέγεθος του ιστορικού.

3.2.7 Υλοποίηση Κεντρικής Διεπαφής Χρήστη (Presentation Layer – HomeScreen)

Η κλάση HomeScreen ([Υπόδειγμα 27](#)) αποτελεί το κεντρικό σημείο πλοήγησης και τη βασική οθόνη αλληλεπίδρασης (Dashboard) της εφαρμογής. Η υλοποίησή της βασίζεται στο δηλωτικό μοντέλο του Compose Multiplatform, εξασφαλίζοντας ενιαία απόδοση του γραφικού περιβάλλοντος και των αλληλεπιδράσεων στις πλατφόρμες Android, iOS και Desktop.

Αρχιτεκτονική Προσέγγιση και Αποσύνδεση Εξαρτήσεων (Decoupling)

Η συνάρτηση HomeScreen έχει σχεδιαστεί σύμφωνα με την αρχή της ανύψωσης κατάστασης (State Hoisting) και τη χρήση συμβάντων (Events) ως παραμέτρων. Δεν διατηρεί εσωτερικά στιγμιότυπα από ViewModels ή αποθετήρια δεδομένων. Αντίθετα, δέχεται μια σειρά από callbacks υψηλής βαθμίδας (lambda functions):

- onLogout: Διαχειρίζεται τον τερματισμό της συνεδρίας (session termination).
- onProfileClick, onBmiClick, onExercisesClick, onWorkoutPlansClick, onPocketCoachClick: Ενορχηστρώνουν την πλοήγηση (Routing) προς τα επιμέρους υποσυστήματα της εφαρμογής.

Η αρχιτεκτονική αυτή προσέγγιση καθιστά το Component πλήρως ελέγξιμο (testable) μέσω αυτοματοποιημένων δοκιμών UI (UI Testing), καθώς και επαναχρησιμοποιήσιμο, αφού η λογική πλοήγησης μετατίθεται στον κεντρικό πλοηγό (NavHost) της εφαρμογής.

Δομή Διάταξης και Σχεδιαστικά Πρότυπα (Layout & Material Design 3)

Η οθόνη ακολουθεί τις προδιαγραφές του Material Design 3, χρησιμοποιώντας ένα σύστημα δυναμικών χρωματικών συνδυασμών (Color Schemes) και τυπογραφίας για τη διασφάλιση υψηλής οπτικής ιεραρχίας.

Επικεφαλίδα και Διαχείριση Κύλισης

Η ιεραρχία των στοιχείων οργανώνεται κατακόρυφα μέσω ενός Column. Το κεντρικό περιεχόμενο περιβάλλεται από έναν μηχανισμό διαχείρισης κατάστασης κύλισης

(verticalScroll(rememberScrollState()))). Η επιλογή αυτή εγγυάται ότι η οθόνη θα παραμείνει λειτουργική και προσβάσιμη σε συσκευές με περιορισμένες διαστάσεις οθόνης (π.χ. παλαιότερα smartphone) ή κατά την αλλαγή του προσανατολισμού (landscape mode) στις φορητές συσκευές.

Διαστρωμάτωση Επιλογών και Ψευδο-πλέγμα (Grid Layout)

Για τη βελτιστοποίηση της γνωστικής φόρτωσης (cognitive load) του χρήστη, οι λειτουργίες κατηγοριοποιούνται με βάση τη σπουδαιότητά τους:

1. Πρωτεύον Στοιχείο (Full-Width Highlight Card): Η υπηρεσία τεχνητής νοημοσύνης PocketCoach AI καταλαμβάνει το πλήρες πλάτος της διάταξης. Χρησιμοποιεί το χρώμα primaryContainer, το οποίο την καθιστά οπτικά κυρίαρχη, υπογραμμίζοντας την προστιθέμενη αξία του υποσυστήματος AI στην εφαρμογή.
2. Δευτερεύοντα Στοιχεία (2x2 Grid): Οι υπόλοιπες τέσσερις λειτουργίες διατάσσονται σε δύο σειρές (Row) των δύο στοιχείων, με τη χρήση του τροποποιητή Modifier.weight(1f). Η τεχνική αυτή εξασφαλίζει την ισομερή κατανομή του διαθέσιμου χώρου (equally proportioned layout) ανεξάρτητα από την πυκνότητα εικονοστοιχείων (pixel density) της εκάστοτε οθόνης.

Επαναχρησιμοποιήσιμα Γραφικά Συστατικά (DashboardCard)

Για την τήρηση της αρχής DRY (Don't Repeat Yourself), η οπτική αναπαράσταση των επιλογών του πλέγματος υλοποιείται μέσω της Composable συνάρτησης DashboardCard.

Το component αυτό παραμετροποιείται πλήρως μέσω των ορισμάτων του, προσφέροντας:

- Αποδοτικότητα Μνήμης: Η επαναχρησιμοποίηση του component μειώνει το μέγεθος του παραγόμενου byte-code και βελτιστοποιεί τη διαδικασία ανασύνθεσης (recomposition) του Compose.
- Περιορισμό Κειμένου (maxLines = 1): Η επιβολή του ορίου μίας γραμμής στα πεδία τίτλου και υπότιτλου αποτρέπει την αλλοίωση της γραφικής διάταξης (layout truncation/overlapping) σε περιπτώσεις όπου το μέγεθος της γραμματοσειράς του συστήματος είναι αυξημένο από τον χρήστη (Accessibility / Accessibility Scaling).

Σταθερές Ενέργειες και Ασφάλεια Διεπαφής (Footer Actions)

Το κουμπί αποσύνδεσης (Sign Out) είναι τοποθετημένο στο κατώτερο τμήμα της κεντρικής διάταξης, εκτός του scrollable περιβάλλοντος, παραμένοντας σταθερά προσβάσιμο.

Ακολουθώντας τα πρότυπα σχεδιασμού για την αποφυγή ακούσιων ενεργειών (destructive actions), χρησιμοποιείται η κλάση `OutlinedButton` με χρωματική σήμανση σφάλματος (`MaterialTheme.colorScheme.error`). Η έντονη κόκκινη οπτική ειδοποίηση (visual cue) ενημερώνει υποσυνείδητα τον χρήστη για τον μη αναστρέψιμο χαρακτήρα της ενέργειας (τερματισμός τρέχουσας συνεδρίας), βελτιώνοντας τη συνολική χρηστικότητα (Usability) του λογισμικού.

3.2.8 Αρχιτεκτονική Πλοήγησης και Διασφάλιση Τύπων (Navigation Layer, Type-Safe Routes & TopAppBar)

Η αρχιτεκτονική πλοήγησης (Navigation Layer) της cross-platform εφαρμογής έχει υλοποιηθεί με βάση τη σύγχρονη έκδοση της βιβλιοθήκης Jetpack Navigation for Compose. Η προσέγγιση αυτή εισάγει το πρότυπο Type-Safe Navigation, το οποίο αντικαθιστά πλήρως τις παραδοσιακές διαδρομές που βασίζονταν σε αλφαριθμητικά κείμενα (String-based routes).

Η δομή του Navigation Layer αναλύεται σε τρεις βασικούς πυλώνες: τον ορισμό των σταθερών και παραμετροποιημένων διαδρομών (Routes) ([Υπόδειγμα 28](#)), την κεντρική μπάρα πλοήγησης (FitnessAppBar) ([Υπόδειγμα 29](#)) και τον κεντρικό ενορχηστρωτή των οθονών (NavApp) ([Υπόδειγμα 30](#)).

Μοντελοποίηση Διαδρομών (Type-Safe Routes)

Για τον καθορισμό των προορισμών στο γράφημα πλοήγησης, χρησιμοποιούνται αντικείμενα (object) και κλάσεις (class) της Kotlin ([Υπόδειγμα 28](#)), τα οποία φέρουν υποχρεωτικά τον σχολιασμό `@Serializable` της βιβλιοθήκης `kotlinx.serialization`. Η προσέγγιση αυτή επιτρέπει στο Navigation Component να αναγνωρίζει τις διαδρομές ως μοναδικούς στατικούς τύπους δεδομένων κατά τη φάση της μεταγλώττισης (Compile-time Type Safety), εκμηδενίζοντας σφάλματα πληκτρολόγησης (syntax/runtime errors). Οι διαδρομές διαχωρίζονται σε δύο κατηγορίες:

1. Στατικοί Προορισμοί (Objects):

Χρησιμοποιούνται για οθόνες που δεν απαιτούν εξωτερικά δεδομένα εισόδου για να αρχικοποιηθούν:

- α. Auth: Η οθόνη εισέλευσης και αυθεντικοποίησης του χρήστη.
- β. Home: Το κεντρικό ταμπλό ελέγχου (Fitness Dashboard).
- γ. Profile: Η οθόνη διαχείρισης των προσωπικών και σωματομετρικών στοιχείων.
- δ. Bmi & BmiHistory: Οι οθόνες υπολογισμού του δείκτη μάζας σώματος και προβολής του ιστορικού καταγραφών αντίστοιχα. Exercises & InsertExercises: Η βιβλιοθήκη ασκήσεων και η φόρμα προσθήκης νέων ασκήσεων.
- ε. WorkoutPlans & InsertWorkoutPlan: Η διαχείριση και η δημιουργία πλάνων προπόνησης.
- στ. Chat: Η οθόνη αλληλεπίδρασης με τον ψηφιακό βοηθό τεχνητής νοημοσύνης PocketCoach AI.

2. Δυναμικοί/Παραμετροποιημένοι Προορισμοί (Classes)

Χρησιμοποιούνται για οθόνες που απαιτούν τη μεταφορά δεδομένων (arguments) κατά τη μετάβαση, διασφαλίζοντας ότι οι παράμετροι μεταφέρονται με απόλυτη ασφάλεια τύπων:

- α. ExerciseDetail(val id: Long): Δέχεται το μοναδικό αναγνωριστικό (id) της άσκησης ως Long, ώστε να αντλήσει τις συγκεκριμένες λεπτομέρειες από το Data Layer.
- β. WorkoutPlanDetail(val planId: String): Δέχεται το αναγνωριστικό του πλάνου ως String, επιτρέποντας τη δυναμική φόρτωση των ασκήσεων που ανήκουν στο συγκεκριμένο πρόγραμμα.

Δυναμική Μπάρα Πλοήγησης (FitnessAppBar)

Η Composable συνάρτηση FitnessAppBar υλοποιεί την κλάση CenterAlignedTopAppBar του Material Design 3, προσφέροντας μια ομοιόμορφη και κεντραρισμένη αισθητική στην κορυφή της εφαρμογής σε Android, iOS και Desktop όπως παρακάτω:

- Ανάκτηση Τίτλου μέσω hasRoute: Η μπάρα προσαρμόζει τον τίτλο της δυναμικά εξετάζοντας την τρέχουσα τοποθεσία του χρήστη (currentScreen). Ο έλεγχος γίνεται με την inline επέκταση currentScreen.hasRoute<T>(). Μέσω μιας πολυμορφικής δομής when, ταυτοποιείται ο ακριβής τύπος της διαδρομής (π.χ.

BmiHistory, WorkoutPlans, Chat) και ανακτάται το αντίστοιχο μεταφρασμένο λεκτικό από τους διεθνοποιημένους πόρους της εφαρμογής (stringResource(Res.string.*)).

- Έλεγχος Ιστορικού (Back-Navigation): Μέσω της Boolean μεταβλητής `canNavigateBack`, η μπάρα ελέγχει αν υπάρχει προηγούμενη σελίδα στο Backstack. Εάν η συνθήκη είναι αληθής, εμφανίζει αυτόματα ένα `IconButton` με το διανυσματικό γραφικό της επιστροφής (`arrow_back_24px`), το οποίο ενεργοποιεί το callback `navigateUp()` για την οπισθοδρόμηση του χρήστη.

Κεντρικός Ενορχηστρωτής Πλοήγησης (NavApp)

Η συνάρτηση `NavApp` αποτελεί τη ρίζα (Root Component) του οπτικού περιβάλλοντος, ενοποιώντας τη μπάρα πλοήγησης και το γράφημα των οθονών μέσω ενός `Material Scaffold`.

- Διαχείριση της Κατάστασης (Backstack State): Μέσω του `rememberNavController()`, η εφαρμογή παρακολουθεί την κατάσταση πλοήγησης. Η γραμμή κώδικα `navController.currentBackStackEntryAsState()` μετατρέπει το ιστορικό σε αντιδραστική ροή (State), ενημερώνοντας σε πραγματικό χρόνο τη `FitnessAppBar` για οποιαδήποτε αλλαγή οθόνης.
- Ασφάλεια Επιστροφής: Η ιδιότητα `canNavigateBack` παραμετροποιείται ως:

```
canNavigateBack = navController.previousBackStackEntry !=  
                    null && !currentScreen!!.hasRoute<Auth>()
```

Εικόνα 3.1 Η ιδιότητα `canNavigateBack`

Η λογική αυτή αποτρέπει την εμφάνιση του κουμπιού επιστροφής όταν ο χρήστης βρίσκεται στην οθόνη αυθεντικοποίησης (Auth), διασφαλίζοντας τη σωστή ροή της εφαρμογής.

- Δομή του `NavHost` και Καθαρισμός Ιστορικού: Το `NavHost` ορίζει το γράφημα πλοήγησης με αρχικό προορισμό (`startDestination`) το Auth. Κατά την επιτυχή είσοδο (`onSignInSuccess`) ή εγγραφή (`onSignUpSuccess`), εκτελείται μετάβαση προς το Home ή το Profile αντίστοιχα, ενώ με τη χρήση του `popUpTo(Auth) { inclusive = true }` η οθόνη αυθεντικοποίησης αφαιρείται οριστικά από τη μνήμη.

Αυτό εμποδίζει τον χρήστη να επιστρέψει στο Login σε περίπτωση που πατήσει το κουμπί επιστροφής της συσκευής.

- Εξαγωγή Παραμέτρων: Στις composable διαδρομές που απαιτούν ορίσματα (όπως οι ExerciseDetail και WorkoutPlanDetail), η ανάκτηση των δεδομένων εκτελείται με απόλυτη ασφάλεια μέσω της συνάρτησης `backStackEntry.toRoute<T>()`. Το αντικείμενο που επιστρέφεται χρησιμοποιείται άμεσα για το φιλτράρισμα των δεδομένων (π.χ. `find { it.id == exerciseDetail.id }`) από την τρέχουσα κατάσταση των ViewModels.

Αρχιτεκτονικά Οφέλη του Σχεδιασμού

Η συγκεκριμένη υλοποίηση προσφέρει τρία κεντρικά πλεονεκτήματα στη μηχανική του λογισμικού (Software Engineering) της διπλωματικής εργασίας:

- Διαχωρισμός Ευθυνών (Separation of Concerns): Οι επιμέρους οθόνες παραμένουν αποσυνδεδεμένες (decoupled) από τη λογική της πλοήγησης. Δέχονται απλά callbacks (όπως `onBmiClick`), αφήνοντας τον κεντρικό NavHost να διαχειριστεί τις μεταβάσεις.
- Ασφάλεια Τύπων κατά τη Μεταγλώττιση (Compile-Time Safety): Οποιαδήποτε αλλαγή στη δομή μιας οθόνης ή στις παραμέτρους της (π.χ. αλλαγή του `id` από Long σε String) θα προκαλέσει σφάλμα κατά το build, αποτρέποντας την κυκλοφορία ελαττωματικού κώδικα (bugs).
- Κοινή Χρήση Κατάστασης (Shared State): Το κεντρικό `generalViewModel` περνά ως εξάρτηση σε όλους τους κόμβους, επιτρέποντας την απρόσκοπτη επικοινωνία και τον διαμοιρασμό δεδομένων μεταξύ των διαφορετικών υποσυστημάτων της εφαρμογής.

3.2.9 Υλοποίηση Κεντρικής Συνάρτησης App και Ενορχήστρωση Επιπέδου Παρουσίασης

Η αρχιτεκτονική της αναπτυχθείσας εφαρμογής βασίζεται στην κεντρική συνάρτηση App ([Υπόδειγμα 31](#)), η οποία αποτελεί το ριζικό σημείο εισόδου της διεπαφής χρήστη και τον πυρήνα ενορχήστρωσης του Επιπέδου Παρουσίασης. Η συγκεκριμένη composable συνάρτηση αναλαμβάνει τη διαχείριση του κύκλου ζωής των καθολικών εξαρτήσεων, την προετοιμασία των μοντέλων προβολής μέσω του μοτίβου Composite ViewModel, καθώς και τον έλεγχο των καταστάσεων μετάβασης προς τον κύριο πλοηγό της εφαρμογής.

Διαχείριση Εξαρτήσεων και Αρχικοποίηση Υποδομών

Η συνάρτηση App δέχεται ως παράμετρο εισόδου το στιγμιότυπο της τοπικής βάσης δεδομένων FitnessDatabase. Με τη χρήση του API remember, διασφαλίζεται ότι οι δαπανηρές λειτουργίες αρχικοποίησης των υποστηρικτικών μηχανισμών εκτελούνται αποκλειστικά κατά την αρχική σύνθεση (Initial Composition) και δεν επαναλαμβάνονται άσκοπα κατά τις ανασυνθέσεις (Recompositions):

- Supabase Integration: Αρχικοποιείται ο πελάτης δικτύου μέσω του SupabaseClientProvider.client και ενσωματώνεται στην υπηρεσία ταυτοποίησης SupabaseAuthService για τον έλεγχο των δικαιωμάτων πρόσβασης των χρηστών.
- Μηχανισμός Συγχρονισμού: Δημιουργείται το στιγμιότυπο του SyncManager για την παρακολούθηση των εκκρεμών offline εγγραφών και τον αμφίδρομο συγχρονισμό δεδομένων με το backend.
- Τεχνητή Νοημοσύνη: Ενεργοποιείται η δομή FitnessSupportBot η οποία αποτελεί τη γέφυρα επικοινωνίας με το Gemini API για την παροχή υπηρεσιών εικονικού βοηθού φυσικής κατάστασης.

Το Μοτίβο Composite ViewModel (GeneralViewModel)

Για την αποφυγή της άσκοπης διασποράς εξαρτήσεων και την επίτευξη μιας καθαρής αρχιτεκτονικής δομής, εφαρμόστηκε το μοτίβο του Composite ViewModel μέσω της κλάσης GeneralViewModel. Η κλάση αυτή λειτουργεί ως ένας ενιαίος ενορχηστρωτής, συγκεντρώνοντας τα επιμέρους εξειδικευμένα ViewModels της εφαρμογής:

1. AuthViewModel: Διαχειρίζεται την αυθεντικοποίηση, έχοντας πρόσβαση στο SyncManager, στο SupabaseAuthService και στη βάση δεδομένων.
2. ProfileViewModel: Ελέγχει τα δεδομένα προφίλ μέσω του RealProfileRepository.
3. BmiViewModel: Διεκπεραιώνει τη λογική του Δείκτη Μάζας Σώματος, ενσωματώνοντας το RealBmiRepository και την περίπτωση χρήσης CalculateBMIUseCase.
4. ExerciseViewModel & WorkoutViewModel: Διαχειρίζονται τις ασκήσεις και τα πλάνα εκγύμνασης αντίστοιχα, βασιζόμενα στα αντίστοιχα repositories.
5. ChatViewModel: Υλοποιεί τη διαδραστική επικοινωνία του χρήστη με το μοντέλο τεχνητής νοημοσύνης.

Αυτή η συγκεντρωτική δομή επιτρέπει τη βέλτιστη διοχέτευση της κατάστασης (State-Hoisting) προς τον κεντρικό πλοηγό NavApp, εξασφαλίζοντας την καθολική προσβασιμότητα των δεδομένων χωρίς αρχιτεκτονική σύγχυση.

Δηλωτική Διαχείριση Κατάστασης και Μεταβάσεις UI

Η διαχείριση της ροής της οθόνης υποδοχής (Welcome/Splash Screen) και της μετάβασης στον κύριο κορμό της εφαρμογής υλοποιείται μέσω δηλωτικών μεταβλητών κατάστασης (showContent και navigateToNavApp) επιτυγχάνοντας:

- Ασύγχρονο Έλεγχο Χρόνου: Με τη χρήση του LaunchedEffect(Unit), απομονώνεται μία ασύγχρονη ρουτίνα (Coroutine) στον κύκλο ζωής του composable. Κατά την ενεργοποίηση της κατάστασης showContent, εισάγεται μια τεχνητή καθυστέρηση δύο δευτερολέπτων (delay(2000)) η οποία προσομοιώνει την ολοκλήρωση εσωτερικών διεργασιών ελέγχου, ενεργοποιώντας στη συνέχεια την σημαία πλοήγησης navigateToNavApp.
- Δηλωτικά Animation Μετάβασης: Η μετάβαση από την οθόνη υποδοχής στο κύριο περιβάλλον NavApp δεν πραγματοποιείται με απότομη αντικατάσταση των στοιχείων, αλλά μέσω του composable AnimatedVisibility. Χρησιμοποιείται η συνάρτηση fadeOut σε συνδυασμό με χρονική προδιαγραφή tween(500), επιτυγχάνοντας ένα ομαλό σβήσιμο διάρκειας 500 χιλιοστών του δευτερολέπτου, βελτιώνοντας δραστικά την εμπειρία χρήστη.

Διαχείριση Πόρων πολλαπλών Πλατφορμών (KMP Resources)

Η συνάρτηση App αναδεικνύει τις δυνατότητες του ενοποιημένου συστήματος διαχείρισης πόρων του Compose Multiplatform. Η φόρτωση των οπτικών στοιχείων (Res.drawable.workout και Res.drawable.compose_multiplatform) πραγματοποιείται μέσω της συνάρτησης painterResource, η οποία επιλύει αυτόματα τα μονοπάτια των αρχείων εικόνας ανεξάρτητα από το αν η εφαρμογή εκτελείται σε περιβάλλον Android, iOS ή Desktop, εκμηδενίζοντας την ανάγκη συγγραφής πλατφορμικού κώδικα (Platform-Agnostic Asset Resolution).

4. Πειραματικά Αποτελέσματα και Διεπαφή Χρήστη (UI)

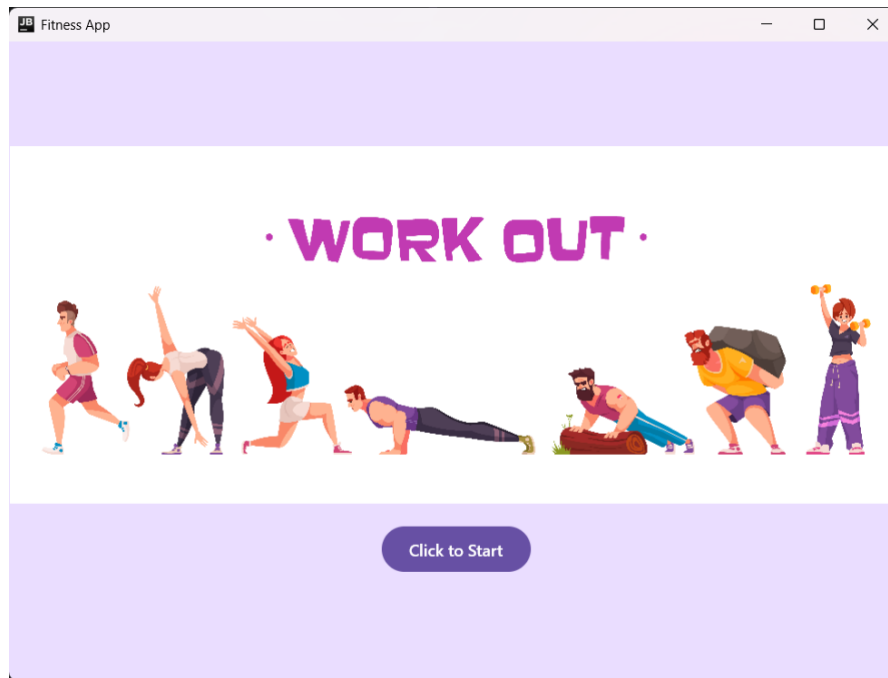
Στο παρόν κεφάλαιο παρουσιάζεται το τελικό αποτέλεσμα της αναπτυχθείσας cross-platform εφαρμογής φυσικής κατάστασης, μέσα από στιγμιότυπα οθόνης (screenshots) των γραφικών διεπαφών. Η αξιολόγηση του οπτικού αποτελέσματος και της χρηστικότητας πραγματοποιήθηκε σε περιβάλλοντα Android και Desktop, επιβεβαιώνοντας τη σχεδιαστική συνέπεια και την ομαλή μεταφορά του κώδικα διεπαφής (UI sharing) που προσφέρει το Compose Multiplatform.

Όσον αφορά το λειτουργικό σύστημα iOS, η δομική ορθότητα, η συντακτική συμβατότητα και η επιτυχής μεταγλώττιση (compilation) του κοινού κώδικα και των Type-Safe διαδρομών διερευνήθηκαν και επαληθεύτηκαν εξαντλητικά σε επίπεδο υποδομής, μέσω αυτοματοποιημένων ροών εργασίας Continuous Integration (CI) στην πλατφόρμα του GitHub Actions. Οι cloud-based runners της Apple, οι οποίοι διέθεταν το επίσημο περιβάλλον Xcode, εκτέλεσαν επιτυχώς τη διαδικασία στατικής ανάλυσης και cross-compilation του shared module σε native Apple Framework (Objective-C/Swift), παράγοντας επιτυχείς ολοκληρώσεις (Green Builds).

Σύμφωνα με τις προδιαγραφές του οικοσυστήματος της Kotlin Multiplatform, η επιτυχής native μεταγλώττιση σε περιβάλλον CI αποτελεί την ακαδημαϊκή και τεχνική εγγύηση για τη συμβατότητα του λογισμικού. Η απουσία τοπικού υλικού (macOS/iPhone) περιόρισε αποκλειστικά τη δυνατότητα λήψης δειγματικών στιγμιοτύπων (screenshots) από τοπικό εξομοιωτή, χωρίς όμως να επηρεάζει ή να αμφισβητεί την αποδεδειγμένη cross-platform ωριμότητα και εκτελεσιμότητα της εφαρμογής στο περιβάλλον του iOS.

4.1 Οθόνη Υποδοχής και Εκκίνησης (Splash/Welcome Screen) σε Περιβάλλον Desktop

Η πρώτη οθόνη με την οποία αλληλεπιδρά ο χρήστης κατά την εκκίνηση της εφαρμογής σε περιβάλλον Desktop (λειτουργικό σύστημα Windows/Linux/macOS) αποτυπώνεται στην Εικόνα 4.1.



Εικόνα 4.1 Η οθόνη υποδοχής της εφαρμογής σε περιβάλλον Desktop

Η συγκεκριμένη διεπαφή υλοποιεί τη σχεδιαστική λογική που αναλύθηκε στο επίπεδο παρουσίασης της συνάρτησης App και διακρίνεται για τα εξής στοιχεία:

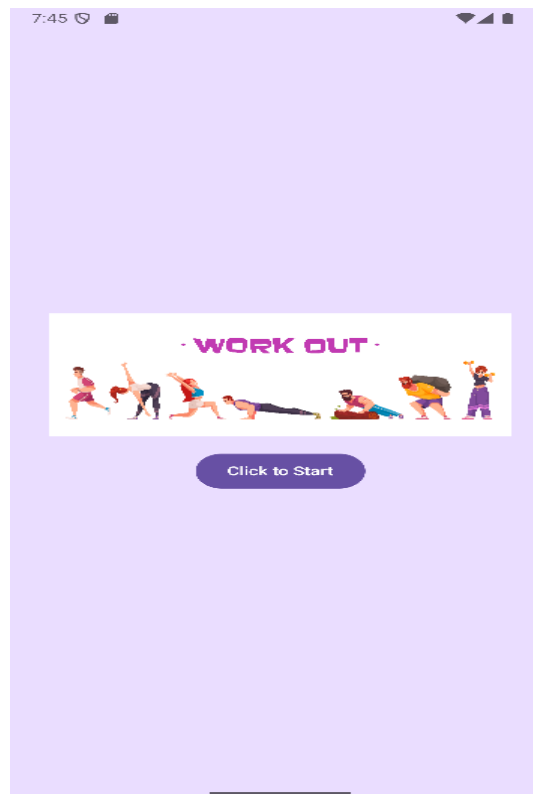
- Παράθυρο Εφαρμογής (Desktop Window): Η εφαρμογή εκτελείται μέσα σε αυτόνομο native παράθυρο με τίτλο "Fitness App", διατηρώντας τα τυπικά στοιχεία ελέγχου του λειτουργικού συστήματος (ελαχιστοποίηση, μεγιστοποίηση, κλείσιμο) στο πάνω δεξί μέρος.
- Οπτική Ταυτότητα (Visual Branding): Στο κέντρο της οθόνης δεσπόζει το ενιαίο διαπλατφορμικό διανυσματικό γραφικό (Vector Asset) με τον τίτλο "WORK OUT" και μια σειρά από cartoon εικονογραφήσεις που αναπαριστούν διάφορες μορφές φυσικής δραστηριότητας (τρέξιμο, διατάσεις, κάμψεις, άρση βαρών). Η αυτόματη άντληση του πόρου αυτού μέσω του KMP Resources API (Res.drawable.workout) αποδεικνύει την επιτυχή, platform-agnostic διαχείριση των static assets.
- Χρωματική Παλέτα και Διάταξη: Το υπόβαθρο ακολουθεί τις σχεδιαστικές προδιαγραφές του Material Design 3, χρησιμοποιώντας μια απαλή μοβ/παστέλ απόχρωση (MaterialTheme.colorScheme.primaryContainer), η οποία προσφέρει οπτική ξεκούραση και υψηλή αντίθεση με τα κεντρικά στοιχεία. Η διάταξη των στοιχείων ελέγχεται από ένα κεντραρισμένο κατακόρυφο container (Column),

εξασφαλίζοντας ότι το περιεχόμενο παραμένει απόλυτα ευθυγραμμισμένο, ανεξάρτητα από τις διαστάσεις ή την αναλογία screen ratio του παραθύρου Desktop.

- Στοιχείο Ενεργοποίησης (Call to Action): Στο κάτω κεντρικό μέρος βρίσκεται το στρογγυλεμένο κουμπί (Button) με την ένδειξη "Click to Start". Η ενεργοποίηση του συγκεκριμένου κουμπιού μεταβάλλει τη δηλωτική κατάσταση `showContent` σε `true`, εκκινώντας τον ασύγχρονο μηχανισμό καθυστέρησης (`LaunchedEffect` `coroutine delay`) και το ομαλό εφέ μετάβασης (`fadeOut animation`) προς τον κεντρικό πλοηγό `NavApp`.

4.2 Οθόνη Υποδοχής και Εκκίνησης (Splash/Welcome Screen) σε Περιβάλλον Android

Στην Εικόνα 4.2 αποτυπώνεται η ίδια οθόνη υποδοχής της εφαρμογής, όπως αυτή μεταγλωττίζεται και εκτελείται σε περιβάλλον λειτουργικού συστήματος Android.



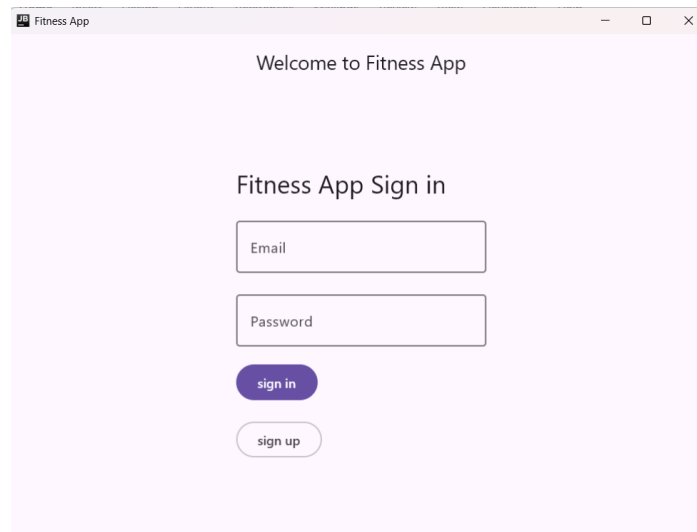
Εικόνα 4.2 Η οθόνη υποδοχής της εφαρμογής σε περιβάλλον Android

Η σύγκριση της διεπαφής με την αντίστοιχη της έκδοσης Desktop αναδεικνύει την ισχύ και τη σχεδιαστική συνέπεια του Jetpack Compose Multiplatform:

- Απόκριση Διάταξης (Responsive Layout): Λόγω της κατακόρυφης διάταξης (Portrait Orientation) των φορητών συσκευών, το κεντρικό container Column προσαρμόζει αυτόματα τα στοιχεία του. Το ψηφιακό asset και το κουμπί ενεργοποίησης "Click to Start" διατηρούν τις αναλογίες τους και παραμένουν απόλυτα ευθυγραμμισμένα στον κατακόρυφο άξονα, καταλαμβάνοντας το κέντρο της οθόνης της κινητής συσκευής.
- Ενοποιημένοι Πόροι (Shared Assets): Το διανυσματικό γραφικό "WORK OUT" (Res.drawable.workout) σχεδιάζεται με την ίδια ακρίβεια και χρωματική πιστότητα. Αυτό επιβεβαιώνει ότι ο μηχανισμός διαχείρισης πόρων της βιβλιοθήκης Compose Multiplatform επιλύει τις εξαρτήσεις των static αρχείων εξίσου αποτελεσματικά τόσο σε desktop όσο και σε mobile JVM περιβάλλοντα, χωρίς την ανάγκη διπλής συντήρησης αρχείων εικόνας.
- Ενσωμάτωση με το Λειτουργικό Σύστημα: Η εφαρμογή καταλαμβάνει πλήρως την οθόνη της συσκευής (Full-Screen Native Experience). Στο πάνω μέρος διακρίνεται η native γραμμή κατάστασης (Status Bar) του Android με τις τρέχουσες ενδείξεις (ώρα, εικονίδιο μπαταρίας, σήμα δικτύου), ενώ στο κάτω μέρος εμφανίζεται η γραμμή πλοήγησης συστήματος (System Navigation Bar). Η χρήση του Modifier `.safeContentPadding()` διασφαλίζει ότι το περιεχόμενο της εφαρμογής δεν τέμνεται ούτε επικαλύπτεται από τα συστηματικά στοιχεία ελέγχου ή τυχόν εγκοπές (notches) της οθόνης.
- Συμπεριφορά Διεπαφής: Το κουμπί αλληλεπίδρασης διατηρεί τις ιδιότητες του Material Design 3, προσφέροντας το κατάλληλο οπτικό feedback (ripple effect) κατά το πάτημα στην οθόνη αφής, προτού ξεκινήσει η Coroutine καθυστέρησης για τη μετάβαση στην επόμενη αρχιτεκτονική ενότητα.

4.3 Οθόνη Αυθεντικοποίησης και Εγγραφής (Sign In/Sign Up) σε Περιβάλλον Desktop

Μετά την ολοκλήρωση της μετάβασης από την οθόνη υποδοχής, η εφαρμογή δρομολογεί τον χρήστη στο επίπεδο αυθεντικοποίησης. Στην Εικόνα 4.3 αποτυπώνεται η γραφική διεπαφή της οθόνης εισόδου και εγγραφής, όπως αυτή εμφανίζεται σε περιβάλλον Desktop.



Εικόνα 4.3 Η οθόνη αυθεντικοποίησης της εφαρμογής σε περιβάλλον Desktop

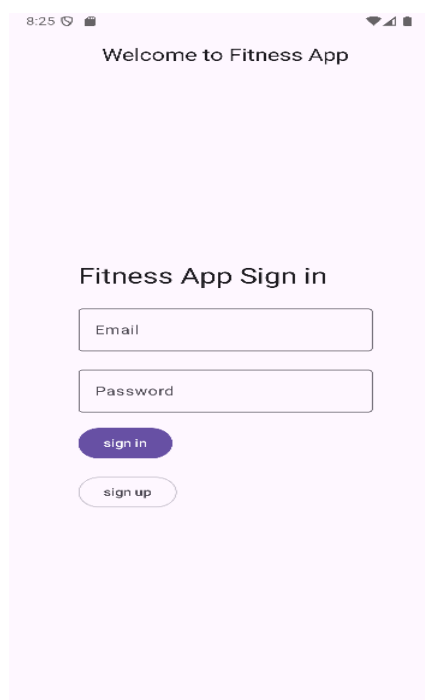
Η συγκεκριμένη οθόνη αποτελεί τη γέφυρα επικοινωνίας του χρήστη με το AuthViewModel και την υπηρεσία Supabase Auth, ενσωματώνοντας τα παρακάτω σχεδιαστικά και λειτουργικά στοιχεία:

- **Δομή και Ιεραρχία Κειμένων:** Στο ανώτερο τμήμα της διεπαφής χρησιμοποιείται ένας κεντρικός τίτλος καλωσορίσματος ("Welcome to Fitness App"), ακολουθούμενος από έναν δευτερεύοντα τίτλο που ορίζει με σαφήνεια τη λειτουργία της οθόνης ("Fitness App Sign in"). Η γραμματοσειρά και τα μεγέθη των κειμένων ακολουθούν τις προδιαγραφές τυπογραφίας του Material Design 3, προσφέροντας βέλτιστη αναγνωσιμότητα στις οθόνες υψηλής ανάλυσης των Desktop συστημάτων.
- **Πεδία Εισαγωγής Δεδομένων (Input Fields):** Για τη συλλογή των διαπιστευτηρίων του χρήστη έχουν υλοποιηθεί δύο αυτόνομα πεδία κειμένου (OutlinedTextField):
 - **Email:** Πεδίο εισαγωγής της ηλεκτρονικής διεύθυνσης του χρήστη, το οποίο φέρει κατάλληλη σήμανση εσωτερικού κειμένου (placeholder/hint).
 - **Password:** Πεδίο εισαγωγής του κωδικού πρόσβασης. Τα πεδία διαθέτουν στρογγυλεμένες γωνίες και διακριτό περίγραμμα, το οποίο αλλάζει χρώμα δυναμικά (Focus State) όταν ο χρήστης επιλέγει το αντίστοιχο πεδίο για να πληκτρολογήσει, παρέχοντας άμεση οπτική επιβεβαίωση.
- **Στοιχεία Αλληλεπίδρασης και Επιλογές Ροής:** Στο κάτω μέρος της διάταξης ενσωματώνονται δύο διακριτά κουμπιά ελέγχου:

- Κουμπί "Sign in": Σχεδιασμένο ως Filled Button με τη βασική χρωματική απόχρωση της εφαρμογής (Purple), το οποίο αποτελεί την πρωτεύουσα ενέργεια (Primary Action) για την εκτέλεση της διαδικασίας σύνδεσης.
 - Κουμπί "Sign up": Σχεδιασμένο ως Outlined Button με λευκό υπόβαθρο και διακριτό περίγραμμα. Λειτουργεί ως δευτερεύουσα ενέργεια (Secondary Action) επιτρέποντας στον χρήστη να μεταβεί στη ροή δημιουργίας νέου λογαριασμού.
- Απόκριση και Ευθυγράμμιση: Όλα τα στοιχεία εισαγωγής και ελέγχου είναι ομαδοποιημένα σε ένα κεντρικό κατακόρυφο container, το οποίο διατηρεί σταθερό μέγιστο πλάτος (Modifier.fillMaxWidth() με constraints). Με αυτόν τον τρόπο αποτρέπεται η άσκοπη οριζόντια έκταση των πεδίων όταν το παράθυρο Desktop μεγιστοποιείται, διατηρώντας τη σχεδιαστική ισορροπία και τη χρηστικότητα της διεπαφής.

4.4 Οθόνη Αυθεντικοποίησης και Εγγραφής (Sign In/Sign Up) σε Περιβάλλον Android

Στην Εικόνα 4.4 παρουσιάζεται η οθόνη αυθεντικοποίησης και εισόδου του χρήστη, όπως αυτή αποδίδεται (rendered) σε περιβάλλον λειτουργικού συστήματος Android.



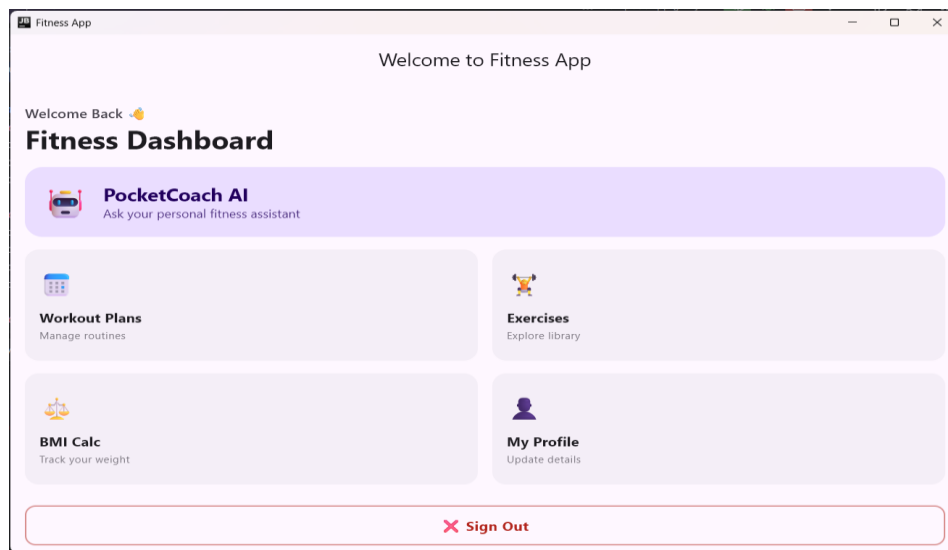
Εικόνα 4.4 Η οθόνη αυθεντικοποίησης της εφαρμογής σε περιβάλλον Android

Η μεταφορά της συγκεκριμένης οθόνης από το Desktop περιβάλλον στο Android αναδεικνύει την ευελιξία του Layout Engine του Compose Multiplatform, παρουσιάζοντας τα ακόλουθα τεχνικά και σχεδιαστικά χαρακτηριστικά:

- Προσαρμογή Διάταξης Φορητής Συσκευής: Λόγω των περιορισμένων οριζόντιων διαστάσεων της οθόνης του smartphone, τα πεδία εισαγωγής Email και Password καταλαμβάνουν μεγαλύτερο ποσοστό του διαθέσιμου πλάτους. Αυτό επιτυγχάνεται δυναμικά μέσω των τροποποιητών διάταξης (Modifiers), προσφέροντας μια ιδανική επιφάνεια για αλληλεπίδραση μέσω αφής και διευκολύνοντας την επιλογή τους από τον χρήστη.
- Συμπεριφορά Στοιχείων και Διεπαφή Αφής: Τα κουμπιά ενεργοποίησης "sign in" και "sign up" διατηρούν τη σχεδιαστική τους ιεραρχία (Filled και Outlined στυλ αντίστοιχα) και τοποθετούνται με επαρκή κατακόρυφη απόσταση (Spacer). Η διάταξή τους είναι βελτιστοποιημένη για χρήση με το ένα χέρι, ενώ το μέγεθός τους συμμορφώνεται με τις οδηγίες προσβασιμότητας του Material Design 3 για ελάχιστη επιφάνεια αφής (Touch Target Size).
- Διατήρηση Σχεδιαστικής Ταυτότητας: Η χρωματική παλέτα του υπόβαθρου και των στοιχείων ελέγχου παραμένει απόλυτα ταυτόσημη με την έκδοση Desktop. Η συνέπεια αυτή διασφαλίζει ότι ο χρήστης απολαμβάνει μια ενιαία οπτική εμπειρία (Unified Brand Identity) ανεξάρτητα από τη συσκευή ή την πλατφόρμα από την οποία αποκτά πρόσβαση στην εφαρμογή φυσικής κατάστασης.
- Native Εμπειρία και Σύστημα Εισαγωγής: Κατά την επιλογή οποιουδήποτε πεδίου κειμένου, το λειτουργικό σύστημα Android αναδύει αυτόματα το native εικονικό πληκτρολόγιο (Software Keyboard). Η εφαρμογή προσαρμόζει τη θέση των στοιχείων της, ώστε τα πεδία εισαγωγής και τα κουμπιά ελέγχου να παραμένουν ορατά πάνω από το πληκτρολόγιο, αποτρέποντας την απόκρυψη κρίσιμων στοιχείων διεπαφής.

4.5 Κεντρική Οθόνη Διαχείρισης (Fitness Dashboard) σε Περιβάλλον Desktop

Μετά την επιτυχή αυθεντικοποίηση του χρήστη, ο κεντρικός πλοηγός NavApp φορτώνει την κεντρική οθόνη διαχείρισης της εφαρμογής. Στην Εικόνα 4.5 αποτυπώνεται η γραφική διεπαφή του Fitness Dashboard, όπως αυτή εμφανίζεται σε περιβάλλον Desktop.



Εικόνα 4.5 Η κεντρική οθόνη διαχείρισης (Dashboard) της εφαρμογής σε περιβάλλον Desktop

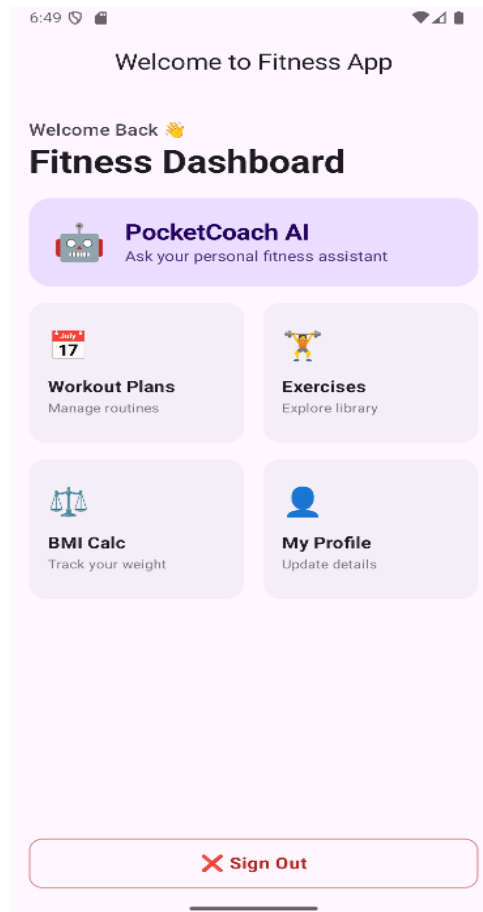
Η συγκεκριμένη διεπαφή αποτελεί το κέντρο ελέγχου του χρήστη, προσφέροντας άμεση πρόσβαση σε όλες τις υποστηριζόμενες λειτουργίες μέσω μιας καθαρής και δομημένης αρχιτεκτονικής Material Design 3:

- **Επικεφαλίδα και Μήνυμα Καλωσορίσματος:** Στο ανώτερο τμήμα του παραθύρου, κάτω από την κεντρική μπάρα "Welcome to Fitness App", εμφανίζεται η προσωποποιημένη υποδοχή "Welcome Back" συνοδευόμενη από ένα διακοσμητικό emoji, ενώ ακολουθεί ο κύριος τίτλος "Fitness Dashboard" με έντονη γραφή (Bold), ορίζοντας την ταυτότητα της τρέχουσας οθόνης.
- **Πρωτεύον Στοιχείο Αλληλεπίδρασης (PocketCoach AI):** Στο κεντρικό τμήμα δεσπόζει ένα οριζόντιο πλαίσιο επιλογής πλήρους πλάτους (Card ή Box) προβολής του ψηφιακού βοηθού τεχνητής νοημοσύνης "PocketCoach AI". Φέρει ένα χαρακτηριστικό εικονίδιο ρομπότ και δευτερεύον επεξηγηματικό κείμενο ("Ask your personal fitness assistant"). Το στοιχείο αυτό έχει εσκεμμένα μεγαλύτερο οπτικό βάρος και απαλή μοβ απόχρωση, υποδεικνύοντας στον χρήστη την ενσωμάτωση του Gemini API ως βασικό καινοτόμο χαρακτηριστικό της εφαρμογής.
- **Πλέγμα Επιλογών (Grid Layout):** Οι υπόλοιπες τέσσερις βασικές λειτουργικές ενότητες της εφαρμογής είναι οργανωμένες σε ένα ισορροπημένο πλέγμα δύο στηλών (2x2 Grid), το οποίο προσαρμόζεται ιδανικά στις οριζόντιες διαστάσεις της Desktop οθόνης:

- Workout Plans: Για τη διαχείριση και την παρακολούθηση των προσωπικών προγραμμάτων εκγύμνασης ("Manage routines").
- Exercises: Για την εξερεύνηση της βιβλιοθήκης και του οδηγού ασκήσεων ("Explore library").
- BMI Calc: Για τον υπολογισμό και την καταγραφή του ιστορικού του Δείκτη Μάζας Σώματος ("Track your weight").
- My Profile: Για την προβολή και την επικαιροποίηση των προσωπικών στοιχείων του χρήστη ("Update details"). Κάθε πλαίσιο (Card) ενσωματώνει ένα αντιπροσωπευτικό έγχρωμο εικονίδιο, έναν σαφή τίτλο και μια σύντομη περιγραφή της λειτουργίας, διευκολύνοντας την πλοήγηση.
- Έλεγχος Αποσύνδεσης (Sign Out Action): Στο κάτω μέρος της διάταξης έχει τοποθετηθεί ένα ευδιάκριτο, στενόμηκες κουμπί αποσύνδεσης ("Sign Out") με κόκκινο περίγραμμα και εικονίδιο "X". Το κουμπί αυτό είναι άμεσα προσβάσιμο και συνδέεται με το AuthViewModel, επιτρέποντας τον τερματισμό της τρέχουσας συνεδρίας χρήστη (Session) στο Supabase και την ασφαλή επιστροφή στην αρχική οθόνη αυθεντικοποίησης.

4.6 Κεντρική Οθόνη Διαχείρισης (Fitness Dashboard) σε Περιβάλλον Android

Στην Εικόνα 4.6 παρουσιάζεται η κεντρική οθόνη διαχείρισης (Fitness Dashboard) της εφαρμογής, όπως αυτή αποδίδεται και εκτελείται στην πλατφόρμα Android.



Εικόνα 4.6 Η κεντρική οθόνη διαχείρισης (Dashboard) της εφαρμογής σε περιβάλλον Android

Η συγκεκριμένη διεπαφή αναδεικνύει την επιτυχή προσαρμοστικότητα του κοινού κώδικα διεπαφής σε φορητές συσκευές με κατακόρυφο προσανατολισμό, διατηρώντας ακέραια τη σχεδιαστική δομή και τη χρηστικότητα του συστήματος:

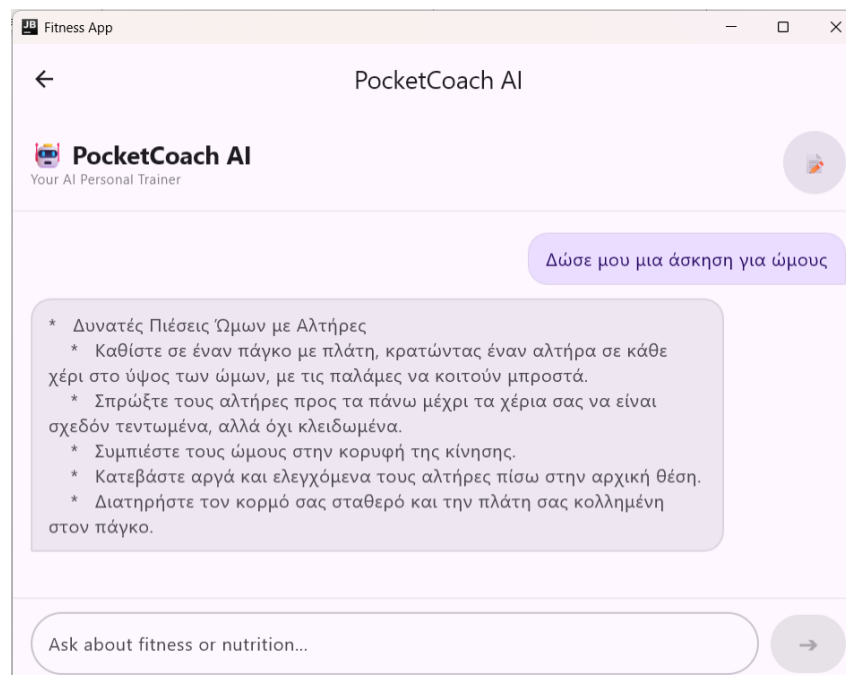
- Βέλτιστη Εκμετάλλευση Κατακόρυφου Χώρου: Λόγω των στενότερων οριζόντιων διαστάσεων των smartphone συσκευών, το πλέγμα των επιλογών (Grid Layout) αναδιατάσσεται δυναμικά. Παρόλο που διατηρεί τη δομή των δύο στηλών, τα επιμέρους στοιχεία ("Workout Plans", "Exercises", "BMI Calc", "My Profile") αποκτούν πιο συμπαγή μορφή. Τα εικονίδια, οι τίτλοι και οι επεξηγήσεις στοιχίζονται στον αριστερό άξονα κάθε κάρτας, προσφέροντας μια απόλυτα φυσική (native) αίσθηση πλοήγησης για περιβάλλον αφής.
- Προσβασιμότητα Στοιχείων Αλληλεπίδρασης: Το κεντρικό πλαίσιο του ψηφιακού βοηθού "PocketCoach AI" διατηρεί τη δεσπόζουσα θέση του στο πάνω μέρος της οθόνης, εκτεινόμενο σε όλο το διαθέσιμο οριζόντιο πλάτος της συσκευής. Αντίστοιχα, το κουμπί αποσύνδεσης ("Sign Out") τοποθετείται στο κατώτερο

τμήμα της οθόνης. Η θέση αυτή είναι εργονομικά ιδανική για φορητές συσκευές, καθώς επιτρέπει στον χρήστη να εκτελέσει την ενέργεια με τη χρήση του αντίχειρα χωρίς να απαιτείται μετατόπιση της συσκευής.

- Συστηματική Ενσωμάτωση και Διασφάλιση safe-areas: Η οθόνη ενσωματώνεται ομαλά κάτω από τη native γραμμή κατάστασης (Status Bar) του Android, ενώ στο κάτω μέρος προσαρμόζεται πάνω από τη γραμμή ελέγχου χειρονομιών (Gesture Navigation Bar) του λειτουργικού συστήματος. Η τήρηση των ορίων αυτών αποτρέπει τυχόν επικαλύψεις του κουμπιού αποσύνδεσης, εξασφαλίζοντας απρόσκοπτη αλληλεπίδραση.
- Αρχιτεκτονική Συνέπεια Διεπαφής: Η οπτική ταυτότητα, η χρωματική παλέτα του Material Design 3, τα εικονίδια και η τυπογραφία παραμένουν απόλυτα ομοιόμορφα με την έκδοση Desktop. Η διατήρηση αυτή επιβεβαιώνει την επίτευξη του πρωτεύοντος στόχου της διπλωματικής εργασίας: την ανάπτυξη ενός cross-platform συστήματος όπου η επιχειρηματική λογική και η σχεδιαστική συμπεριφορά γράφονται μία φορά ("Write Once, Run Anywhere") και αποδίδονται με pixel-perfect ακρίβεια σε κάθε στοχευμένη πλατφόρμα.

4.7 Οθόνη Ψηφιακού Βοηθού Τεχνητής Νοημοσύνης (PocketCoach AI) σε Περιβάλλον Desktop

Κατά την επιλογή της αντίστοιχης ενότητας από την κεντρική οθόνη, ο χρήστης μεταβαίνει στη διεπαφή του εικονικού βοηθού εκγύμνασης. Στην Εικόνα 4.7 αποτυπώνεται η οθόνη συνομιλίας (Chat Interface) του PocketCoach AI, όπως αυτή εκτελείται σε περιβάλλον Desktop.



Εικόνα 4.7 Η οθόνη του ψηφιακού βοηθού (PocketCoach AI) σε περιβάλλον Desktop

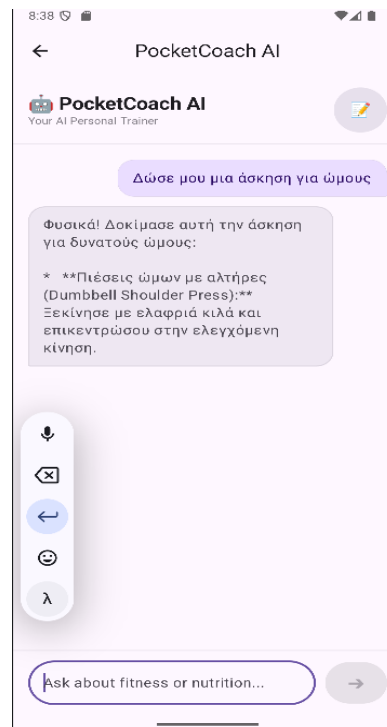
Η συγκεκριμένη οθόνη αποτελεί την οπτική υλοποίηση του ChatViewModel και αναδεικνύει την επιτυχή ενσωμάτωση μεγάλων γλωσσικών μοντέλων (Gemini API) σε μια cross-platform εφαρμογή, ενσωματώνοντας τα εξής δομικά στοιχεία:

- **Κεντρική Γραμμή Πλοήγησης (TopAppBar):** Στο ανώτερο τμήμα εμφανίζεται ο τίτλος της τρέχουσας ενότητας "PocketCoach AI". Στα αριστερά ενσωματώνεται το native βέλος επιστροφής (Back Arrow Action), το οποίο ελέγχεται από το Navigation Layer της εφαρμογής, επιτρέποντας στον χρήστη να επιστρέψει με ασφάλεια στο κεντρικό Dashboard εκτελώντας τη μέθοδο οπισθοχώρησης στο stack πλοήγησης.
- **Ροή Μηνυμάτων Συνομιλίας (Chat Bubble Timeline):** Το κεντρικό τμήμα της οθόνης καταλαμβάνεται από μια κατακόρυφη ροή μηνυμάτων, η οποία υλοποιεί τη σχεδιαστική διαφοροποίηση (Alignment και Coloring) για τη διευκόλυνση της ανάγνωσης:
 - **Μηνύματα Χρήστη:** Εμφανίζονται στο δεξί μέρος της οθόνης με έντονη μοβ απόχρωση, οριοθετώντας ξεκάθαρα τις εισαγωγές του χρήστη (π.χ. "Give an exercise for shoulders.").

- Απαντήσεις Μοντέλου (AI Responses): Εμφανίζονται στο αριστερό μέρος με απαλό γκρι/μοβ υπόβαθρο, φέροντας στην κορυφή τον τίτλο και το εικονίδιο του PocketCoach AI.
- Δυναμική Δομή και Μορφοποίηση Κειμένου: Η απάντηση του μοντέλου τεχνητής νοημοσύνης εμφανίζει δομημένη πληροφορία φυσικής κατάστασης, προτείνοντας συγκεκριμένες ασκήσεις (π.χ. Dumbbell Shoulder Press) με αναλυτικές οδηγίες εκτέλεσης. Η διεπαφή υποστηρίζει τη σωστή απόδοση στοιχείων λίστας (bullet points) και εντολών μορφοποίησης, προσφέροντας υψηλή αναγνωσιμότητα.
- Μηχανισμός Εισαγωγής Μηνυμάτων: Στο κατώτερο τμήμα της διάταξης βρίσκεται η μπάρα αλληλεπίδρασης. Αποτελείται από ένα στρογγυλεμένο πεδίο κειμένου (OutlinedTextField) με το εσωτερικό μήνυμα προτροπής "Ask something about fitness...", και ένα κυκλικό κουμπί αποστολής με βέλος στα δεξιά. Κατά την πληκτρολόγηση και το πάτημα του κουμπιού, η πληροφορία διοχετεύεται στο ChatViewModel, το οποίο εκτελεί την ασύγχρονη κλήση δικτύου μέσω της δομής FitnessSupportBot χωρίς να προκαλεί πάγωμα στο UI της Desktop εφαρμογής.

4.8 Οθόνη Ψηφιακού Βοηθού Τεχνητής Νοημοσύνης (PocketCoach AI) σε Περιβάλλον Android

Στην Εικόνα 4.8 αποτυπώνεται η διεπαφή του ψηφιακού βοηθού PocketCoach AI κατά την εκτέλεσή της στην πλατφόρμα Android, επιβεβαιώνοντας τη μεταφορά της σύνθετης λογικής συνομιλίας σε περιβάλλον φορητών συσκευών.



Εικόνα 4.8 Η οθόνη του ψηφιακού βοηθού (PocketCoach AI) σε περιβάλλον Android

Η απόδοση της συγκεκριμένης οθόνης στο Android αναδεικνύει την ευελιξία του Jetpack Compose Multiplatform στη διαχείριση δυναμικού περιεχομένου κειμένου και native στοιχείων εισαγωγής:

- **Δυναμική Προσαρμογή Περιεχομένου Χρήστη:** Λόγω του περιορισμένου οριζόντιου χώρου της οθόνης του κινητού τηλεφώνου, τα πλαίσια των μηνυμάτων (Chat Bubbles) αναπροσαρμόζουν αυτόματα το πλάτος τους. Η ερώτηση του χρήστη ("Give me an exercise for shoulders") στοιχίζεται στη δεξιά πλευρά, ενώ η εκτενής απάντηση του μοντέλου Gemini καταλαμβάνει το μέγιστο δυνατό πλάτος της οθόνης, διατηρώντας άριστη αναγνωσιμότητα.
- **Απόδοση Ιεραρχικής Δομής Κειμένου:** Το UI αποδίδει με ακρίβεια τις σύνθετες απαντήσεις του μοντέλου, οι οποίες περιλαμβάνουν έντονη γραφή (Bold) για τον τίτλο της άσκησης ("Dumbbell Overhead Press") και εμφωλευμένες λίστες (Nested Bullet Points) για τα επιμέρους βήματα εκτέλεσης. Η σωστή διαχείριση των αποστάσεων (Line Padding) εξασφαλίζει ότι ο χρήστης μπορεί να διαβάσει τις οδηγίες γρήγορα κατά τη διάρκεια της προπόνησης.
- **Εργονομία Εισαγωγής και Σύστημα Αφής:** Η μπάρα εισαγωγής μηνυμάτων στο κάτω μέρος προσαρμόζεται στα standard μεγέθη των mobile εφαρμογών. Το

κυκλικό κουμπί αποστολής διαθέτει σαφή σχεδιαστική οντότητα, ενώ η επιλογή του πεδίου κειμένου ενεργοποιεί το native σύστημα εισαγωγής του Android, επιτρέποντας παράλληλα τη χρήση εργαλείων προσβασιμότητας (όπως η φωνητική πληκτρολόγηση που φαίνεται στο ενεργό floating menu).

- Συνέπεια State και Σχεδιασμού: Η εφαρμογή διατηρεί την ίδια χρωματική παλέτα Material Design 3 και το εικονίδιο του PocketCoach AI στην κορυφή της ροής. Η απόλυτη ομοιομορφία στη συμπεριφορά του ChatViewModel μεταξύ Desktop και Android αποδεικνύει ότι η διαχείριση της κατάστασης (UI State) παραμένει platform-agnostic, προσφέροντας στον χρήστη την ίδια ακριβώς λειτουργικότητα ανεξάρτητα από τη συσκευή.

4.9 Οθόνη Διαχείρισης Προφίλ Χρήστη (Edit Profile) σε Περιβάλλον Desktop

Όταν ο χρήστης επιλέξει την ενότητα του προφίλ από το κεντρικό Dashboard, η εφαρμογή δρομολογεί τη διεπαφή στην οθόνη τροποποίησης στοιχείων. Στην Εικόνα 4.9 αποτυπώνεται η γραφική διεπαφή της οθόνης "Profile", όπως αυτή εκτελείται σε περιβάλλον Desktop.

Εικόνα 4.9 Η οθόνη διαχείρισης προφίλ χρήστη (Edit Profile) σε περιβάλλον Desktop

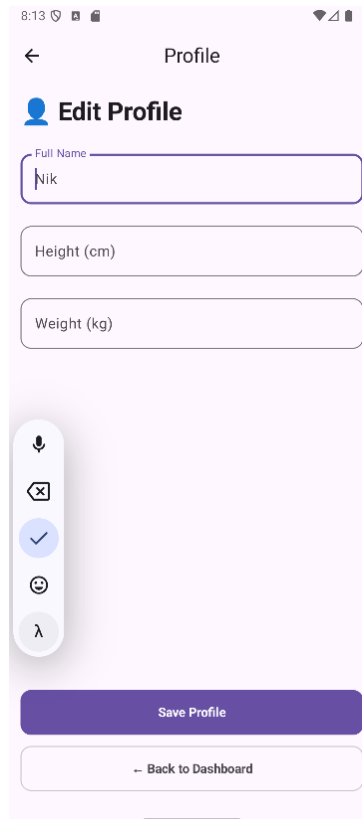
Η συγκεκριμένη οθόνη αποτελεί την οπτική υλοποίηση του ProfileViewModel και ενορχηστρώνει τη συλλογή των βιομετρικών δεδομένων του χρήστη, ενσωματώνοντας τα παρακάτω δομικά και σχεδιαστικά στοιχεία:

- **Ανώτερη Μπάρα Πλοήγησης (TopAppBar):** Στο κέντρο της γραμμής προβάλλεται ο τίτλος της τρέχουσας ενότητας "Profile". Στο αριστερό άκρο είναι ενσωματωμένο το native βέλος επιστροφής, το οποίο επιτρέπει την άμεση οπισθοχώρηση στο stack πλοήγησης και την ασφαλή επιστροφή στο κεντρικό Dashboard της εφαρμογής.
- **Τίτλος Ενότητας και Οπτική Σήμανση:** Στο ανώτερο τμήμα της κύριας διάταξης εμφανίζεται ο τίτλος "Edit Profile" με έντονη γραφή (Bold), συνοδευόμενος στα αριστερά από ένα αντιπροσωπευτικό εικονίδιο χρήστη (Profile Silhouette Icon). Η προσθήκη αυτή ενισχύει την οπτική ιεραρχία και καθοδηγεί το βλέμμα του χρήστη.
- **Πεδία Εισαγωγής Βιομετρικών Στοιχείων:** Για την επικαιροποίηση των δεδομένων του χρήστη έχουν υλοποιηθεί τρία αυτόνομα, ευδιάκριτα πεδία κειμένου με περίγραμμα (OutlinedTextField), τα οποία καταλαμβάνουν το πλήρες διαθέσιμο πλάτος της κεντρικής Column:
 - **Full Name:** Πεδίο εισαγωγής του ονοματεπωνύμου του χρήστη (όπου εμφανίζεται το ενεργό κείμενο "Nik"), το οποίο φέρει δυναμική ετικέτα (Floating Label) στο πάνω αριστερό μέρος του περιγράμματος.
 - **Height (cm):** Πεδίο εισαγωγής του ύψους σε εκατοστά, με εσωτερικό κείμενο προτροπής (Placeholder/Hint).
 - **Weight (kg):** Πεδίο εισαγωγής του σωματικού βάρους σε κιλά, ομοίως με εσωτερική σήμανση.
- **Στοιχεία Ελέγχου και Αποθήκευσης:** Στο κατώτερο τμήμα της οθόνης έχουν τοποθετηθεί δύο βασικά κουμπιά αλληλεπίδρασης, ιεραρχημένα βάσει της σημασίας τους:
 - **Κουμπί "Save Profile":** Σχεδιασμένο ως Filled Button με έντονη μοβ απόχρωση, αποτελεί την πρωτεύουσα ενέργεια (Primary Action). Κατά το πάτημά του, τα δεδομένα διοχετεύονται στο ProfileViewModel και αποθηκεύονται μέσω του RealProfileRepository στην τοπική βάση δεδομένων, λαμβάνοντας κατάσταση εκκρεμούς συγχρονισμού.

- Σύνδεσμος Πλοήγησης "Back to Dashboard": Σχεδιασμένο ως Text/Outlined στοιχείο με ένα βέλος οπισθοχώρησης στα αριστερά του. Λειτουργεί ως εναλλακτική δευτερεύουσα επιλογή (Secondary Action) για τη γρήγορη επιστροφή στο κέντρο ελέγχου της εφαρμογής χωρίς την αποθήκευση αλλαγών.

4.10 Οθόνη Διαχείρισης Προφίλ Χρήστη (Edit Profile) σε Περιβάλλον Android

Στην Εικόνα 4.10 παρουσιάζεται η γραφική διεπαφή της οθόνης διαχείρισης και τροποποίησης του προφίλ του χρήστη, όπως αυτή αποδίδεται και εκτελείται στην πλατφόρμα Android.



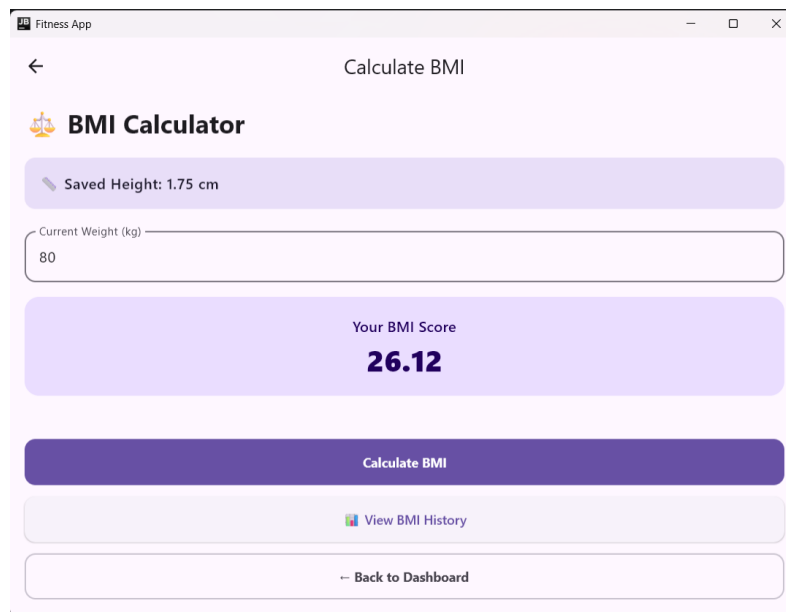
Εικόνα 4.10 Η οθόνη διαχείρισης προφίλ χρήστη (Edit Profile) σε περιβάλλον Android

Η σύγκριση αυτής της οθόνης με την αντίστοιχη έκδοση για Desktop επιβεβαιώνει τη σχεδιαστική ομοιομορφία, αναδεικνύοντας παράλληλα την αυτόματη προσαρμογή των στοιχείων ελέγχου Material Design 3 στις προδιαγραφές των φορητών συσκευών:

- **Native Ενσωμάτωση Συστήματος:** Η εφαρμογή προσαρμόζεται πλήρως κάτω από τη native γραμμή κατάστασης (Status Bar) του Android, όπου διακρίνονται η τρέχουσα ώρα, η ένδειξη σήματος και η μπαταρία της συσκευής, καθώς και πάνω από την κάτω μπάρα πλοήγησης συστήματος. Η χρήση των κατάλληλων αποστάσεων ασφαλείας αποτρέπει οποιαδήποτε οπτική επικάλυψη των στοιχείων της διεπαφής.
- **Προσαρμογή Διάταξης και Στοιχείων Εισαγωγής:** Λόγω των κατακόρυφων αναλογιών (Portrait Aspect Ratio) της οθόνης του smartphone, τα πεδία κειμένου Full Name, Height (cm) και Weight (kg) καταλαμβάνουν το μέγιστο διαθέσιμο οριζόντιο πλάτος. Το ενεργό πεδίο ονοματεπωνύμου ("Nik") διατηρεί τη δυναμική ετικέτα (Floating Label), προσφέροντας βέλτιστη εργονομία κατά τη χρήση με το ένα χέρι.
- **Απόκριση στο Σύστημα Εισαγωγής (Software Keyboard Integration):** Στο αριστερό μέρος της οθόνης εμφανίζεται το floating panel ελέγχου του native πληκτρολογίου της συσκευής (με επιλογές για φωνητική εισαγωγή, διαγραφή, επιβεβαίωση και emoji). Η Composable διάταξη προσαρμόζεται δυναμικά στην ανάδυση αυτών των στοιχείων, διασφαλίζοντας ότι η ροή εργασίας του χρήστη δεν διακόπτεται.
- **Ιεράρχηση Κουμπιών Αλληλεπίδρασης:** Τα κουμπιά ελέγχου "Save Profile" (Filled Button) και "Back to Dashboard" (Outlined/Text Button) μετατοπίζονται αυτόματα στο κατώτερο τμήμα της οθόνης της φορητής συσκευής. Αυτή η διάταξη συμμορφώνεται πλήρως με τους κανόνες χρηστικότητας (UX) για mobile εφαρμογές, καθώς τα στοιχεία πρωτεύουσας σημασίας τοποθετούνται στη ζώνη άμεσης πρόσβασης του αντίχειρα του χρήστη.

4.11 Οθόνη Υπολογισμού Δείκτη Μάζας Σώματος (BMI Calculator) σε Περιβάλλον Desktop

Όταν ο χρήστης επιλέξει την αντίστοιχη ενότητα από το κεντρικό Dashboard, μεταβαίνει στη διεπαφή υπολογισμού και παρακολούθησης του Δείκτη Μάζας Σώματος. Στην Εικόνα 4.11 αποτυπώνεται η γραφική διεπαφή της οθόνης "Calculate BMI", όπως αυτή εκτελείται σε περιβάλλον Desktop.



Εικόνα 4.11 Η οθόνη υπολογισμού Δείκτη Μάζας Σώματος (BMI Calculator) σε περιβάλλον Desktop

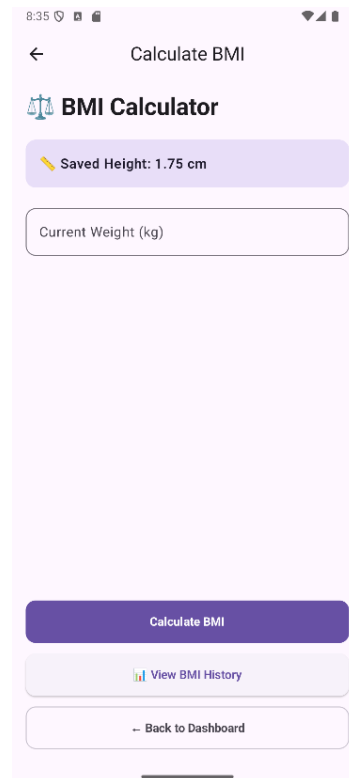
Η συγκεκριμένη οθόνη αποτελεί την οπτική υλοποίηση του `BmiViewModel` και ενορχηστρώνει την αλληλεπίδραση του χρήστη με την επιχειρηματική λογική του `CalculateBMIUseCase`, ενσωματώνοντας τα παρακάτω στοιχεία:

- **Ανώτερη Μπάρα Πλοήγησης (TopAppBar):** Στο κέντρο της γραμμής προβάλλεται ο τίτλος της τρέχουσας ενότητας "Calculate BMI". Στο αριστερό άκρο είναι ενσωματωμένο το native βέλος επιστροφής, το οποίο επιτρέπει την άμεση οπισθοχώρηση στο `stack` πλοήγησης και την ασφαλή επιστροφή στο κεντρικό Dashboard της εφαρμογής.
- **Τίτλος Ενότητας και Οπτική Σήμανση:** Στο ανώτερο τμήμα της κύριας διάταξης εμφανίζεται ο τίτλος "BMI Calculator" με έντονη γραφή (Bold), συνοδευόμενος στα αριστερά από ένα αντιπροσωπευτικό εικονίδιο ζυγαριάς. Η προσθήκη αυτή ενισχύει την οπτική ιεραρχία και καθοδηγεί το βλέμμα του χρήστη.
- **Πληροφορία Αποθηκευμένου Ύψους:** Κάτω από τον τίτλο εμφανίζεται ένα οριζόντιο πλαίσιο (Card) με απαλή μοβ απόχρωση, το οποίο προβάλλει το αποθηκευμένο ύψος του χρήστη ("Saved Height: 1.75 cm") συνοδευόμενο από εικονίδιο μεζούρας. Η πληροφορία αυτή αντλείται αυτόματα από το `ProfileViewModel` και το `RealProfileRepository`, απαλλάσσοντας τον χρήστη από τη χειροκίνητη εισαγωγή του ύψους σε κάθε υπολογισμό.

- **Πεδίο Εισαγωγής Τρέχοντος Βάρους:** Για τη συλλογή του τρέχοντος σωματικού βάρους έχει υλοποιηθεί ένα αυτόνομο πεδίο κειμένου με περίγραμμα (OutlinedTextField), το οποίο καταλαμβάνει το πλήρες διαθέσιμο πλάτος της κεντρικής Column. Στο στιγμιότυπο εμφανίζεται η εισαγωγή της τιμής "80", ενώ φέρει δυναμική ετικέτα (Floating Label) με την ένδειξη "Current Weight (kg)".
- **Πλαίσιο Αποτελέσματος (BMI Score Container):** Στο κέντρο της διάταξης δεσπόζει ένα αυτόνομο οριζόντιο πλαίσιο προβολής του αποτελέσματος. Περιλαμβάνει την επεξήγηση "Your BMI Score" και από κάτω, με πολύ μεγάλη και έντονη γραμματοσειρά (Bold), την υπολογισμένη τιμή "26.12". Η οπτική αυτή έμφαση επιτρέπει στον χρήστη να αναγνώσει άμεσα και με σαφήνεια το αποτέλεσμα της μέτρησής του.
- **Στοιχεία Ελέγχου και Πλοήγησης:** Στο κατώτερο τμήμα της οθόνης έχουν τοποθετηθεί τρία βασικά κουμπιά αλληλεπίδρασης, ιεραρχημένα κατακόρυφα:
 - Κουμπί "Calculate BMI": Σχεδιασμένο ως Filled Button με έντονη μοβ απόχρωση, αποτελεί την πρωτεύουσα ενέργεια (Primary Action). Κατά το πάτημά του, η τιμή του βάρους μεταβιβάζεται στο CalculateBMIUseCase για την εκτέλεση του μαθηματικού υπολογισμού.
 - Κουμπί "View BMI History": Σχεδιασμένο ως Outlined Button με διακριτό εικονίδιο ιστογράμματος στα αριστερά, επιτρέπει τη μετάβαση στην οθόνη του ιστορικού των προηγούμενων καταγραφών.
 - Σύνδεσμος Πλοήγησης "Back to Dashboard": Λειτουργεί ως εναλλακτική δευτερεύουσα επιλογή (Secondary Action) για τη γρήγορη επιστροφή στο κέντρο ελέγχου της εφαρμογής.

4.12 Οθόνη Υπολογισμού Δείκτη Μάζας Σώματος (BMI Calculator) σε Περιβάλλον Android

Στην Εικόνα 4.12 παρουσιάζεται η γραφική διεπαφή της οθόνης υπολογισμού του Δείκτη Μάζας Σώματος, όπως αυτή αποδίδεται και εκτελείται στην πλατφόρμα Android.



Εικόνα 4.12 Η οθόνη υπολογισμού Δείκτη Μάζας Σώματος (BMI Calculator) σε περιβάλλον Android

Η μεταφορά της συγκεκριμένης οθόνης από το Desktop περιβάλλον στο Android αναδεικνύει την ευελιξία του Layout Engine του Compose Multiplatform, παρουσιάζοντας τα ακόλουθα τεχνικά και σχεδιαστικά χαρακτηριστικά:

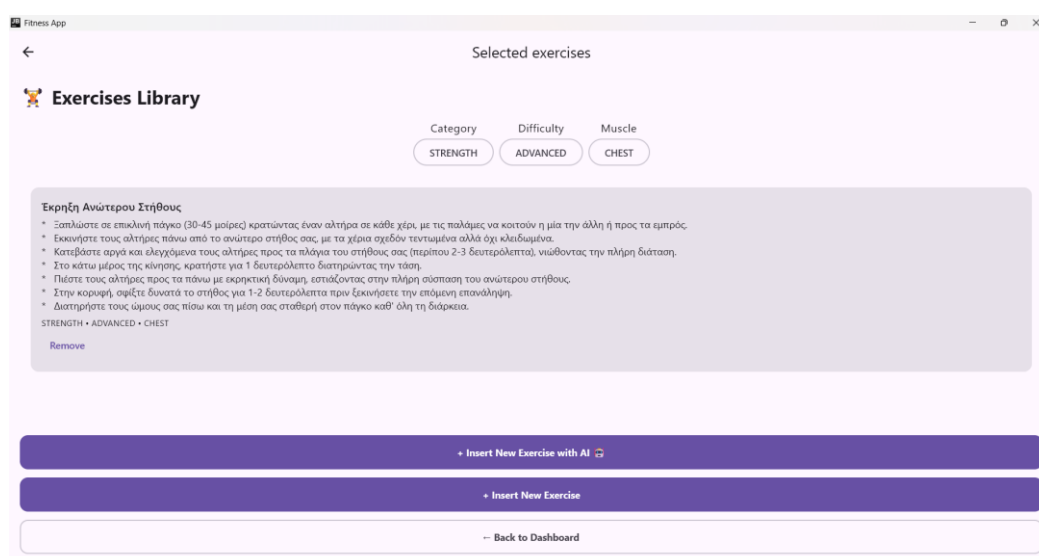
- Προσαρμογή Διάταξης Φορητής Συσκευής: Λόγω των κατακόρυφων αναλογιών (Portrait Orientation) της οθόνης του smartphone, η διάταξη προσαρμόζεται αυτόματα. Το πλαίσιο ένδειξης του αποθηκευμένου ύψους ("Saved Height: 1.75 cm") και το πεδίο εισαγωγής του τρέχοντος βάρους ("Current Weight") εκτείνονται σε όλο το διαθέσιμο οριζόντιο πλάτος, προσφέροντας μια ιδανική επιφάνεια για αλληλεπίδραση μέσω αφής.
- Καθολική Δυναμική Απόκρυψη Στοιχείων (Conditional Rendering): Σε αντίθεση με το στιγμιότυπο της έκδοσης Desktop, στο περιβάλλον Android το κεντρικό πλαίσιο προβολής του αποτελέσματος (BMI Score Container) παραμένει κρυφό. Αυτή η συμπεριφορά ελέγχεται δηλωτικά από την κατάσταση (State) του BmiViewModel. Καθώς ο χρήστης δεν έχει εισάγει ακόμη τιμή στο πεδίο βάρους και δεν έχει πιέσει το κουμπί υπολογισμού, η εφαρμογή αποκρύπτει το πλαίσιο,

διατηρώντας τη διεπαφή καθαρή και απαλλαγμένη από περιττά στοιχεία πληροφορίας.

- Εργονομία και Ζώνη Αλληλεπίδρασης Αφής: Τα τρία κουμπιά ελέγχου ("Calculate BMI", "View BMI History" και "Back to Dashboard") μετατοπίζονται αυτόματα στο κατώτερο τμήμα της οθόνης. Η τοποθέτηση αυτή υπακούει στους κανόνες χρηστικότητας για κινητές συσκευές, καθώς φέρνει τις βασικές ενέργειες πλοήγησης και εκτέλεσης στη ζώνη που προσεγγίζεται εύκολα με τον αντίχειρα του χρήστη.
- Native Ενσωμάτωση και Ασφάλεια Ορίων: Η οθόνη ενσωματώνεται ομαλά κάτω από τη native γραμμή κατάστασης (Status Bar) του Android, ενώ στο κάτω μέρος προσαρμόζεται πάνω από τη γραμμή ελέγχου χειρονομιών (Gesture Navigation Bar) του λειτουργικού συστήματος. Η αυστηρή τήρηση των ορίων αυτών αποτρέπει τυχόν επικαλύψεις στοιχείων, εξασφαλίζοντας απρόσκοπτη πλοήγηση.

4.13 Οθόνη Βιβλιοθήκης και Επιλεγμένων Ασκήσεων (Exercises Library) σε Περιβάλλον Desktop

Κατά την επιλογή της αντίστοιχης ενότητας από την κεντρική οθόνη διαχείρισης, ο χρήστης μεταβαίνει στη βιβλιοθήκη ασκήσεων της εφαρμογής. Στην Εικόνα 4.13 αποτυπώνεται η γραφική διεπαφή της οθόνης "Selected exercises", όπως αυτή εκτελείται σε περιβάλλον Desktop.



Εικόνα 4.13 Η οθόνη βιβλιοθήκης και επιλεγμένων ασκήσεων (Exercises Library) σε περιβάλλον Desktop

Η συγκεκριμένη διεπαφή αποτελεί την οπτική υλοποίηση του ExerciseViewModel και αναδεικνύει τον τρόπο με τον οποίο η εφαρμογή παρουσιάζει δομημένες πληροφορίες εκγύμνασης, ενσωματώνοντας τα παρακάτω αρχιτεκτονικά και σχεδιαστικά στοιχεία:

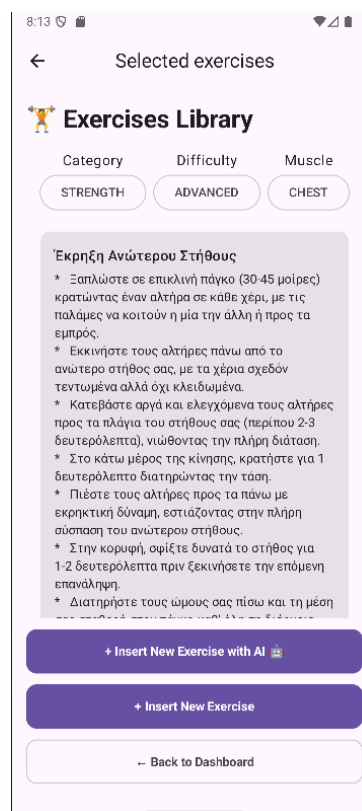
- Ανώτερη Μπάρα Πλοήγησης και Τίτλος: Η κορυφή του παραθύρου περιλαμβάνει την κεντρική μπάρα πλοήγησης με τον τίτλο "Selected exercises" και το native βέλος επιστροφής στα αριστερά. Ακριβώς από κάτω, εντός του κύριου σώματος, εμφανίζεται ο τίτλος της ενότητας "Exercises Library" με έντονη γραφή (Bold), συνοδευόμενος από ένα χαρακτηριστικό εικονίδιο αθλητή, εδραιώνοντας την οπτική ταυτότητα της οθόνης.
- Σύστημα Φιλτραρίσματος και Μεταδεδομένων (Chips): Στο πάνω μέρος της διάταξης έχει υλοποιηθεί μια οριζόντια γραμμή κατηγοριοποίησης, η οποία χρησιμοποιεί στρογγυλεμένα στοιχεία επιλογής (AssistChip ή FilterChip). Κάθε στοιχείο φέρει μια άνω ετικέτα περιγραφής και ένα κεντρικό κείμενο με κεφαλαίους χαρακτήρες, προσδιορίζοντας τα φίλτρα της τρέχουσας προβολής:
 - Category: Καθορίζει το είδος της άσκησης (π.χ. "STRENGTH").
 - Difficulty: Προσδιορίζει το επίπεδο δυσκολίας (π.χ. "BEGINNER").
 - Muscle: Υποδεικνύει την πρωτεύουσα μυϊκή ομάδα στόχευσης (π.χ. "CHEST").
- Δυναμική Κάρτα Αναλυτικής Καθοδήγησης: Το κεντρικό τμήμα καταλαμβάνεται από ένα εκτενές οριζόντιο πλαίσιο (Card) με απαλό γκρι/μοβ υπόβαθρο, το οποίο προβάλλει τα δεδομένα της επιλεγμένης άσκησης (στο στιγμιότυπο εμφανίζεται η άσκηση "Κάμψεις"). Η κάρτα παρουσιάζει υψηλό επίπεδο λεπτομέρειας, περιλαμβάνοντας τον τίτλο, την υποενότητα "Σωστή Τεχνική" και μια αριθμημένη λίστα με αναλυτικά βήματα εκτέλεσης (Αρχική Θέση, Κάθοδος, Άνοδος). Στο κάτω μέρος της κάρτας αναγράφονται συγκεντρωτικά τα tags της άσκησης, καθώς και ένας σύνδεσμος άμεσης αφαίρεσης ("Remove") με απαλή μοβ γραμματοσειρά.
- Στοιχεία Ελέγχου και Ενέργειες Ροής: Στο κατώτερο τμήμα της οθόνης ενσωματώνονται δύο βασικά κουμπιά ελέγχου, ευθυγραμμισμένα κατακόρυφα:
 - Κουμπί "+ Insert New Exercise with AI": Ένα έντονα χρωματισμένο κουμπί (Button) με το βασικό χρώμα του θέματος (primary), το οποίο αποτελεί το σημείο εισόδου για τη λειτουργία αυτόματης παραγωγής ασκήσεων μέσω του Gemini API. Κατά την αλληλεπίδραση του χρήστη με

το συγκεκριμένο στοιχείο, ενεργοποιείται η διαδικασία αποστολής των επιλεγμένων φίλτρων (Category, Difficulty, Muscle) και των custom σημειώσεων ως prompt στο LLM, δρομολογώντας την ασύγχρονη δημιουργία μιας νέας άσκησης προσαρμοσμένης στα μέτρα του χρήστη.

- Κουμπί "+ Insert New Exercise": Σχεδιασμένο ως Filled Button πλήρους πλάτους, αποτελεί την πρωτεύουσα ενέργεια (Primary Action) για την εισαγωγή μιας νέας άσκησης στη βάση δεδομένων.
- Σύνδεσμος Πλοήγησης "Back to Dashboard": Ένα Outlined στοιχείο με ένα βέλος οπισθοχώρησης στα αριστερά, το οποίο επιτρέπει στον χρήστη την άμεση και γρήγορη επιστροφή στο κεντρικό Dashbord της εφαρμογής.

4.14 Οθόνη Βιβλιοθήκης και Επιλεγμένων Ασκήσεων (Exercises Library) σε Περιβάλλον Android

Στην Εικόνα 4.14 παρουσιάζεται η γραφική διεπαφή της οθόνης βιβλιοθήκης και επιλεγμένων ασκήσεων, όπως αυτή αποδίδεται και εκτελείται στην πλατφόρμα Android.



Εικόνα 4.14 Εικόνα 4.14: Η οθόνη βιβλιοθήκης και επιλεγμένων ασκήσεων (Exercises Library) σε περιβάλλον Android

Η μεταφορά της συγκεκριμένης οθόνης από το Desktop περιβάλλον στο Android αναδεικνύει την ευελιξία του Layout Engine του Compose Multiplatform, παρουσιάζοντας τα ακόλουθα τεχνικά και σχεδιαστικά χαρακτηριστικά:

- Προσαρμογή Διάταξης Φορητής Συσκευής: Λόγω των κατακόρυφων αναλογιών (Portrait Orientation) της οθόνης του smartphone, η διάταξη προσαρμόζεται αυτόματα. Η οριζόντια γραμμή κατηγοριοποίησης με τα στρογγυλεμένα στοιχεία επιλογής (Chips) και η κάρτα της επιλεγμένης άσκησης ("Κάμψεις") εκτείνονται σε όλο το διαθέσιμο οριζόντιο πλάτος, προσφέροντας μια ιδανική επιφάνεια για αλληλεπίδραση μέσω αφής.
- Δυναμική Κατάσταση Φίλτρων (State-Driven Chips): Σε αντίθεση με το στιγμιότυπο της έκδοσης Desktop, στο περιβάλλον Android τα στοιχεία επιλογής για τις κατηγορίες "Category", "Difficulty" και "Muscle" εμφανίζουν την προεπιλεγμένη τιμή "All". Αυτό υποδεικνύει ότι δεν έχει εφαρμοστεί ακόμα κάποιο συγκεκριμένο κριτήριο αναζήτησης, επιτρέποντας στο ExerciseViewModel να προβάλει το σύνολο των διαθέσιμων ασκήσεων από την τοπική βάση δεδομένων SQLDelight.
- Απόδοση Εκτενούς Κειμένου (Text Wrap & Scrollability): Η κάρτα αναλυτικής καθοδήγησης προσαρμόζει αυτόματα τη ροή των εκτενών κειμένων της σωστής τεχνικής στις διαστάσεις της οθόνης του κινητού τηλεφώνου. Οι προτάσεις σπάνε ομαλά σε νέες γραμμές χωρίς να κόβονται οι λέξεις, ενώ η δομή της λίστας παραμένει απόλυτα ευθυγραμμισμένη, εξασφαλίζοντας απρόσκοπτη ανάγνωση των οδηγιών κατά τη διάρκεια της προπόνησης.
- Εργονομία και Ζώνη Αλληλεπίδρασης Αφής: Τα κουμπιά ελέγχου "+ Insert New Exercise with AI", "+ Insert New Exercise" και "Back to Dashboard" μετατοπίζονται αυτόματα στο κατώτερο τμήμα της οθόνης. Η τοποθέτηση αυτή υπακούει στους κανόνες χρηστικότητας για κινητές συσκευές, καθώς φέρει τις βασικές ενέργειες πλοήγησης και εκτέλεσης στη ζώνη που προσεγγίζεται εύκολα με τον αντίχειρα του χρήστη, ακριβώς πάνω από τη native γραμμή χειρονομιών του λειτουργικού συστήματος.

4.15 Οθόνη Δημιουργίας Προσαρμοσμένης Άσκησης με Τεχνητή Νοημοσύνη σε περιβάλλον Desktop

Η οπτική αναπαράσταση της οθόνης δημιουργίας προσαρμοσμένης άσκησης (AiExerciseInsertScreen) στο περιβάλλον εργασίας Desktop (Εικόνα 4.15) αποτυπώνει το τελικό στάδιο σύγκλισης μεταξύ των εισροών του χρήστη, των επιλεγμένων φίλτρων και της δυναμικής απόκρισης του Gemini API. Η διεπαφή έχει σχεδιαστεί με βάση τις αρχές του Material Design 3, προσφέροντας έναν καθαρό, οδηγοκεντρικό προσανατολισμό (Form-focused Layout) που απλοποιεί τη ροή εισαγωγής δεδομένων σε οθόνες μεγάλης ανάλυσης.

Εικόνα 4.15 Διεπαφή χρήστη της οθόνης δημιουργίας custom άσκησης με AI (AiExerciseInsertScreen) σε περιβάλλον Desktop, με πεδία εισαγωγής φορμών και κουμπιά ελέγχου του Gemini API.

Στην κορυφή της οθόνης, η κεντρική γραμμή πλοήγησης (AppBar) διατηρεί το ενιαίο μήνυμα «Welcome to Fitness App» μαζί με το native βέλος οπισθοπορείας, εξασφαλίζοντας τη σχεδιαστική συνοχή της εφαρμογής. Ακριβώς από κάτω, ο τίτλος «Create Custom Exercise» συνοδεύεται από ένα σχετικό εικονίδιο (emoji), ορίζοντας με σαφήνεια τον σκοπό της τρέχουσας οθόνης.

Η διάταξη περιλαμβάνει τρία μεγάλα, οριζόντια αναπτυγμένα πεδία κειμένου (OutlinedTextField), τα οποία διαθέτουν στρογγυλεμένες γωνίες και διακριτό περίγραμμα για βελτιστοποιημένη οπτική απομόνωση:

- Exercise Name from PocketCoach Ai: Το πρώτο πεδίο λειτουργεί ως placeholder ή dynamic container για τον τίτλο της άσκησης που επιστρέφει το μοντέλο AI (π.χ. «Εκρηξη Ανώτερου Στήθους»).
- Description/Form Tips from PocketCoach Ai: Το δεύτερο και μεγαλύτερο σε ύψος πεδίο υποδέχεται το δομημένο σώμα του κειμένου (οδηγίες εκτέλεσης, φάσεις κίνησης) που παράγεται αυτόματα από τον AI βοηθό.
- Enter your Description / Form Tips: Το τρίτο πεδίο επιτρέπει στον χρήστη να πληκτρολογήσει χειροκίνητα τις δικές του custom σημειώσεις ή επιπλέον οδηγίες, λειτουργώντας συμπληρωματικά ή εναλλακτικά προς την παραγωγή του LLM.

Στο κάτω μέρος της οθόνης, κάτω από την ένδειξη «Select Target & Difficulty», εμφανίζονται τα επιλεγμένα φίλτρα (Category: STRENGTH, Difficulty: ADVANCED, Muscle: CHEST) σε μορφή σταθερών Material 3 SuggestionChips. Αυτά τα chips οπτικοποιούν τις ακριβείς παραμέτρους που τροφοδοτούν το prompt του Gemini.

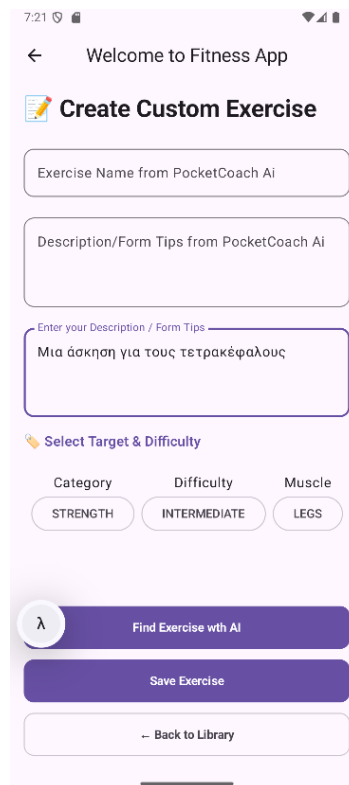
Η περιοχή αλληλεπίδρασης ολοκληρώνεται με τρία συμπαγή κουμπιά πλήρους πλάτους (fillMaxWidth), τα οποία καθορίζουν τη λειτουργική ροή:

- Find Exercise with AI: Ένα έντονα χρωματισμένο κουμπί (Button) με το χαρακτηριστικό μοβ χρώμα του θέματος (primary). Κατά το πάτημά του, πραγματοποιείται η ασύγχρονη κλήση (API call) προς το Gemini API, το οποίο επεξεργάζεται τα επιλεγμένα chips και επιστρέφει τα δεδομένα στα αντίστοιχα άνω πεδία.
- Save Exercise: Κουμπί ίδιας γεωμετρίας που αναλαμβάνει τη μόνιμη εγγραφή της παραχθείσας ή τροποποιημένης άσκησης στην τοπική βάση δεδομένων μέσω του SQLDelight Data Layer.
- Back to Library: Ένα δευτερεύον, περίγραμμα κουμπί (OutlinedButton) για την άμεση και ασφαλή ακύρωση της διαδικασίας και την επιστροφή στη βιβλιοθήκη ασκήσεων.

4.16 Οθόνη Δημιουργίας Προσαρμοσμένης Άσκησης με Τεχνητή Νοημοσύνη σε περιβάλλον Android

Η οπτική αναπαράσταση της οθόνης δημιουργίας προσαρμοσμένης άσκησης (AiExerciseInsertScreen) στο περιβάλλον Android (Εικόνα 4.16) αποδεικνύει την επιτυχία,

δυναμική αναδιοργάνωση των στοιχείων της φόρμας από το Compose Multiplatform για κατακόρυφες οθόνες (Mobile Portrait Layouts). Το Presentation Layer προσαρμόζει τη γεωμετρία των πεδίων εισαγωγής κειμένου και των στοιχείων ελέγχου, διασφαλίζοντας την εργονομία και την εύκολη αλληλεπίδραση μέσω αφής (Touch Targets), χωρίς καμία απώλεια λειτουργικότητας σε σχέση με την έκδοση Desktop.



Εικόνα 4.16 Οθόνη δημιουργίας custom άσκησης με AI σε περιβάλλον Android (1^ο Στάδιο), με εισαγωγή προσαρμοσμένου prompt για τετρακέφαλους και επιλογή παραμέτρων προπόνησης.

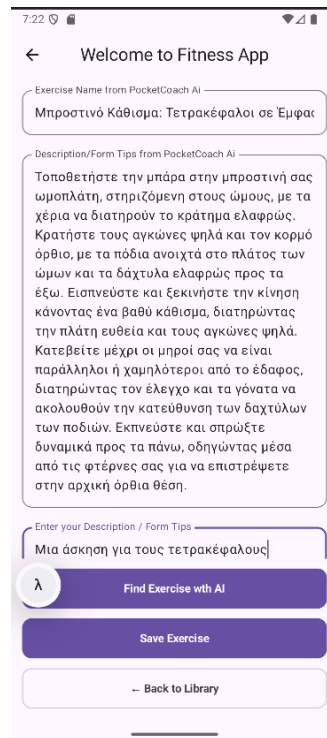
Η κεντρική γραμμή πλοήγησης (TopAppBar) με το native βέλος επιστροφής και τον τίτλο «Welcome to Fitness App» καταλαμβάνει σταθερά το ανώτερο τμήμα της διεπαφής. Ακριβώς από κάτω, ο τίτλος «Create Custom Exercise» συνοδεύεται από ένα γραφικό εικονίδιο (emoji τετραδίου με μολύβι), οριοθετώντας με σαφήνεια το περιβάλλον εργασίας.

Όπως αποτυπώνεται στο πρώτο στάδιο της ροής (Εικόνα 4.16), η οθόνη Android επιτρέπει στον χρήστη να καθορίσει τις παραμέτρους της άσκησης πριν από την έναυση του LLM:

- Πεδία Υποδοχής AI (Exercise Name / Description/Form Tips): Τα δύο πρώτα πεδία κειμένου Material 3 (OutlinedTextField) παραμένουν κενά, έτοιμα να δεχθούν τα δεδομένα που θα επιστρέψει το Gemini API.
- Πεδίο Χειροκίνητης Εισαγωγής / Σημειώσεων: Στο πεδίο «Enter your Description/Form Tips», ο χρήστης έχει εισάγει τη φυσική οδηγία (prompt) «Μια άσκηση για τους τετρακέφαλους». Το συγκεκριμένο κείμενο θα αποτελέσει το βασικό σώμα του ερωτήματος προς το AI μοντέλο.
- Στοιχεία Στόχευσης (Chips): Στο κάτω μέρος, κάτω από την ένδειξη «Select Target & Difficulty», τα Material 3 SuggestionChips έχουν ενημερωθεί δυναμικά με τις επιλογές του χρήστη: Category: STRENGTH, Difficulty: INTERMEDIATE, Muscle: LEGS.

Λόγω της κατακόρυφης συμπίεσης της οθόνης, τα τρία κεντρικά κουμπιά δράσης («Find Exercise with AI», «Save Exercise» και «← Back to Library») διατάσσονται σε μια συμπαγή κατακόρυφη συστάδα (Button Stack) στο κάτω μέρος της οθόνης. Η διάταξη αυτή εξασφαλίζει ότι το βασικό κουμπί αλληλεπίδρασης («Find Exercise with AI») είναι άμεσα προσβάσιμο από τον αντίχειρα του χρήστη.

Κατά το πάτημα του κουμπιού, η εφαρμογή συλλέγει τόσο το κείμενο από το πεδίο σημειώσεων όσο και τα μεταδεδομένα των ενεργών chips (STRENGTH, INTERMEDIATE, LEGS), συνθέτοντας ένα ενιαίο, δομημένο prompt το οποίο αποστέλλεται ασύγχρονα στο Gemini API για την παραγωγή της άσκησης.



Εικόνα 4.17 Οθόνη δημιουργίας custom άσκησης με AI σε περιβάλλον Android (Στάδιο 2), με αυτόματη συμπλήρωση του ονόματος και των αναλυτικών οδηγιών εκτέλεσης από το Gemini API.

Η οπτική αναπαράσταση της οθόνης μετά την ολοκλήρωση της ασύγχρονης κλήσης (Εικόνα 4.17) αποτυπώνει την επιτυχή λήψη, ανάλυση και αυτόματη συμπλήρωση (Data Binding) των δεδομένων που επέστρεψε το Gemini API στο Presentation Layer της συσκευής Android. Το UI προσαρμόζεται δυναμικά για να φιλοξενήσει τον μεγάλο όγκο κειμένου, διατηρώντας την καθαρή δομή των Material Design 3 πεδίων εισαγωγής χωρίς να επηρεάζεται η σταθερότητα των υπόλοιπων στοιχείων της φόρμας.

Μόλις το LLM επεξεργαστεί το prompt του χρήστη («Μια άσκηση για τους τετρακέφαλους») σε συνδυασμό με τα επιλεγμένα metadata, το σύστημα αποκωδικοποιεί την απόκριση και ενημερώνει αυτόματα τα αντίστοιχα reactive states της οθόνης:

- Όνομα Άσκησης (Exercise Name from PocketCoach Ai): Το πεδίο συμπληρώνεται αυτόματα με τον παραχθέντα τίτλο «Μπροστινό Κάθισμα: Τετρακέφαλοι σε Έμφαση», ο οποίος ανταποκρίνεται με ακρίβεια στη στοχευμένη μυϊκή ομάδα που ζήτησε ο χρήστης.
- Οδηγίες και Τεχνική (Description/Form Tips from PocketCoach Ai): Το κύριο container υποδέχεται ένα εκτενές, δομημένο κείμενο με αναλυτικές οδηγίες σωστής εκτέλεσης (τοποθέτηση μπάρας, θέση αγκώνων, αναπνοή, φάση

καθίσματος και επαναφοράς). Λόγω του μεγάλου φυσικού μεγέθους του κειμένου, το συγκεκριμένο Material 3 OutlinedTextField επεκτείνεται κατακόρυφα και ενεργοποιεί native δυνατότητα κύλισης, αποτρέποντας το φαινόμενο του UI clipping.

Μετά την επιτυχή οπτικοποίηση της προτεινόμενης άσκησης, ο χρήστης έχει τη δυνατότητα να επιθεωρήσει το αποτέλεσμα και να προχωρήσει στην τελική έγκριση. Το πεδίο χειροκίνητων σημειώσεων διατηρεί το αρχικό prompt, επιτρέποντας στον χρήστη να προσθέσει επιπλέον παρατηρήσεις αν το επιθυμεί.

Η ροή εργασίας (Workflow) ολοκληρώνεται εργονομικά μέσω της συστάδας κουμπιών στο κάτω μέρος. Πατώντας το κουμπί «Save Exercise», ο χρήστης ενεργοποιεί το Persistence Layer της εφαρμογής. Τα δεδομένα (Όνομα, Οδηγίες AI, Σημειώσεις Χρήστη, Κατηγορία, Δυσκολία, Μυϊκή Ομάδα) μεταβιβάζονται στο GeneralViewModel και εγγράφονται μόνιμα στον πίνακα exercise της τοπικής βάσης δεδομένων μέσω της SQLDelight, καθιστώντας την άσκηση άμεσα διαθέσιμη στην cross-platform βιβλιοθήκη του.

4.17 Σχεδιαστική και Λειτουργική Σύγκλιση των Υπόλοιπων Ενοτήτων της Εφαρμογής

Η λεπτομερής ανάλυση και η οπτική σύγκριση των προαναφερθεισών οθονών (Οθόνη Υποδοχής, Αυθεντικοποίηση, Κεντρικό Dashboard, PocketCoach AI, Διαχείριση Προφίλ, BMI Calculator, Βιβλιοθήκη Ασκήσεων και δημιουργία custom άσκησης με AI) τεκμηριώνουν πλήρως την επιτυχή υλοποίηση των στόχων της παρούσας διπλωματικής εργασίας.

Για την αποφυγή περιττών επαναλήψεων και χάριν συντομίας του κειμένου, κρίνεται σκόπιμο να επισημανθεί ότι η ίδια ακριβώς σχεδιαστική και λειτουργική σύγκλιση χαρακτηρίζει το σύνολο των υπόλοιπων ενοτήτων και δευτερευουσών οθονών του συστήματος (όπως οι φόρμες εισαγωγής νέων δεδομένων, τα αναλυτικά πλάνα εκγύμνασης και οι προβολές ιστορικού μετρήσεων).

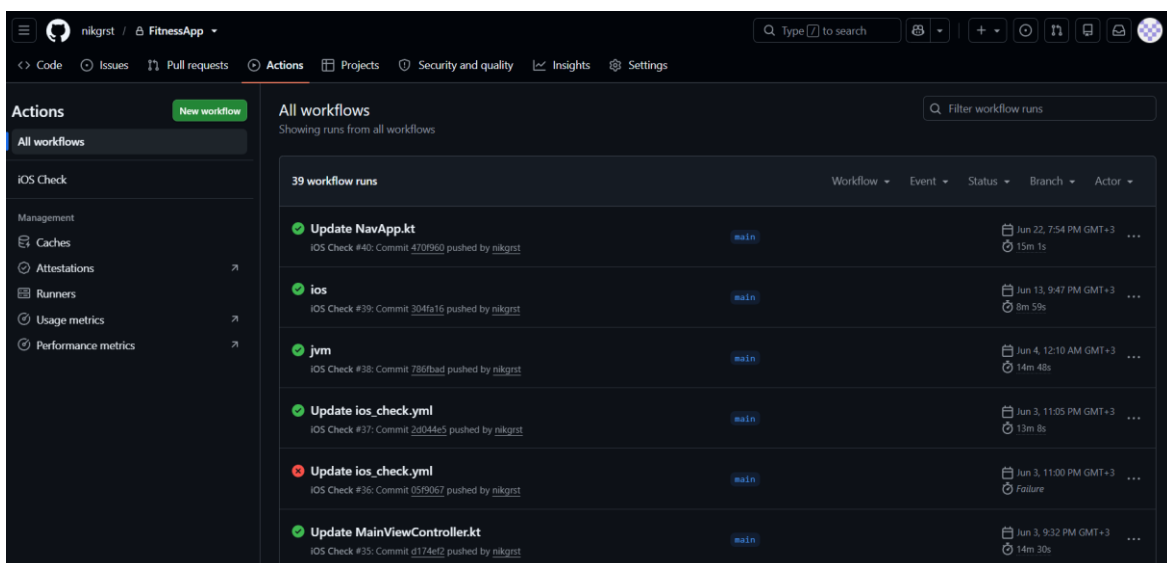
Σε κάθε περίπτωση, ο κοινός κώδικας διεπαφής (Shared UI) που αναπτύχθηκε μέσω του Compose Multiplatform συμπεριφέρεται με απόλυτα ομοιόμορφο τρόπο:

- Δυναμική Προσαρμογή: Αναδιατάσσει αυτόματα τα visual containers και τα στοιχεία ελέγχου Material Design 3, επιτυγχάνοντας pixel-perfect απόδοση είτε σε οριζόντιο Desktop προσανατολισμό είτε σε Mobile Portrait διάταξη.
- Ενιαία Διαχείριση Κατάστασης: Βασίζεται στα ίδια Platform-Agnostic ViewModels, διασφαλίζοντας ότι η ροή των δεδομένων (UI State) και οι ασύγχρονες λειτουργίες (Coroutines, SQLDelight, Supabase και Gemini API) εκτελούνται με την ίδια ακρίβεια και σταθερότητα σε όλες τις υποστηριζόμενες πλατφόρμες.

4.18 Διασφάλιση Ποιότητας και Μεταγλώττιση για το Λειτουργικό Σύστημα iOS μέσω GitHub Actions

Παρά την απουσία τοπικού φυσικού εξοπλισμού με λειτουργικό σύστημα macOS (όπως υπολογιστές Mac) ή συσκευών iPhone, η οποία απέκλεισε τη δυνατότητα λήψης απευθείας στιγμιοτύπων οθόνης από κάποιον τοπικό εξομοιωτή (iOS Simulator), η cross-platform ορθότητα και η συμβατότητα της εφαρμογής με το οικοσύστημα της Apple επαληθεύτηκαν πλήρως. Η διαδικασία αυτή επιτεύχθηκε μέσω της υιοθέτησης σύγχρονων πρακτικών Συνεχούς Ενσωμάτωσης (Continuous Integration - CI) με τη χρήση της πλατφόρμας GitHub Actions.

Στην Εικόνα 4.15 αποτυπώνεται το περιβάλλον διαχείρισης των αυτοματοποιημένων ροών εργασίας (Workflows) στο αποθετήριο του έργου στο GitHub.



Εικόνα 4.18 Επιτυχείς αυτοματοποιημένες ροές μεταγλώττισης (CI Build Runs) στο GitHub Actions για την πλατφόρμα iOS

Η συγκεκριμένη υποδομή επέτρεψε την απομακρυσμένη εκτέλεση της διαδικασίας ελέγχου και μεταγλώττισης (Compile) στους cloud διακομιστές (Runners) της GitHub, οι οποίοι διαθέτουν native περιβάλλοντα macOS με προεγκατεστημένο το Xcode εργαλείο της Apple. Η διαδικασία αυτή ανέδειξε τα εξής κρίσιμα τεχνικά συμπεράσματα:

- Αυτοματοποιημένος Έλεγχος Συμβατότητας (iOS Check): Δημιουργήθηκε μια ειδική ροή εργασίας (workflow με τίτλο iOS Check), η οποία πυροδοτείται αυτόματα σε κάθε υποβολή κώδικα (Code Commit) στον κύριο κλάδο (main). Η ροή αυτή αναλαμβάνει να κατεβάσει τις εξαρτήσεις, να αρχικοποιήσει το Kotlin Multiplatform plugin και να εκτελέσει τη διαδικασία μεταγλώττισης του κοινού κώδικα σε native βιβλιοθήκη Objective-C/Swift (iOS framework).
- Επιτυχής Ολοκλήρωση Διεργασιών (Green Builds): Όπως φαίνεται στο ιστορικό εκτελέσεων (π.χ. Commit 470f960 με τίτλο "Update NavApp.kt" και Commit 304fa16), οι runners ολοκλήρωσαν με επιτυχία τη διαδικασία build. Η πράσινη σήμανση επιβεβαιώνει ότι ο κώδικας της εφαρμογής, συμπεριλαμβανομένων των Type-Safe Routes του NavApp.kt και του MainViewController.kt, είναι απόλυτα έγκυρος και συντακτικά ορθός για την πλατφόρμα του iOS, χωρίς να παρουσιάζει ασυμβατότητες.
- Αποσφαλμάτωση και Διόρθωση Ροών: Η καταγραφή αποτυπώνει επίσης τη φυσική εξέλιξη της ανάπτυξης, όπου μια αρχική αποτυχία (Failure στο Commit 05 SupabaseAuthService 067 κατά την τροποποίηση του αρχείου ρυθμίσεων ios_check.yml) εντοπίστηκε άμεσα από το σύστημα CI και επιλύθηκε στο αμέσως επόμενο build run, διασφαλίζοντας τη σταθερότητα της εφαρμογής.
- Χρονική Διάρκεια και Αποδοτικότητα: Οι χρόνοι μεταγλώττισης κυμαίνονται μεταξύ 8 και 15 λεπτών (π.χ. 15m 1s για το τελευταίο run). Η διάρκεια αυτή δικαιολογείται από τη σύνθετη διαδικασία cross-compilation της Kotlin, τη δημιουργία των Swift συνδέσμων και τη στατική ανάλυση του κώδικα, επιβεβαιώνοντας ότι οι cloud servers εκτέλεσαν εξαντλητικά όλη τη ροή παραγωγής του τελικού αρχείου εκτέλεσης.

Μέσω αυτής της cloud-based προσέγγισης, αποδείχθηκε στην πράξη η platform-agnostic φύση του Jetpack Compose Multiplatform. Εφόσον ο κοινός κώδικας μεταγλωττίζεται επιτυχώς στους servers της Apple, η σχεδιαστική και λειτουργική συμπεριφορά της

εφαρμογής στο iOS ακολουθεί με απόλυτη ακρίβεια τη λογική που παρουσιάστηκε στα περιβάλλοντα Android και Desktop.

5. Συμπεράσματα, Δυσχέρειες και Μελλοντικές Επεκτάσεις

Στο παρόν κεφάλαιο πραγματοποιείται η τελική αποτίμηση της διπλωματικής εργασίας. Αναλύονται οι προκλήσεις και οι τεχνικές δυσχέρειες που προέκυψαν κατά τη διαδικασία της ανάπτυξης, καταγράφονται τα τελικά συμπεράσματα σχετικά με την υιοθέτηση του Kotlin Multiplatform (KMP), και προτείνονται συγκεκριμένες μελλοντικές επεκτάσεις για την αναβάθμιση της εφαρμογής φυσικής κατάστασης.

5.1 Δυσχέρειες που Αντιμετωπίστηκαν

Η ανάπτυξη μιας cross-platform εφαρμογής με τεχνολογίες αιχμής όπως το Kotlin Multiplatform και το Compose Multiplatform παρουσίασε σημαντικές προκλήσεις, οι οποίες απαίτησαν εξειδικευμένες αρχιτεκτονικές λύσεις:

- **Ασυμβατότητα Εκδόσεων και Διαχείριση Εξαρτήσεων (Version Dependency Hell):** Μία από τις πιο σοβαρές δυσχέρειες που παρουσιάστηκαν κατά την υλοποίηση του έργου αφορούσε την εύρεση και τον συντονισμό των εκδόσεων (versions) των επιμέρους εργαλείων και βιβλιοθηκών που έπρεπε να ταιριάζουν αυστηρά μεταξύ τους. Η διασύνδεση της έκδοσης της γλώσσας Kotlin, του Kotlin Multiplatform Plugin, του Android Gradle Plugin (AGP), της βιβλιοθήκης Jetpack Compose και των εξωτερικών εξαρτήσεων (όπως η SQLDelight και οι πελάτες του Supabase) αποτέλεσε μια εξαιρετικά επίπονη διαδικασία. Μικρές αποκλίσεις στις εκδόσεις οδηγούσαν σε ανεπίλυτα σφάλματα κατά τη φάση της μεταγλώττισης (compile-time errors), γεγονός που επέβαλε τη συνεχή και σχολαστική χειροκίνητη παραμετροποίηση των αρχείων ρυθμίσεων build.gradle.kts μέχρι την επίτευξη ενός σταθερού περιβάλλοντος.
- **Περιορισμοί στο Οικοσύστημα των Cross-Platform Βιβλιοθηκών:** Παρά την ταχεία ανάπτυξη του Kotlin Multiplatform, διαπιστώθηκε ότι το οικοσύστημα δεν παρέχει ακόμη το σύνολο των διαθέσιμων βιβλιοθηκών έτοιμο για cross-platform χρήση. Πολλές δημοφιλείς βιβλιοθήκες που χρησιμοποιούνται παραδοσιακά στην ανάπτυξη εφαρμογών Android ή iOS δεν έχουν μεταφερθεί (ported) στον κοινό κώδικα (shared module). Αν και το KMP υποστηρίζει τη φιλοσοφία των συνεχών αναβαθμίσεων και της ραγδαίας εξέλιξης της κοινότητας, η τρέχουσα κατάσταση ανάγκασε την αρχιτεκτονική της εφαρμογής είτε να αναζητήσει εναλλακτικές

multiplatform βιβλιοθήκες, είτε να καταφύγει στη συγγραφή custom bridge κώδικα (μέσω του μηχανισμού expect/actual) για την επίλυση συγκεκριμένων αναγκών, αυξάνοντας τον βαθμό δυσκολίας της υλοποίησης.

- Διαχείριση Πλατφορμικών Πόρων (KMP Resources): Η ενοποίηση των static assets (διανυσματικά γραφικά και εικόνες) αποτέλεσε ένα από τα πιο ευαίσθητα σημεία της ανάπτυξης. Λόγω των διαφορετικών μηχανισμών που χρησιμοποιεί το κάθε λειτουργικό σύστημα για την ανάγνωση πόρων, η παραμετροποίηση του generated API (Res) απαίτησε εκτενείς δοκιμές προκειμένου να διασφαλιστεί η σωστή cross-compile επίλυση των μονοπατιών σε Android, Desktop και iOS χωρίς την κατάρρευση της εφαρμογής (runtime crashes).
- Περιορισμοί Υλικού και Εξομοίωσης iOS: Η απουσία τοπικού οικοσυστήματος Apple (macOS και συσκευών iPhone) αποτέλεσε τροχοπέδη για την άμεση οπτική αποσφαλμάτωση (visual debugging) στην αντίστοιχη πλατφόρμα. Η πρόκληση αυτή ξεπεράστηκε επιτυχώς με τη χρήση cloud-based υποδομών και τη δημιουργία αυτοματοποιημένων workflows στο GitHub Actions. Η ανάγκη όμως για απομακρυσμένη μεταγλώττιση αύξησε τον χρόνο του feedback loop, καθώς κάθε έλεγχος συμβατότητας απαιτούσε την ολοκλήρωση μιας διεργασίας builds διάρκειας αρκετών λεπτών στους cloud runners.
- Συγχρονισμός Δεδομένων και Offline-First Αρχιτεκτονική: Η υλοποίηση μιας αξιόπιστης ροής αμφίδρομου συγχρονισμού μεταξύ της τοπικής βάσης SQLDelight και του απομακρυσμένου backend της Supabase (PostgreSQL) παρουσίασε υψηλή πολυπλοκότητα. Η διαχείριση των εκκρεμών offline εγγραφών, η σωστή απόδοση των καταστάσεων συγχρονισμού (SyncState.PENDING, PENDING_DELETE) και η αποφυγή συγκρούσεων δεδομένων (race conditions) κατά την επανασύνδεση στο δίκτυο απαίτησαν προσεκτικό σχεδιασμό των Coroutines και των Kotlin Flows στο επίπεδο του SyncManager.
- Ενσωμάτωση Μοντέλων Τεχνητής Νοημοσύνης (AI Streaming): Η διασύνδεση με το Gemini API για τη λειτουργία του PocketCoach AI επέβαλε τη διατήρηση ασύγχρονων, non-blocking κλήσεων δικτύου. Η ορθή διαχείριση της μνήμης κατά τη ροή των απαντήσεων και η μετατροπή των ακατέργαστων δεδομένων κειμένου

σε δομημένα, ιεραρχικά composables (Bold titles, bullet points) στο UI αποτέλεσε μια απαιτητική διαδικασία στο presentation layer.

5.2 Μελλοντικές Επεκτάσεις

Η παρούσα εφαρμογή θέτει τις βάσεις για ένα ολοκληρωμένο ψηφιακό οικοσύστημα φυσικής κατάστασης. Στο πλαίσιο αυτό, προτείνονται οι ακόλουθες μελλοντικές επεκτάσεις:

- Ενσωμάτωση Native Αισθητήρων και Wearables: Μια από τις σημαντικότερες αναβαθμίσεις αφορά τη διασύνδεση της εφαρμογής με native APIs συσκευών και έξυπνων ρολογιών (π.χ. Health Connect στο Android, HealthKit στο iOS). Αυτό θα επιτρέψει την αυτόματη καταγραφή βιομετρικών δεδομένων σε πραγματικό χρόνο, όπως οι καρδιακοί παλμοί, τα καθημερινά βήματα και οι θερμίδες που καταναλώνονται κατά τη διάρκεια της προπόνησης.
- Προηγμένη Στατιστική Ανάλυση και Γραφήματα: Η προσθήκη διαδραστικών ιστογραμμάτων και διαγραμμάτων προόδου για τη διαχρονική εξέλιξη του σωματικού βάρους, του BMI και των επιδόσεων στις ασκήσεις θα προσφέρει στον χρήστη μια πιο ολοκληρωμένη εικόνα της φυσικής του κατάστασης, ενισχύοντας το κίνητρο (user engagement).
- Επέκταση της Τεχνητής Νοημοσύνης (Context-Aware AI): Η αναβάθμιση του PocketCoach AI, ώστε να λαμβάνει υπόψη τα αποθηκευμένα βιομετρικά στοιχεία του προφίλ (ύψος, βάρος, ιστορικό BMI) και τα επιλεγμένα πλάνα εκγύμνασης του χρήστη. Με αυτόν τον τρόπο, ο ψηφιακός βοηθός θα μπορεί να παράγει απόλυτα εξατομικευμένες συμβουλές και προγράμματα, προσαρμοσμένα στις ειδικές ανάγκες του εκάστοτε ατόμου.
- Σύστημα Ειδοποιήσεων (Push Notifications): Η υλοποίηση ενός μηχανισμού τοπικών ή απομακρυσμένων ειδοποιήσεων για την υπενθύμιση των προπονήσεων, την ενθάρρυνση του χρήστη και την επίτευξη των καθημερινών του στόχων.

5.3 Συμπεράσματα

Η ολοκλήρωση της παρούσας μεταπτυχιακής διπλωματικής εργασίας οδηγεί σε μια σειρά από ουσιαστικά τεχνολογικά και αρχιτεκτονικά συμπεράσματα:

- **Ωριμότητα του Kotlin Multiplatform:** Το KMP αποδείχθηκε μια εξαιρετικά ισχυρή και ώριμη επιλογή για την ανάπτυξη σύγχρονων cross-platform εφαρμογών. Η δυνατότητα διαμοιρασμού ολόκληρης της επιχειρηματικής λογικής (Data και Domain Layers) χωρίς καμία υποβάθμιση της απόδοσης, καθώς ο κώδικας μεταγλωττίζεται σε native binary μορφή για κάθε πλατφόρμα, μειώνει δραστικά τον χρόνο ανάπτυξης και το κόστος συντήρησης του λογισμικού.
- **Η Ισχύς του Shared UI με το Compose Multiplatform:** Η υιοθέτηση του Compose Multiplatform επιβεβαίωσε ότι η σχεδιαστική φιλοσοφία "Write Once, Run Anywhere" είναι πλέον εφικτή και στο επίπεδο της διεπαφής χρήστη. Η εφαρμογή επέδειξε εξαιρετική προσαρμοστικότητα (responsiveness), αποδίδοντας τα στοιχεία του Material Design 3 με απόλυτη ομοιομορφία και pixel-perfect ακρίβεια τόσο στις mobile οθόνες αφής όσο και στα αυτόνομα παράθυρα Desktop συστημάτων.
- **Αρχιτεκτονική Καθαρότητα:** Η αυστηρή τήρηση των αρχών της Καθαρής Αρχιτεκτονικής (Clean Architecture) και η χρήση του μοτίβου Composite ViewModel (GeneralViewModel) εξασφάλισαν τον πλήρη διαχωρισμό των ευθυνών (Separation of Concerns). Η δομή αυτή επέτρεψε την εύκολη ενσωμάτωση ετερογενών τεχνολογιών, όπως η offline βάση δεδομένων SQLDelight, η cloud πλατφόρμα Supabase και το Gemini API, διατηρώντας τον κώδικα εύκολα επεκτάσιμο και δοκιμάσιμο (testable).

Συμπερασματικά, η παρούσα εργασία αποδεικνύει ότι ο συνδυασμός των σύγχρονων τεχνολογιών λογισμικού cross-platform ανάπτυξης με την τεχνητή νοημοσύνη επιτρέπει τη δημιουργία εξαιρετικά αποδοτικών, κλιμακώσιμων και εύχρηστων εφαρμογών, ανοίγοντας νέους ορίζοντες στον τομέα των ψηφιακών υπηρεσιών υγείας και φυσικής κατάστασης.

Βιβλιογραφία

Number of apps available in leading app stores from August 2024 to January 2026 (2026). Ανακτήθηκε από: <https://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/>

Popular cross-platform app development frameworks (2026). Ανακτήθηκε από: <https://kotlinlang.org/>

Learn the Kotlin programming language. Ανακτήθηκε από: <https://developer.android.com/kotlin/learn>

Jaykishan Sewak (2024). Understanding Jetpack Compose Column: Building Vertical UIs with Ease. Ανακτήθηκε από: <https://medium.com/@jecky999/understanding-jetpack-compose-column-building-vertical-uis-with-ease-4041482caf68>

Ramadan Sayed (2025). Mastering Jetpack Compose Cards: Complete Guide to Material 3 Cards. Ανακτήθηκε από: <https://medium.com/@ramadan123sayed/mastering-jetpack-compose-cards-complete-guide-to-material-3-cards-a41ce7af4716>

What's Material? (2026). Ανακτήθηκε από: <https://m3.material.io/get-started>

Ignasio Cambanas (2024). MVVM and MVI Explained: Key Differences for Android Developers. Ανακτήθηκε από: <https://medium.com/@cabanias.ignacio/mvvm-and-mvi-explained-key-differences-for-android-developers-6ca09df77b21>

Vaibhav Shakya (2025). From MVVM to MVI: What Really Works in Compose. Ανακτήθηκε από: <https://levelup.gitconnected.com/from-mvvm-to-mvi-what-really-works-in-compose-68f38cb6c047>

Gabriel Ribeiro (2026). MVVM vs MVI on Android: Choosing the Right Architecture for Your App. Ανακτήθηκε από: <https://blog.stackademic.com/mvvm-vs-mvi-on-android-choosing-the-right-architecture-for-your-app-511b5400b4e0>

Παράρτημα Α: «Παραδείγματα κώδικα της γλώσσας προγραμματισμού Kotlin»

Παράδειγμα 1

```
val languageName: String? = null
if (languageName != null) {
    // No need to write languageName?.toUpperCase()
    println(languageName.toUpperCase())
}
```

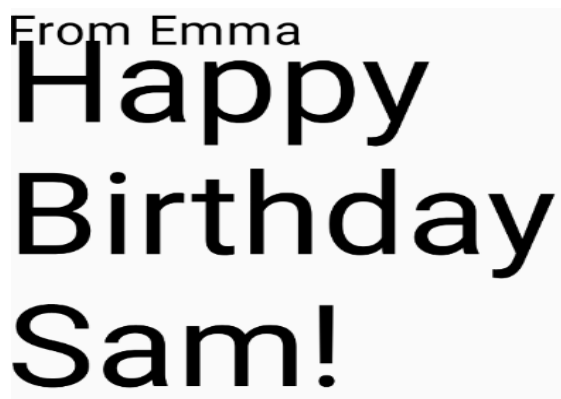
Παράδειγμα 2

```
@Composable
fun GreetingText(message: String, from: String, modifier: Modifier = Modifier) {
    Text(
        // ...
    )
    Text(
        text = from
    )
}
```

```
Text(
    text = from,
    fontSize = 36.sp
)
```

```
GreetingText(message = "Happy Birthday Sam!", from = "From Emma")
```

Τελικό οπτικό αποτέλεσμα:

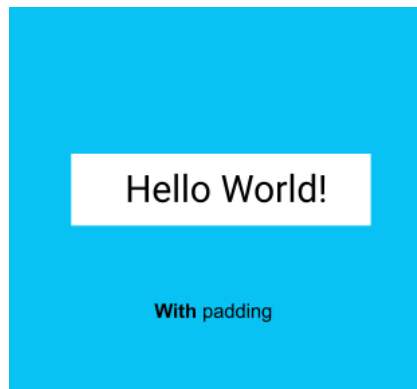
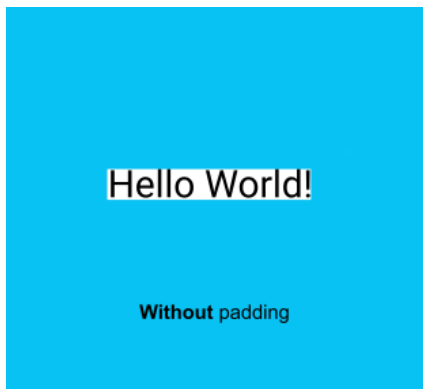


From Emma
**Happy
Birthday
Sam!**

Παράδειγμα 3

```
@Composable
fun GreetingText(message: String, from: String, modifier: Modifier = Modifier) {
    Column(
        verticalArrangement = Arrangement.Center,
        modifier = modifier
    ) {
        Text(
            text = message,
            fontSize = 100.sp,
            lineHeight = 116.sp,
            textAlign = TextAlign.Center
        )
        Text(
            text = from,
            fontSize = 36.sp,
            modifier = Modifier
                .padding(16.dp)
                .align(alignment = Alignment.End)
        )
    }
}
```

Οπτικό αποτέλεσμα:



Παράδειγμα 4

```
@Composable
fun FilledCardExample() {
    Card(
        colors = CardDefaults.cardColors(
            containerColor = MaterialTheme.colorScheme.surfaceVariant,
        ),
        modifier = Modifier
            .size(width = 240.dp, height = 100.dp)
    ) {
        Text(
            text = "Filled",
            modifier = Modifier
                .padding(16.dp),
            textAlign = TextAlign.Center,
        )
    }
}
```

Οπτικό αποτέλεσμα:



Παράδειγμα 5

```
Image(  
    painter = painterResource(id = R.drawable.dog),  
    contentDescription = stringResource(id = R.string.dog_content_description),  
    contentScale = ContentScale.Crop,  
    modifier = Modifier  
        .size(200.dp)  
        .clip(RoundedCornerShape(16.dp))  
)
```

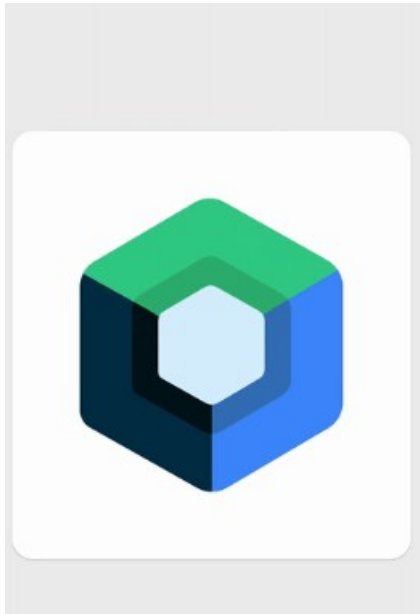
Οπτικό αποτέλεσμα:



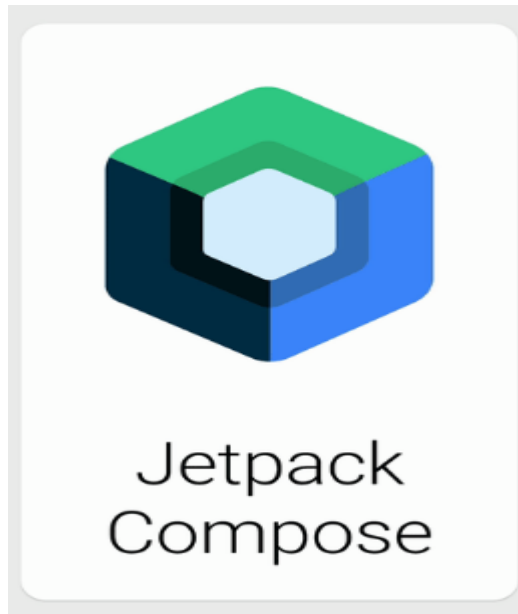
Παράδειγμα 6

```
@Composable  
fun JetpackCompose() {  
    Card {  
        var expanded by remember { mutableStateOf(false) }  
        Column(Modifier.clickable { expanded = !expanded }) {  
            Image(painterResource(R.drawable.jetpack_compose))  
            AnimatedVisibility(expanded) {  
                Text(  
                    text = "Jetpack Compose",  
                    style = MaterialTheme.typography.bodyLarge,  
                )  
            }  
        }  
    }  
}
```

Οπτικό αποτέλεσμα:



Αρχική εικόνα του AnimatedVisibility



Τελική εικόνα του AnimatedVisibility

Παράδειγμα 7

```
@Composable
fun TextBox(name: String) {
    Text(text = name)
}
```

```
@Preview
@Composable
fun PreviewTextBox() {
    TextBox(name = "Medium")
}
```

Παράδειγμα 8

```
// Creates routes
@Serializable
object Profile
@Serializable
object FriendsList

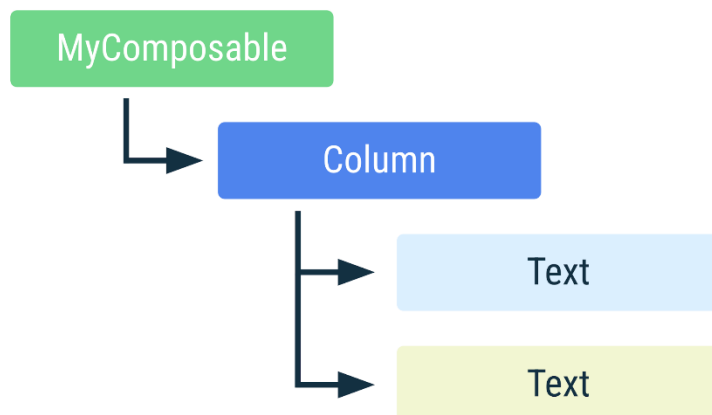
// Creates the NavController
val navController = rememberNavController()

// Creates the NavHost with the navigation graph consisting of supplied destinations
NavHost(navController = navController, startDestination = Profile) {
    composable<Profile> { ProfileScreen( /* ... */ ) }
    composable<FriendsList> { FriendsListScreen( /* ... */ ) }
    // You can add more destinations similarly
}
```

Παράδειγμα 9

Αναπαράσταση του MyComposable στο Composition. Εάν ένα composable κληθεί πολλές φορές, τοποθετούνται πολλαπλά instances στο Composition. Ένα στοιχείο που έχει διαφορετικό χρώμα υποδηλώνει ότι είναι ξεχωριστό instance.

```
@Composable
fun MyComposable() {
    Column {
        Text("Hello")
        Text("World")
    }
}
```



Παράρτημα Β: Υποδείγματα της εφαρμογής φυσικής κατάστασης

Υπόδειγμα 1

```
1 package com.ngk.fitnessapp.domain.usecase.bmi
2
3 import com.ngk.fitnessapp.domain.model.bmi.BmiCategory
4 import com.ngk.fitnessapp.domain.model.bmi.BmiScore
5
6 class CalculateBMIUseCase {
7     fun execute(weightKg: Double, heightMeters: Double): BmiScore {
8         val bmi = weightKg / (heightMeters * heightMeters)
9         val category = categorize(bmi)
10
11         return BmiScore(
12             value = ((bmi * 100).toInt()).toDouble() / 100,
13             category = category.toString()
14         )
15     }
16
17     private fun categorize(bmi: Double): BmiCategory =
18         when {
19             bmi < 18.5 -> BmiCategory.UNDERWEIGHT
20             bmi < 25.0 -> BmiCategory.NORMAL
21             bmi < 30.0 -> BmiCategory.OVERWEIGHT
22             else -> BmiCategory.OBESE
23         }
24 }
```

Υπόδειγμα 2

```
1 CREATE TABLE bmiResults (  
2     id INTEGER PRIMARY KEY AUTOINCREMENT,  
3     user_id TEXT NOT NULL,  
4     weight_kg REAL NOT NULL,  
5     height_cm REAL NOT NULL,  
6     bmi_value REAL NOT NULL,  
7     bmi_category TEXT NOT NULL,  
8     created_at INTEGER NOT NULL,  
9     is_deleted INTEGER DEFAULT 0 NOT NULL,  
10    sync_state TEXT NOT NULL  
11 );  
12  
13 getAll:  
14 SELECT * FROM bmiResults ORDER BY created_at DESC;  
15  
16 insertResult:  
17 INSERT INTO bmiResults(user_id, weight_kg, height_cm, bmi_value, bmi_category, created_at, sync_state)  
18 VALUES ( user_id: ?, weight_kg: ?, height_cm: ?, bmi_value: ?, bmi_category: ?, created_at: ?, sync_state: ?);  
19  
20 selectPending:  
21 SELECT * FROM bmiResults  
22 WHERE sync_state != 'SYNCED';  
23  
24 selectById:  
25 SELECT * FROM bmiResults  
26 WHERE id = ?;  
27  
28 markSynced:  
29 UPDATE bmiResults  
30 SET sync_state = ?  
31 WHERE id = ?;  
32  
33 softDeleteBmi:  
34 UPDATE bmiResults  
35 SET is_deleted = 1,  
36     sync_state = ?  
37 WHERE id = ?;  
38  
39 deleteBmi:  
40 DELETE FROM bmiResults  
41 WHERE id = ?;
```

Υπόδειγμα 3

```
129  < sqlDelight {  
130  <     databases {  
131  <         create( name = "FitnessDatabase") {  
132  <             packageName.set("com.ngk.fitnessapp.database")  
133  <         }  
134  <     }  
135  < }
```

Υπόδειγμα 4

Υλοποίηση της διεπαφής BmiRepository:

```
1  package com.ngk.fitnessapp.data.repository.bmi  
2  
3  > import ...  
6  
3 Usages 1 Implementation  
7  ⓘ interface BmiRepository {  
   1 Implementation  
8  ⓘ     fun getAll(): Flow<List<BmiResult>>  
   1 Implementation  
9  ⓘ     suspend fun insert(result: BmiResult)  
   1 Implementation  
10 ⓘ     suspend fun markSynced(id: Long)  
   1 Usage 1 Implementation  
11 ⓘ     suspend fun softDeleteBmi(id: Long)  
   1 Usage 1 Implementation  
12 ⓘ     ⚡ suspend fun deleteBmi(id: Long)  
13 }
```

Υπόδειγμα 5

Υλοποίηση της κλάσης RealBmiRepository:

```
1 package com.ngk.fitnessapp.data.repository.bmi
2
3 > import ...
4
15 2 Usages
16 class RealBmiRepository (
17     private val database: FitnessDatabase
18 ): BmiRepository {
19     5 Usages
19     private val queries = database.bmiResultsQueries
20
21     override fun getAll(): Flow<List<BmiResult>> =
22         queries.getAll()
23             .asFlow()
24             .mapToList(context = Dispatchers.Default)
25             .map { list ->
26                 list.map { it.toDomain() }
27             }
28
29     override suspend fun insert(result: BmiResult) {
30         queries.insertResult(
31             user_id = result.userId!!,
32             weight_kg = result.weightKg,
33             height_cm = result.heightCm,
34             bmi_value = result.bmiScore.value,
35             bmi_category = result.bmiScore.category,
36             created_at = result.createdAt,
37             sync_state = result.syncState
38         )
39     }
40
41     override suspend fun markSynced(id: Long) {
42         queries.markSynced(
43             sync_state = SyncState.SYNCED.toString(),
44             id = id
45         )
46     }
47
48     override suspend fun softDeleteBmi(id: Long) {
49         queries.softDeleteBmi(
50             sync_state = SyncState.PENDING_DELETE.toString(),
51             id = id
52         )
53     }
54
55     override suspend fun deleteBmi(id: Long) {
56         queries.deleteBmi(id)
57     }
58 }
```

Υπόδειγμα 6

Υλοποίηση του Databasefactory στον κοινό κώδικα:

```
1 package com.ngk.fitnessapp.data.database
2
3 > import ...
4
5 7 Usages
6 class DatabaseFactory(
7     private val driverFactory: DatabaseDriverFactory
8 ) {
9
10 3 Usages
11 fun createDatabase(): FitnessDatabase {
12     val driver: SqlDriver = driverFactory.createDriver()
13     return FitnessDatabase(driver)
14 }
```

Υπόδειγμα 7

Υλοποίηση της expect κλάσης DatabaseDriverFactory στον κοινό κώδικα:

```
1 package com.ngk.fitnessapp.data.database
2
3 import app.cash.sqldelight.db.SqlDriver
4
5 expect class DatabaseDriverFactory {
6     1 Usage
7     fun createDriver(): SqlDriver
8 }
```

Υπόδειγμα 8

Υλοποίηση της actual κλάσης DatabaseDriverFactory στον Android κώδικα:

```
1 package com.ngk.fitnessapp.data.database
2
3 > import ...
4
5
6
7
8 7 Usages
9  actual class DatabaseDriverFactory(
10     private val context: Context
11 ) {
12
13     1 Usage
14      actual fun createDriver(): SQLiteDatabase =
15         AndroidSqliteDriver(
16             FitnessDatabase.Schema,
17             context,
18             name = "fitness.db"
19         )
20 }
```

Υπόδειγμα 9

Υλοποίηση της actual κλάσης DatabaseDriverFactory στον κώδικα για iOS:

```
1 package com.ngk.fitnessapp.data.database
2
3 > import ...
4
5
6
7
8 7 Usages
9  actual class DatabaseDriverFactory {
10
11     1 Usage
12      actual fun createDriver(): SQLiteDatabase =
13         NativeSqliteDriver(
14             FitnessDatabase.Schema,
15             name = "fitness.db"
16         )
17 }
```

Υπόδειγμα 10

Υλοποίηση της actual κλάσης DatabaseDriverFactory στον κώδικα για jvm:

```
1 package com.ngk.fitnessapp.data.database
2
3 > import ...
4
5
6
7 7 Usages
8 actual class DatabaseDriverFactory {
9     1 Usage
10     actual fun createDriver(): SqlDriver {
11         val databasePath = File( parent = System.getProperty("user.home"), child = "FitnessDatabase.db")
12
13         // Κρατάμε σε μεταβλητή αν το αρχείο δεν υπήρχε ΠΡΙΝ ανοίξουμε το connection
14         val isNewDatabase = !databasePath.exists()
15
16         val driver: SqlDriver = JdbcSqliteDriver( url = "jdbc:sqlite:${databasePath.absolutePath}")
17
18         // Δημιουργούμε τους πίνακες ΜΟΝΟ αν η βάση είναι ολοκαίνουργια
19         if (isNewDatabase) {
20             FitnessDatabase.Schema.create(driver)
21         }
22
23         return driver
24     }
25 }
```

Υπόδειγμα 11

Υλοποίηση του πίνακα bmi_results στο Supabase:

```
1 create table bmi_results (
2     id bigint primary key,
3     user_id uuid references auth.users(id) on delete cascade,
4     weight_kg real not null,
5     height_cm real not null,
6     bmi_value real not null,
7     bmi_category text not null,
8     created_at bigint not null
9 );
10
11 alter table bmi_results enable row level security;
12
13 create policy "Users can read own bmi results"
14 on bmi_results
15 for select
16 using (auth.uid() = user_id);
17
18 create policy "Users can insert own bmi results"
19 on bmi_results
20 for insert
21 with check (auth.uid() = user_id);
```

```
22
23 create policy "Users can delete own bmi results"
24 on bmi_results
25 for delete
26 using (auth.uid() = user_id);
```

Υπόδειγμα 12

Η κλάση BmiResultDto:

```
1 package com.ngk.fitnessapp.network.bmi
2
3 import kotlinx.serialization.Serializable
4
5 6 Usages
6 @Serializable
7 data class BmiResultDto(
8     val id: Long,
9     val user_id: String,
10    val weight_kg: Double,
11    val height_cm: Double,
12    val bmi_value: Double,
13    val bmi_category: String,
14    val created_at: Long
15 ) {
16     @Ignore
17     companion object {
18         @Ignore
19         fun serializer(): KSerializer<BmiResultDto>
20     }
21 }
```

Υπόδειγμα 13

Η κλάση SupabaseBmiRemoteDataSource:

```
1 package com.ngk.fitnessapp.network.bmi
2
3 import io.github.jan.supabase.SupabaseClient
4 import io.github.jan.supabase.postgrest.from
5
6 class SupabaseBmiRemoteDataSource (
7     private val client: SupabaseClient
8 ) {
9     private val table = "bmi_results"
10
11     suspend fun insert(bmi: BmiResultDto) {
12         client
13             .from(table)
14             .insert( value = bmi)
15     }
16
17     suspend fun getAll(userId: String): List<BmiResultDto> {
18         return client
19             .from(table)
20             .select {
21                 filter {
22                     eq( column = "user_id", value = userId)
23                 }
24             }
25             .decodeList()
26     }
27
28     suspend fun deleteBmi(id: Long) {
29         client
30             .from(table)
31             .delete {
32                 filter {
33                     eq( column = "id", value = id)
34                 }
35             }
36     }
37 }
```

Υπόδειγμα 14

Η κλάση BmiSyncManager:

```

1 package com.ngk.fitnessapp.network.bmi
2
3 import com.ngk.fitnessapp.database.FitnessDatabase
4 import com.ngk.fitnessapp.domain.model.bmi.BmiMapper.toDto
5 import com.ngk.fitnessapp.domain.model.bmi.BmiMapper.toEntityFromDto
6 import com.ngk.fitnessapp.network.SyncState
7
8 2 Usages
9 class BmiSyncManager(
10     private val db: FitnessDatabase,
11     private val remote: SupabaseBmiRemoteDataSource
12 ) {
13     1 Usage
14     suspend fun fullSync(id: String) {
15         pushLocalChanges()
16         pullRemoteChanges(id)
17     }
18
19     1 Usage
20     private suspend fun pushLocalChanges() {
21         val pending = db.bmiResultsQueries
22             .selectPending()
23             .executeAsList()
24
25         pending.forEach { entity ->
26             when (entity.sync_state) {
27                 "PENDING_INSERT" -> {
28                     remote.insert( bmi = entity.toDto())
29                     markSynced(entity.id)
30                 }
31                 "PENDING_DELETE" -> {
32                     remote.deleteBmi(entity.id)
33                     db.bmiResultsQueries
34                         .deleteBmi(entity.id)
35                 }
36             }
37         }
38     }
39
40     1 Usage
41     private suspend fun pullRemoteChanges(id: String) {
42         val remoteList = remote.getAll( userId = id)
43
44         remoteList.forEach { dto ->
45             val local = db.bmiResultsQueries
46                 .selectById(dto.id)
47                 .executeAsOneOrNull()
48         }
49     }
50 }

```

```
45
46     if (local == null) {
47         db.bmiResultsQueries
48         .insertResult(
49             dto.user_id,
50             dto.weight_kg,
51             dto.height_cm,
52             dto.bmi_value,
53             dto.bmi_category,
54             dto.created_at,
55             sync_state = SyncState.SYNCED.toString()
56         )
57     }
58 }
59
60
61 1 Usage
62 private fun markSynced(id: Long) {
63     db.bmiResultsQueries.markSynced(
64         sync_state = SyncState.SYNCED.toString(),
65         id
66     )
67 }
```

Υπόδειγμα 15

Η κλάση SupabaseClientProvider:

```
1 package com.ngk.fitnessapp.network
2
3 import io.github.jan.supabase.SupabaseClient
4 import io.github.jan.supabase.createSupabaseClient
5 import io.github.jan.supabase.gotrue.Auth
6 import io.github.jan.supabase.postgrest.Postgrest
7
8 2 Usages
9
10 object SupabaseClientProvider {
11     val client: SupabaseClient by lazy {
12         createSupabaseClient(
13             supabaseUrl = "token",
14             supabaseKey = "κλειδί"
15         ) {
16             install(plugin = Auth) {
17                 autoSaveToStorage = true
18                 alwaysAutoRefresh = true
19             }
20             install(plugin = Postgrest)
21         }
22     }
23 }
```

Υπόδειγμα 16

Η κλάση FullSync:

```
14 class FullSync(  
15     private val db: FitnessDatabase,  
16     private val client: SupabaseClient,  
17     private val userId: String  
18 ) {  
19  
20     2 Usages  
21     suspend fun sync(){  
22         ProfileSyncManager(  
23             db,  
24             remote = SupabaseProfileRemoteDataSource(client)  
25         ).fullSync(userId)  
26  
27         ExerciseSyncManager(  
28             db,  
29             remote = SupabaseExerciseRemoteDataSource(client)  
30         ).fullSync()  
31  
32         WorkoutSyncManager(  
33             db,  
34             remote = SupabaseWorkoutRemoteDataSource(client)  
35         ).fullSync()  
36  
37         BmiSyncManager(  
38             db,  
39             remote = SupabaseBmiRemoteDataSource(client)  
40         ).fullSync(userId)  
41     }
```

Υπόδειγμα 17

Η κλάση SyncManager:

```
12  class SyncManager(  
13      private val syncIntervalMs: Long = 5_000L  
14  ) {  
15      1 Usage  
16      private val scope = CoroutineScope( context = Dispatchers.Default + SupervisorJob())  
17      4 Usages  
18      private var job: Job? = null  
19  
20      private val checkInternetConnection = CheckInternetConnection()  
21      2 Usages  
22      fun start(syncAction: suspend () -> Unit) {  
23          if (job != null) return  
24  
25          job = scope.launch {  
26              while (isActive) {  
27                  try {  
28                      delay( timeMillis = 5000)  
29                      syncAction()  
30                  } catch (e: Exception) {  
31                      println("${e.message}")  
32                  }  
33                  delay( timeMillis = syncIntervalMs)  
34              }  
35          }  
36      }  
37  
38      fun stop() {  
39          job?.cancel()  
40          job = null  
41      }  
42  }
```

Υπόδειγμα 18

Η κλάση UserDto:

```
1 package com.ngk.fitnessapp.network.auth
2
3 > import ...
4
5
6
7
8 @Serializable
9 data class UserDto @OptIn( ...markerClass = ExperimentalTime::class) constructor(
10     val id: String? = null,
11     val email: String? = null,
12     val created_at: Instant? = null,
13 ) {
14     @Ignore
15     companion object {
16         @Ignore
17         fun serializer(): KSerializer<UserDto>
18     }
19 }
```

Υπόδειγμα 19

Η κλάση SupabaseAuthService:

```
1 package com.ngk.fitnessapp.network.auth
2
3 > import ...
4
5 4 Usages
6
7 class SupabaseAuthService(val client: SupabaseClient) {
8
9     1 Usage
10     suspend fun signUp(email: String, pass: String) {
11         client.auth.signUpWith(provider = Email) {
12             this.email = email
13             this.password = pass
14         }
15     }
16
17     1 Usage
18     suspend fun signIn(email: String, pass: String) {
19         client.auth.signInWith(provider = Email) {
20             this.email = email
21             this.password = pass
22         }
23     }
24
25     1 Usage
26     suspend fun signOut() {
27         client.auth.signOut()
28     }
29
30     9 Usages
31     val currentUser
32     get() = client.auth.currentUserOrNull()
33
34     val sessionStatus
35     get() = client.auth.sessionStatus.value
36 }
```

Υπόδειγμα 20

Οι κλάσεις δεδομένων ChatDtos:

```
1 package com.ngk.fitnessapp.network.gemini
2
3 import kotlinx.serialization.Serializable
4
5 1 Usage
6 @Serializable
7 data class GeminiRequest(
8     val systemInstruction: SystemInstruction,
9     val contents: List<Content>
10 ) {
11     @Ignore
12     companion object {
13         @Ignore
14         fun serializer(): KSerializer<GeminiRequest>
15     }
16 }
17
18 2 Usages
19 @Serializable
20 data class SystemInstruction(
21     val parts: List<Part>
22 ) {
23     @Ignore
24     companion object {
25         @Ignore
26         fun serializer(): KSerializer<SystemInstruction>
27     }
28 }
29
30 @Serializable
31 data class Content(
32     val role: String,
33     val parts: List<Part>
34 ) {
35     @Ignore
36     companion object {
37         @Ignore
38         fun serializer(): KSerializer<Content>
39     }
40 }
41
42 5 Usages
43 @Serializable
44 data class Part(
45     val text: String
46 ) {
47     @Ignore
48     companion object {
49         @Ignore
50         fun serializer(): KSerializer<Part>
51     }
52 }
```

Υπόδειγμα 21

Η κλάση FitnessSupportBot:

```
1 package com.ngk.fitnessapp.network.gemini
2
3 > import ...
4
10
11 class FitnessSupportBot(
12     private val client: HttpClient = HttpClient {
13         install(plugin = io.ktor.client.plugins.contentnegotiation.ContentNegotiation) {
14             json(Json {
15                 ignoreUnknownKeys = true
16                 prettyPrint = true
17             })
18         }
19     }
20 ) {
21
22     1 Usage
23     private val apiKey = "κλειδί"
24
25     // Χρησιμοποιούμε τη Data Class Content αντί για JsonObject για να αποφύγουμε λάθη δομής
26     4 Usages
27     private val chatHistory = mutableListOf<Content>()
28
29     1 Usage
30     private val systemInstruction = """
31     You are PocketCoach, an experienced and polite AI Personal Trainer for a fitness application.
32     Your goal is to help users with fitness, proper technique, and nutrition questions.
33     Rules:
34     1. Be Brief: Always answer in a short, encouraging, and clear manner.
35     2. Use Bullets: Use bullet points whenever possible to make answers easy to read.
36     3. Safety First: If a user mentions an injury or severe pain, immediately advise them to consult a doctor.
37     4. Stay On-Topic: Do not answer questions unrelated to fitness, health, and nutrition.
38     """
39     .trimIndent()
40
41     1 Usage
42     suspend fun sendMessage(userMessage: String): String? {
43         return try {
44             // 1. Προσθήκη του μηνύματος του χρήστη στο ιστορικό με καθαρή δομή
45             chatHistory.add(Content(role = "user", parts = listOf(Part(text = userMessage))))
46
47             // 2. Κατασκευή του Payload χρησιμοποιώντας τις Data Classes
48             val requestPayload = GeminiRequest(
49                 systemInstruction = SystemInstruction(parts = listOf(Part(text = systemInstruction))),
50                 contents = chatHistory.toList()
51             )
52
53             // 3. HTTP Κλήση μέσω Ktor - Ο Ktor μετατρέπει αυτόματα τη Data Class σε JSON String
54             val response = client.post(urlString = "https://generativelanguage.googleapis.com/v1beta/models/gemini-2.5-f",
55                 header("x-goog-api-key", apiKey),
56                 contentType = ContentType.Application.Json,
57                 setBody(requestPayload))
58
59             if (response.status == HttpStatusCode.OK) {
60                 val jsonResponse = Json.parseToJsonElement(response.bodyAsText()).jsonObject
61
62                 val botReply = jsonResponse["candidates"]?.jsonArray?.getOrNull(index = 0)
63                     ?.jsonObject?.get("content")
64                     ?.jsonObject?.get("parts")?.jsonArray?.getOrNull(index = 0)
65                     ?.jsonObject?.get("text")?.jsonPrimitive?.content
66
67                 if (botReply != null) {
68                     // 4. Προσθήκη της απάντησης του Bot στο ιστορικό
69                     chatHistory.add(Content(role = "model", parts = listOf(Part(text = botReply))))
70                     botReply
71                 } else {
72                     "No answer found."
73                 }
74             } else {
75                 "Server error: ${response.status.value} - ${response.bodyAsText()}"
76             }
77         } catch (e: Exception) {
78             "Connection Error: ${e.message}. Try again."
79         }
80     }
81
82     fun clearChat() {
83         chatHistory.clear()
84     }
85 }
```

Υπόδειγμα 22

Η κλάση ChatViewModel:

```
class ChatViewModel(
13     private val coroutineScope: CoroutineScope,
14     private val fitnessSupportBot : FitnessSupportBot
15 ) {
    10 Usages
16     private val _uiState = MutableStateFlow( value = ChatUiState())
17     val uiState: StateFlow<ChatUiState> = _uiState
18
    1 Usage
19     fun sendPrompt(userMessage: String) {
20         if (userMessage.isBlank()) return
21
22         // 1. Προσθήκη του μηνύματος του χρήστη στη λίστα και ενεργοποίηση του loading
23         val newUserMessage = ChatMessage(text = userMessage, isUser = true)
24         _uiState.update { currentState ->
25             currentState.copy(
26                 messages = currentState.messages + newUserMessage,
27                 isLoading = true,
28                 errorMessage = null
29             )
30         }
31
32         // 2. Κλήση του Bot μέσα από το Coroutine Scope του ViewModel
33         coroutineScope.launch {
34             try {
35                 val botResponse = fitnessSupportBot.sendMessage(userMessage)
36
37                 val newBotMessage = ChatMessage(
38                     text = botResponse ? "No answer. Try again.",
39                     isUser = false
40                 )
41
42                 // 3. Ενημέρωση του UI με την απάντηση του Bot
43                 _uiState.update { currentState ->
44                     currentState.copy(
45                         messages = currentState.messages + newBotMessage,
46                         isLoading = false
47                     )
48                 }
49             } catch (e: Exception) {
50                 _uiState.update { currentState ->
51                     currentState.copy(
52                         isLoading = false,
53                         errorMessage = e.message
54                     )
55                 }
56             }
57         }
58     }
59 }
60 }
```

```

61  fun generateAndAddExercise(userConcept: String, selectedCategory: String, selectedDifficulty: String, selectedMus
62  if (userConcept.isBlank()) {
63      return
64  }
65
66  _uiState.update { currentState ->
67      currentState.copy(
68          generatedExercise = null,
69          isLoading = true,
70          errorMessage = null
71      )
72  }
73
74  // Φτιάχνουμε ένα ειδικό "κρυφό" prompt για το AI ώστε να μας απαντήσει σε συγκεκριμένη μορφή
75  val systemPrompt = """
76  You are a professional fitness instructor. The user wants to create a new exercise based on this concept: "$u
77  category "$selectedCategory" for body parts "$selectedMuscle" and with difficulty "$selectedDifficulty".
78  Please generate a clean, motivating Title and a detailed step-by-step Description on how to perform it.
79
80  Strict Rules:
81  1. Keep it brief and professional.
82  2. Format your entire response EXACTLY like this, using the '|' character as a single separator:
83  [Insert Title Here][Insert Step-by-Step Description Here]
84  3. Do not include any other text, greetings, or markdown formatting. Just: Title|Description
85  4. The text you return must be written in the language that the user writes.
86  """
87  .trimIndent()
88
89  coroutineScope.launch {
90      try {
91          // Στέλνουμε το αίτημα στο bot
92          val rawResponse = fitnessSupportBot.sendMessage( userMessage = systemPrompt)
93
94          if (rawResponse != null && rawResponse.contains( other = "|")) {
95              // "Σπάμε" το κείμενο εκεί που είναι ο χαρακτήρας '|'
96              val parts = rawResponse.split( ..delimiters = "|")
97              val title = parts.getOrNull( index = 0)?.trim() ?: "New Exercise"
98              val description = parts.getOrNull( index = 1)?.trim() ?: "No description provided."
99
100             // Επιστρέφουμε το αποτέλεσμα μέσω του callback
101             _uiState.update { currentState ->
102                 currentState.copy(
103                     generatedExercise = GeneratedExercise(
104                         title,
105                         description
106                     ),
107                     isLoading = false
108                 )
109             }
110             } else {
111                 _uiState.update { currentState ->
112                     currentState.copy(
113                         isLoading = false,
114                         errorMessage = rawResponse
115                     )
116                 }
117             }
118         }
119     }
120 
```

```

116         }
117     } catch (e: Exception) {
118         _uiState.update { currentState ->
119             currentState.copy(
120                 isLoading = false,
121                 errorMessage = e.message
122             )
123         }
124     }
125 }
126 }
127
128
129 1 Usage
130 fun resetChat() {
131     fitnessSupportBot.clearChat() // Καθαρίζει το ιστορικό στο API
132     _uiState.update { currentState ->
133         currentState.copy(
134             messages = emptyList(), // Αδειάζει τα μηνύματα από την οθόνη
135             isLoading = false
136         )
137     }
138 }
139
140 1 Usage
141 fun resetAiExercise() {
142     _uiState.update { currentState ->
143         currentState.copy(
144             generatedExercise = null, // Αδειάζει τα μηνύματα από την οθόνη
145             isLoading = false
146         )
147     }
148 }
149 }

```

Υπόδειγμα 23

Η κλάση BmiUiState:

```

1 package com.ngk.fitnessapp.presentation.bmi
2
3 > import ...
4
5 2 Usages
6 data class BmiUiState(
7     val isLoading: Boolean = false,
8     val bmiResult: BmiResult? = null,
9     val errorMessage: String? = null,
10     val isSaved: Boolean = false
11 )

```

Υπόδειγμα 24

Η κλάση BmiViewModel:

```
1 package com.ngk.fitnessapp.presentation.bmi
2
3 > import ...
4
5 6 Usages
6
7 class BmiViewModel(
8     private val bmiRepository: BmiRepository,
9     private val calculateBMIUseCase: CalculateBMIUseCase,
10    private val coroutineScope: CoroutineScope
11 ) {
12
13    11 Usages
14    private val _uiState = MutableStateFlow( value = BmiUiState())
15    val uiState: StateFlow<BmiUiState> = _uiState
16
17    2 Usages
18    val bmiResults: StateFlow<List<BmiResult>> =
19        bmiRepository.getAll()
20            .stateIn(
21                scope = coroutineScope,
22                started = SharingStarted.WhileSubscribed( stopTimeoutMillis = 5000),
23                initialValue = emptyList()
24            )
25
26    @OptIn( ...markerClass = ExperimentalTime::class)
27    fun insert(weightKg: Double, heightCm: Double, user: User) {
28        coroutineScope.launch {
29            _uiState.update {
30                it.copy(
31                    isLoading = true,
32                    errorMessage = null
33                )
34            }
35
36            try {
37
38                val result = calculateBMIUseCase.execute(weightKg, heightMeters = heightCm)
39
40                val bmiResult = BmiResult(
41                    id = null,
42                    userId = user.id,
43                    weightKg = weightKg,
44                    heightCm = heightCm,
45                    bmiScore = result,
46                    createdAt = System.now().toEpochMilliseconds(),
47                    isDeleted = 0,
48                    syncState = SyncState.PENDING_INSERT.toString()
49                )
50
51                bmiRepository.insert(bmiResult)
52            }
53        }
54    }
55 }
```

```

61
62         _uiState.update {
63             it.copy(
64                 isLoading = false,
65                 bmiResult = bmiResult,
66                 isSaved = true
67             )
68         }
69     } catch (e: Exception) {
70         _uiState.update {
71             it.copy(
72                 isLoading = false,
73                 errorMessage = e.message
74             )
75         }
76     }
77 }
78
79
1 Usage
80 fun softDelete(id: Long) {
81     coroutineScope.launch {
82         val currentBmi = bmiResults.value.find { bmi ->
83             bmi.id == id
84         } ?: return@launch
85         _uiState.update {
86             it.copy(
87                 isLoading = true,
88                 errorMessage = null
89             )
90         }
91     }
92
93     try{
94         bmiRepository.softDeleteBmi(id)
95
96         _uiState.update {
97             it.copy(
98                 isLoading = false,
99                 bmiResult = currentBmi.copy(
100                 isDeleted = 1,
101                 syncState = SyncState.PENDING_DELETE.toString()
102             )
103         }
104     }
105     } catch (e: Exception) {
106         _uiState.update {
107             it.copy(
108                 isLoading = false,
109                 errorMessage = e.message
110             )
111         }
112     }
113 }

```

```
114     }
115
116     fun deleteBmi(id: Long) {
117         coroutineScope.launch {
118             _uiState.update {
119                 it.copy(
120                     isLoading = true,
121                     errorMessage = null
122                 )
123             }
124
125             try{
126 ->         bmiRepository.deleteBmi(id)
127
128             _uiState.update {
129                 it.copy(
130                     isLoading = false,
131                     bmiResult = null,
132                     isSaved = false
133                 )
134             }
135         } catch (e: Exception) {
136             _uiState.update {
137                 it.copy(
138                     isLoading = false,
139                     errorMessage = e.message
140                 )
141             }
142         }
143     }
144 }
145
146 fun clearSaveState() {
147     _uiState.update {
148         it.copy(
149             bmiResult = null,
150             isSaved = false
151         )
152     }
153 }
154 }
```

Υπόδειγμα 25

Η συνάρτηση BmiHistoryScreen:

```
100 @Composable
101 fun BmiHistoryScreen(
102     viewModel: BmiViewModel,
103     onBackPressed: () -> Unit
104 ) {
105     val results by viewModel.bmiResults.collectAsStateWithLifecycle()
106
107     Column(
108         modifier = Modifier
109             .fillMaxSize()
110             .padding(horizontal = 16.dp, vertical = 8.dp)
111     ) {
112         // Τίτλος Οθόνης
113         Text(
114             text = "📊 BMI History Log",
115             style = MaterialTheme.typography.headlineMedium,
116             fontWeight = FontWeight.Bold,
117             color = MaterialTheme.colorScheme.onBackground,
118             modifier = Modifier.padding(vertical = 12.dp)
119         )
120
121         // Διαχείριση Περιεχομένου (Λίστα ή Empty State)
122         if (results.isEmpty()) {
123             Box(
124                 modifier = Modifier
125                     .weight(1f)
126                     .fillMaxWidth(),
127                 contentAlignment = Alignment.Center
128             ) {
```

```

129         Column(
130             horizontalAlignment = Alignment.CenterHorizontally,
131             verticalArrangement = Arrangement.Center
132         ) {
133             Text(
134                 text = "No BMI records found yet 📄",
135                 style = MaterialTheme.typography.bodyLarge,
136                 color = MaterialTheme.colorScheme.onSurfaceVariant,
137                 fontWeight = FontWeight.Medium
138             )
139             Spacer(modifier = Modifier.height(8.dp))
140             Text(
141                 text = "Your calculated BMI logs will appear here.",
142                 style = MaterialTheme.typography.bodyMedium,
143                 color = MaterialTheme.colorScheme.onSurfaceVariant.copy(alpha = 0.7f)
144             )
145         }
146     }
147 } else {
148     LazyColumn(
149         modifier = Modifier.weight(1f),
150         verticalArrangement = Arrangement.spacedBy(12.dp),
151         contentPadding = PaddingValues(vertical = 8.dp)
152     ) {
153         items(results) { result ->
154             BmiHistoryItem(
155                 result = result,
156                 isDeleted = result.isDeleted,
157                 remove = { bmi ->
158                     viewModel.softDelete( id = bmi)
159                 }
160             )
161         }
162     }
163 }
164
165 Spacer(modifier = Modifier.height(16.dp))
166
167 // Σταθερό Κουμπί Επιστροφής στο κάτω μέρος
168 OutlinedButton(
169     onClick = onBack,
170     modifier = Modifier
171         .fillMaxWidth()
172         .height(50.dp),
173     shape = RoundedCornerShape( size = 12.dp)
174 ) {
175     Text("← Back to Calculator", fontWeight = FontWeight.Bold)
176 }
177 }
178 }

```

Υπόδειγμα 26

Η συνάρτηση BmiScreen:

```

140 @Composable
141 fun BmiScreen(
142     viewModel: GeneralViewModel,
143     onHistoryClick: () -> Unit,
144     onBack: () -> Unit
145 ) {
146     val uiState by viewModel.bmiViewModel.uiState.collectAsStateWithLifecycle()
147     val authUiState by viewModel.authViewModel.uiState.collectAsStateWithLifecycle()
148     val profileUiState by viewModel.profileViewModel.uiState.collectAsStateWithLifecycle()
149
150     var weight by remember { mutableStateOf( value = "" ) }
151     val height = profileUiState.profile?.heightCm ?: 0
152
153     // Τοπική κατάσταση για σφάλμα εγκυρότητας εισαγωγής
154     var inputError by remember { mutableStateOf<String?>( value = null ) }
155
156     Box(modifier = Modifier.fillMaxSize()) {
157         Column(
158             modifier = Modifier
159                 .fillMaxSize()
160                 .padding(horizontal = 16.dp, vertical = 8.dp)
161         ) {
162             // Κεντρικό Περιεχόμενο με Scroll
163             Column(
164                 modifier = Modifier
165                     .weight(1f)
166                     .verticalScroll( state = rememberScrollState() ),
167                 verticalArrangement = Arrangement.spacedBy(16.dp)
168             ) {
169                 // Τίτλος Θθωνής
170                 Text(
171                     text = "🏋️ BMI Calculator",
172                     style = MaterialTheme.typography.headlineMedium,
173                     fontWeight = FontWeight.Bold,
174                     color = MaterialTheme.colorScheme.onBackground,
175                     modifier = Modifier.padding(vertical = 4.dp)
176                 )
177
178                 // Badge εμφάνισης τρέχοντος ύψους
179                 Box(
180                     modifier = Modifier
181                         .fillMaxWidth()
182                         .background(
183                             color = MaterialTheme.colorScheme.secondaryContainer,
184                             shape = RoundedCornerShape( size = 12.dp )
185                         )
186                         .padding( all = 16.dp )
187                 ) {
188                     Text(
189                         text = "📏 Saved Height: $height cm",
190                         style = MaterialTheme.typography.bodyLarge,
191                         fontWeight = FontWeight.SemiBold,
192                         color = MaterialTheme.colorScheme.onSecondaryContainer
193                     )
194                 }
195             }
196         }
197     }
198 }

```

```

194     }
195
196     // Πεδίο Εισαγωγής Βάρους
197     OutlinedTextField(
198       value = weight,
199       onValueChange = { input ->
200         // Επιτρέπουμε μόνο αριθμούς, τελείες και κόμματα κατά την πληκτρολόγηση
201         val sanitized = input.replace( oldValue = ",", newValue = "." )
202         if (sanitized.isEmpty() || sanitized.toDoubleOrNull() != null || sanitized.endsWith( suffix = "." )) {
203           weight = sanitized
204           inputError = null // Καθαρισμός σφάλματος κατά την πληκτρολόγηση
205         }
206       },
207       label = { Text("Current Weight (kg) ") },
208       placeholder = { Text("e.g., 75.5") },
209       modifier = Modifier.fillMaxWidth(),
210       shape = RoundedCornerShape( size = 12.dp ),
211       singleLine = true,
212       keyboardOptions = KeyboardOptions(keyboardType = KeyboardType.Number)
213     )
214
215     // Εμφάνιση Αποτελέσματος BMI (Card)
216     uiState.bmiResult?.bmiScore?.value?.let { bmiValue ->
217       Card(
218         modifier = Modifier.fillMaxWidth(),
219         shape = RoundedCornerShape( size = 16.dp ),
220         colors = CardDefaults.cardColors(
221           containerColor = MaterialTheme.colorScheme.primaryContainer
222         )
223       ) {
224         Column(
225           modifier = Modifier
226             .fillMaxWidth()
227             .padding( all = 20.dp ),
228           horizontalAlignment = Alignment.CenterHorizontally
229         ) {
230           Text(
231             text = "Your BMI Score",
232             style = MaterialTheme.typography.titleMedium,
233             color = MaterialTheme.colorScheme.onPrimaryContainer
234           )
235           Spacer( Modifier.height( 4.dp ) )
236           Text(
237             text = bmiValue.toString(),
238             style = MaterialTheme.typography.headlineLarge,
239             fontWeight = FontWeight.Black,
240             color = MaterialTheme.colorScheme.onPrimaryContainer
241           )
242         }
243       }
244     }

```

```

245
246 // Εμφάνιση Σφαλμάτων (Είτε από το ViewModel είτε από την τοπική εισαγωγή)
247 val displayError = inputError ?: uiState.errorMessage
248 if (displayError != null) {
249     Text(
250         text = "✖ $displayError",
251         color = MaterialTheme.colorScheme.error,
252         style = MaterialTheme.typography.bodyMedium,
253         fontWeight = FontWeight.Medium,
254         modifier = Modifier.padding(horizontal = 4.dp)
255     )
256 }
257 }
258
259 // Σταθερό Κάτω Μέρος: Κουμπιά Ενεργειών
260 Column(
261     modifier = Modifier
262         .fillMaxWidth()
263         .padding(vertical = 12.dp),
264     verticalArrangement = Arrangement.spacedBy(12.dp)
265 ) {
266     // Κουμπιά Υπολογισμού
267     Button(
268         onClick = {
269             onClick = {
270                 val weightValue = weight.toDoubleOrNull()
271                 if (weightValue != null && weightValue > 0 && weightValue < 300) {
272                     viewModel.bmiViewModel.insert( weightKg = weightValue, heightCm = height.toDouble(), user = authU
273                     viewModel.profileViewModel.clearSaveState()
274                     viewModel.profileViewModel.updateProfile(
275                         fullName = profileUiState.profile!!.fullName!!,
276                         height = null,
277                         weight = weightValue
278                     )
279                 } else {
280                     inputError = "Please enter a realistic weight between 1 and 300 kg."
281                 }
282             },
283             modifier = Modifier
284                 .fillMaxWidth()
285                 .height(50.dp),
286             shape = RoundedCornerShape( size = 12.dp)
287         ) {
288             Text("Calculate BMI", fontWeight = FontWeight.Bold)
289         }
290
291     // Κουμπιά Ιστορικού
292     ElevatedButton(
293         onClick = onHistoryClick,
294         modifier = Modifier

```

```

294         .fillMaxWidth()
295         .height(50.dp),
296         shape = RoundedCornerShape( size = 12.dp)
297     ) {
298         Text("📊 View BMI History", fontWeight = FontWeight.SemiBold)
299     }
300
301     // Κουμπί Επιστροφής
302     OutlinedButton(
303         onClick = {
304             onBack()
305             viewModel.bmiViewModel.clearSaveState()
306         },
307         modifier = Modifier
308             .fillMaxWidth()
309             .height(50.dp),
310         shape = RoundedCornerShape( size = 12.dp)
311     ) {
312         Text("← Back to Dashboard", fontWeight = FontWeight.Bold)
313     }
314 }
315
316
317 // Loading Overlay
318 if (uiState.isLoading) {
319     Box(
320         modifier = Modifier
321             .fillMaxSize()
322             .background( color = MaterialTheme.colorScheme.background.copy(alpha = 0.5f)),
323         contentAlignment = Alignment.Center
324     ) {
325         CircularProgressIndicator(color = MaterialTheme.colorScheme.primary)
326     }
327 }
328 }
329 }

```

Υπόδειγμα 27

Η συνάρτηση HomeScreen:

```

119     @OptIn( ...markerClass = ExperimentalMaterial3Api::class)
120     @Composable
121     fun HomeScreen(
122         onLogout: () -> Unit,
123         onProfileClick: () -> Unit,
124         onBmiClick: () -> Unit,
125         onExercisesClick: () -> Unit,
126         onWorkoutPlansClick: () -> Unit,
127         onPocketCoachClick: () -> Unit
128     ) {
129         Column(
130             modifier = Modifier
131                 .fillMaxSize()
132                 .padding(horizontal = 16.dp, vertical = 8.dp),
133             horizontalAlignment = Alignment.CenterHorizontally
134         ) {
135             // Καλωσόρισμα / Τίτλος Dashboard
136             Column(
137                 modifier = Modifier
138                     .fillMaxWidth()
139                     .padding(vertical = 16.dp)
140             ) {
141                 Text(
142                     text = "Welcome Back 🌞",
143                     style = MaterialTheme.typography.titleMedium,
144                     color = MaterialTheme.colorScheme.onSurfaceVariant
145                 )
146                 Text(
147                     text = "Fitness Dashboard",
148                     style = MaterialTheme.typography.headlineLarge,
149                     fontWeight = FontWeight.Bold,
150                     color = MaterialTheme.colorScheme.onBackground
151                 )
152             }
153
154             // Κεντρικό Περιεχόμενο με Scroll (Grid Διάταξη)
155             Column(
156                 modifier = Modifier
157                     .weight(1f)
158                     .verticalScroll( state = rememberScrollState()),
159                 verticalArrangement = Arrangement.spacedBy(16.dp)
160             ) {
161                 // 1. Highlight Card: PocketCoach AI (Full Width)
162                 Card(
163                     modifier = Modifier
164                         .fillMaxWidth()
165                         .clickable { onPocketCoachClick() },
166                     shape = RoundedCornerShape( size = 20.dp),
167                     colors = CardDefaults.cardColors(
168                         containerColor = MaterialTheme.colorScheme.primaryContainer
169                     )
170                 ) {
171                     Row(
172                         modifier = Modifier.padding( all = 20.dp),
173                         verticalAlignment = Alignment.CenterVertically
174                     ) {
175                         Text("🗨️", fontSize = 36.sp)

```

```
176         Spacer(Modifier.width(16.dp))
177         Column {
178             Text(
179                 text = "PocketCoach AI",
180                 style = MaterialTheme.typography.titleLarge,
181                 fontWeight = FontWeight.Bold,
182                 color = MaterialTheme.colorScheme.onPrimaryContainer
183             )
184             Text(
185                 text = "Ask your personal fitness assistant",
186                 style = MaterialTheme.typography.bodyMedium,
187                 color = MaterialTheme.colorScheme.onPrimaryContainer.copy(alpha = 0.8f)
188             )
189         }
190     }
191 }
192
193 // 2. Grid Row 1: Workout Plans & Exercises
194 Row(
195     modifier = Modifier.fillMaxWidth(),
196     horizontalArrangement = Arrangement.spacedBy(16.dp)
197 ) {
198     DashboardCard(
199         icon = "📅",
200         title = "Workout Plans",
201         subtitle = "Manage routines",
202         modifier = Modifier.weight(1f),
203         onClick = onWorkoutPlansClick
204     )
205     DashboardCard(
206         icon = "🏋️",
207         title = "Exercises",
208         subtitle = "Explore library",
209         modifier = Modifier.weight(1f),
210         onClick = onExercisesClick
211     )
212 }
213
214 // 3. Grid Row 2: BMI Calculator & Profile
215 Row(
216     modifier = Modifier.fillMaxWidth(),
217     horizontalArrangement = Arrangement.spacedBy(16.dp)
218 ) {
219     DashboardCard(
220         icon = "⚖️",
221         title = "BMI Calc",
222         subtitle = "Track your weight",
223         modifier = Modifier.weight(1f),
224         onClick = onBmiClick
225     )
226     DashboardCard(
227         icon = "👤",
228         title = "My Profile",
229         subtitle = "Update details",
```

```
230         modifier = Modifier.weight(1f),
231         onClick = onProfileClick
232     )
233 }
234 }
235
236 Spacer(Modifier.height(16.dp))
237
238 // Αναβαθμισμένο Κουμπί Αποσύνδεσης (Σταθερό στο κάτω μέρος)
239 OutlinedButton(
240     onClick = onLogout,
241     modifier = Modifier
242         .fillMaxWidth()
243         .height(50.dp),
244     shape = RoundedCornerShape(size = 12.dp),
245     colors = ButtonDefaults.outlinedButtonColors(
246         contentColor = MaterialTheme.colorScheme.error
247     ),
248     border = BorderStroke(
249         width = 1.dp,
250         color = MaterialTheme.colorScheme.error.copy(alpha = 0.5f)
251     )
252 ) {
253     Text(
254         text = "✖ Sign Out",
255         fontWeight = FontWeight.Bold,
256         style = MaterialTheme.typography.bodyLarge
257     )
258 }
259 }
260 }
261
262 // Reusable Component για τα μικρότερα καρτελάκια του Grid
263 4 Usages
264 @Composable
265 fun DashboardCard(
266     icon: String,
267     title: String,
268     subtitle: String,
269     modifier: Modifier = Modifier,
270     onClick: () -> Unit
271 ) {
272     Card(
273         modifier = modifier
274             .height(140.dp)
275             .clickable { onClick() },
276         shape = RoundedCornerShape(size = 16.dp),
277         colors = CardDefaults.cardColors(
278             containerColor = MaterialTheme.colorScheme.surfaceVariant.copy(alpha = 0.4f)
279         )
280     ) {
281         Column(
282             modifier = Modifier
283                 .fillMaxSize()
284                 .padding(all = 16.dp),
```

```
284         verticalArrangement = Arrangement.Center,  
285         horizontalAlignment = Alignment.Start  
286     ) {  
287         Text(icon, fontSize = 28.sp)  
288         Spacer(Modifier.height(12.dp))  
289         Text(  
290             text = title,  
291             style = MaterialTheme.typography.titleMedium,  
292             fontWeight = FontWeight.Bold,  
293             maxLines = 1  
294         )  
295         Spacer(Modifier.height(2.dp))  
296         Text(  
297             text = subtitle,  
298             style = MaterialTheme.typography.bodySmall,  
299             color = MaterialTheme.colorScheme.onSurfaceVariant.copy(alpha = 0.7f),  
300             maxLines = 1  
301         )  
302     }  
303 }  
304 }
```

Υπόδειγμα 28

Τα object Routes:

```
1 package com.ngk.fitnessapp.presentation
2
3 > import ...
4
5
6 @Serializable
7 object Auth {
8     @Ignore
9     fun serializer(): K<Auth> {
10         Generated by a Kotlin compiler plugin
11     }
12 }
13
14 @Serializable
15 object Home {
16     @Ignore
17     fun serializer(): KSerializer<Home> {
18     }
19 }
20
21 @Serializable
22 object Profile {
23     @Ignore
24     fun serializer(): KSerializer<Profile> {
25     }
26 }
27
28 @Serializable
29 object Bmi {
30     @Ignore
31     fun serializer(): KSerializer<Bmi> {
32     }
33 }
34
35 3 Usages
36 @Serializable
37 object BmiHistory {
38     @Ignore
39     fun serializer(): KSerializer<BmiHistory> {
40     }
41 }
42
43 3 Usages
44 @Serializable
45 object Exercises {
46     @Ignore
47     fun serializer(): KSerializer<Exercises> {
48     }
49 }
50
51 3 Usages
52 @Serializable
53 object InsertExercises {
54     @Ignore
55     fun serializer(): KSerializer<InsertExercises> {
56     }
57 }
```

```
34  @Serializable
35  class ExerciseDetail(val id: Long){
    @Ignore
    companion object {
        @Ignore
        fun serializer(): KSerializer<ExerciseDetail>
    }
36  }
37
38  3 Usages
39  @Serializable
40  object WorkoutPlans{
    @Ignore
    fun serializer(): KSerializer<WorkoutPlans>
41  }
42  4 Usages
43  @Serializable
44  class WorkoutPlanDetail(val planId: String){
    @Ignore
    companion object {
        @Ignore
        fun serializer(): KSerializer<WorkoutPlanDetail>
    }
45  }
46  @Serializable
47  object InsertWorkoutPlan{
    @Ignore
    fun serializer(): KSerializer<InsertWorkoutPlan>
48  }
49
50  3 Usages
51  @Serializable
52  object Chat{
    @Ignore
    fun serializer(): KSerializer<Chat>
53  }
```

Υπόδειγμα 29

Η συνάρτηση FitnessAppBar:

```
26 @OptIn( ...markerClass = ExperimentalMaterial3Api::class)
27 @Composable
28 fun FitnessAppBar(
29     currentScreen: NavDestination?,
30     canNavigateBack: Boolean,
31     navigateUp: () -> Unit
32 ){
33     if (currentScreen != null) {
34         CenterAlignedTopAppBar(
35             title = {
36                 val title = when {
37
38                     currentScreen.hasRoute<Profile>() -> stringResource(Res.string.Profile)
39                     currentScreen.hasRoute<Bmi>() -> stringResource(Res.string.bmi)
40                     currentScreen.hasRoute<BmiHistory>() -> stringResource(Res.string.bmi_history)
41                     currentScreen.hasRoute<Exercises>() -> stringResource(Res.string.exercises)
42                     currentScreen.hasRoute<ExerciseDetail>() -> stringResource(Res.string.exercise_detail)
43                     currentScreen.hasRoute<InsertExercises>() -> stringResource(Res.string.insert_exercise)
44                     currentScreen.hasRoute<WorkoutPlans>() -> stringResource(Res.string.workout_plans)
45                     currentScreen.hasRoute<WorkoutPlanDetail>() -> stringResource(Res.string.workout_plan_exercises)
46                     currentScreen.hasRoute<Chat>() -> stringResource(Res.string.pocket_coach)
47                     else -> stringResource(Res.string.Homescreen)
48                 }
49                 Text(text = title)
50             },
51             navigationIcon = {
52                 if (canNavigateBack) {
53                     IconButton(onClick = navigateUp) {
54                         Icon(
55                             painter = painterResource(Res.drawable.arrow_back_24px),
56                             contentDescription = "Return"
57                         )
58                     }
59                 }
60             }
61         )
62     }
63 }
```

Υπόδειγμα 30

Η συνάρτηση NavApp:

```
1 package com.ngk.fitnessapp.presentation
2
3 > import ...
4
5 2 Usages
6
7 @OptIn(markerClass = ExperimentalMaterial3Api::class)
8 @Composable
9 fun NavApp(
10     generalViewModel: GeneralViewModel
11 ) {
12     val navController = rememberNavController()
13     val backStackEntry by navController.currentBackStackEntryAsState()
14     val currentScreen = backStackEntry?.destination
15
16     Scaffold(
17         topBar = {
18             FitnessAppBar(
19                 currentScreen = currentScreen,
20                 canNavigateBack = navController.previousBackStackEntry != null && !currentScreen!!.hasRoute<Auth>(),
21                 navigateUp = { navController.navigateUp() }
22             )
23         }
24     ) { innerPadding ->
25
26         NavHost(
27             navController = navController,
28             startDestination = Auth,
29             modifier = Modifier
30                 .fillMaxSize()
31                 .padding(paddingValues = innerPadding)
32         ) {
33             composable<Auth> {
34                 AuthScreen(
35                     viewModel = generalViewModel,
36                     onSignInSuccess = {
37                         navController.navigate(route = Home) {
38                             popUpTo(route = Auth) { inclusive = true }
39                         }
40                     },
41                     onSignUpSuccess = {
42                         navController.navigate(route = Profile) {
43                             popUpTo(route = Auth) { inclusive = true }
44                         }
45                     }
46                 )
47             }
48
49             composable<Home> {
50                 HomeScreen(
51                     onLogout = {
52                         generalViewModel.authViewModel.signOut()
53                         navController.navigate(route = Auth)
54                     },
55                     onProfileClick = { navController.navigate(route = Profile) },
56                     onBmiClick = { navController.navigate(route = Bmi) },
57                     onExercisesClick = { navController.navigate(route = Exercises) },
58                     onWorkoutPlansClick = { navController.navigate(route = WorkoutPlans) },
59                     onPocketCoachClick = { navController.navigate(route = Chat) }
60                 )
61             }
62         }
63     }
64 }
```

```

84         )
85     }
86
87     composable<Profile> {
88         ProfileScreen(
89             viewModel = generalViewModel,
90             onSignUpSuccess = {
91                 navController.navigate( route = Home) {
92                     popUpTo( route = Home) { inclusive = true }
93                 }
94             },
95             onBack = { navController.popBackStack() }
96         )
97     }
98
99     // BMI
100    composable<Bmi> {
101        BmiScreen(
102            viewModel = generalViewModel,
103            onHistoryClick = { navController.navigate( route = BmiHistory) },
104            onBack = { navController.popBackStack() }
105        )
106    }
107
108    composable<BmiHistory> {
109        BmiHistoryScreen(
110            viewModel = generalViewModel.bmiViewModel,
111            onBack = { navController.popBackStack() }
112        )
113    }
114
115    // Exercises
116    composable<Exercises> {
117        ExerciseListScreen(
118            viewModel = generalViewModel,
119            onExerciseClick = { exercise ->
120                navController.navigate( route = ExerciseDetail(exercise.id))
121            },
122            onInsertExerciseClick = { navController.navigate( route = InsertExercises) },
123            onBack = { navController.popBackStack() }
124        )
125    }
126
127    composable<ExerciseDetail> { backStackEntry ->
128        val exerciseDetail: ExerciseDetail = backStackEntry.toRoute()
129
130        ExerciseDetailScreen(
131            viewModel = generalViewModel.exerciseViewModel,
132            exercise = generalViewModel.exerciseViewModel
133                .uiState
134                .value
135                .exercises
136                .find { it.id == exerciseDetail.id },
137            onBack = { navController.popBackStack() }

```

```
138     )
139 }
140
141 composable<InsertExercises> {
142     ExerciseInsertScreen(
143         viewModel = generalViewModel.exerciseViewModel,
144         onBack = { navController.popBackStack() }
145     )
146 }
147
148 // WorkoutPlans
149 composable<WorkoutPlans> {
150     WorkoutPlanListScreen(
151         viewModel = generalViewModel,
152         onPlanClick = { planId ->
153             navController.navigate( route = WorkoutPlanDetail(planId))
154         },
155         onBack = { navController.popBackStack() }
156     )
157 }
158
159 composable<WorkoutPlanDetail> { backStackEntry ->
160     val workoutPlanDetail: WorkoutPlanDetail = backStackEntry.toRoute()
161
162     WorkoutPlanDetailScreen(
163         planId = workoutPlanDetail.planId,
164         viewModel = generalViewModel,
165         onBack = { navController.popBackStack() }
166     )
167 }
168
169 composable<Chat> {
170     ChatScreen(
171         viewModel = generalViewModel.chatViewModel
172     )
173 }
174 }
175 }
176 }
```

Υπόδειγμα 31

Η συνάρτηση App:

```

53     @Composable
54     fun App(
55         db: FitnessDatabase
56     ) {
57         val coroutineScope = rememberCoroutineScope()
58
59         val client = remember { SupabaseClientProvider.client }
60         val supabaseAuthService = remember { SupabaseAuthService(client) }
61
62         val syncManager = remember { SyncManager() }
63
64         val fitnessSupportBot = remember { FitnessSupportBot() }
65
66         val generalViewModel = remember {
67             GeneralViewModel(
68                 authViewModel = AuthViewModel(
69                     coroutineScope,
70                     supabaseAuthService,
71                     syncManager,
72                     db
73                 ),
74                 profileViewModel = ProfileViewModel(
75                     repository = RealProfileRepository( database = db ),
76                     coroutineScope
77                 ),
78                 bmiViewModel = BmiViewModel(
79                     bmiRepository = RealBmiRepository( database = db ),
80                     calculateBMIUseCase = CalculateBMIUseCase(),
81                     coroutineScope
82                 ),
83                 exerciseViewModel = ExerciseViewModel(
84                     repository = RealExerciseRepository( database = db ),
85                     coroutineScope
86                 ),
87                 workoutViewModel = WorkoutViewModel(
88                     repository = RealWorkoutRepository( database = db ),
89                     coroutineScope
90                 ),
91                 chatViewModel = ChatViewModel(
92                     coroutineScope,
93                     fitnessSupportBot
94                 )
95             )
96         }
97
98         MaterialTheme {
99             var showContent by remember { mutableStateOf( value = false ) }
100             var navigateToNavApp by remember { mutableStateOf( value = false ) }
101
102             if (showContent && !navigateToNavApp) {
103                 LaunchedEffect( key1 = Unit ) {
104                     delay( timeMillis = 2000 ) // Αναμονή 2 δευτερολέπτων
105                     navigateToNavApp = true // Ενεργοποίηση πλοήγησης
106                 }
107             }
108
109             Box(modifier = Modifier.fillMaxSize()) {
110

```

```

111 NavApp(
112     generalViewModel
113 )
114
115 AnimatedVisibility(
116     visible = !navigateToNavApp,
117     exit = fadeOut(animationSpec = tween( durationMillis = 500)) // 0μαλό σβήσιμο όταν τελει
118 ) {
119     Column(
120         modifier = Modifier
121             .background( color = MaterialTheme.colorScheme.primaryContainer)
122             .safeContentPadding()
123             .fillMaxSize(),
124         horizontalAlignment = Alignment.CenterHorizontally,
125         verticalArrangement = Arrangement.Center
126     ) {
127         if (!showContent) {
128             Image(painterResource(
129                 Res
130                 .drawable
131                 .workout
132             ),
133                 contentDescription = null
134             )
135             Spacer(modifier = Modifier.height(16.dp))
136             Button(onClick = { showContent = true }) {
137                 Text("Click to Start")
138             }
139         }
140
141         AnimatedVisibility( visible = showContent) {
142             val greeting = remember { Greeting().greet() }
143             Column(
144                 modifier = Modifier.fillMaxWidth(),
145                 horizontalAlignment = Alignment.CenterHorizontally,
146             ) {
147                 Image(painterResource(
148                     Res
149                     .drawable
150                     .compose_multiplatform),
151                     contentDescription = null
152                 )
153                 Text("Compose: $greeting")
154             }
155         }
156     }
157 }
158
159 }
160

```

Υπεύθυνη Δήλωση Συγγραφέα:

Δηλώνω ρητά ότι, σύμφωνα με το άρθρο 8 του Ν.1599/1986, η παρούσα εργασία αποτελεί αποκλειστικά προϊόν προσωπικής μου εργασίας, δεν προσβάλλει κάθε μορφής δικαιώματα διανοητικής ιδιοκτησίας, προσωπικότητας και προσωπικών δεδομένων τρίτων, δεν περιέχει έργα/εισφορές τρίτων για τα οποία απαιτείται άδεια των δημιουργών/δικαιούχων και δεν είναι προϊόν μερικής ή ολικής αντιγραφής, οι πηγές δε που χρησιμοποιήθηκαν περιορίζονται στις βιβλιογραφικές αναφορές και μόνον και πληρούν τους κανόνες της επιστημονικής παράθεσης.