ΟΙΚΟΝΟΜΙΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΔΙΟΙΚΗΣΗΣ ΕΠΙΧΕΙΡΗΣΕΩΝ

ΤΜΗΜΑ ΔΙΟΙΚΗΤΙΚΗΣ ΕΠΙΣΤΗΜΗΣ ΚΑΙ ΤΕΧΝΟΛΟΓΙΑΣ

Εργασία στο μάθημα: «Ανάπτυξη & Σχεδίαση Κινητών Εφαρμογών»

7ο εξάμηνο

«Gym App: Εφαρμογή Διαχείρισης Προπονήσεων»

ΦΟΙΤΗΤΕΣ

ΔΟΥΚΑΣ ΔΗΜΗΤΡΙΟΣ – t8210038

ΚΑΡΥΩΤΑΚΗΣ ΙΩΑΝΝΗΣ – t8210055

ΜΑΡΚΟΓΙΑΝΝΗΣ ΓΕΩΡΓΙΟΣ - t8210089

ΠΡΟΕΣΤΑΚΗΣ ΦΟΙΒΟΣ ΤΙΜΟΘΕΟΣ – t8210126

ΕΠΙΒΛΕΠΩΝ: ΜΑΡΙΟΣ ΦΡΑΓΚΟΥΛΗΣ

Αθήνα, Ιανουάριος 2025

Περιεχόμενα

Κεφάλαιο 1: Εισαγωγή

1.1 Περιγραφή της Εφαρμογής

Η εφαρμογή, στοχεύει στην εύκολη διαχείριση προπονήσεων γυμναστηρίου μέσω αναλυτικής καταγραφής ασκήσεων. Επιτρέπει τον υπολογισμό μετρικών όπως θερμίδες, χρονική διάρκεια και είδος ασκήσεων, ενώ παρακολουθεί την πρόοδο του χρήστη με γραφήματα. Επιπλέον, ενσωματώνει αθλητικές ειδήσεις για να κρατά τους χρήστες ενημερωμένους και κινητοποιημένους. Γενικότερα, ως σκοπός της εφαρμογής είναι η βελτίωση της υγείας και της φυσικής κατάστασης των χρηστών, μέσω της παρακολούθησης της προόδου τους.

Το αποθετήριο της εργασίας, το οποίο είναι διαθέσιμο εδώ, περιλαμβάνει περισσότερα από 120 commits, με πάνω από 20 pull requests να έχουν ενσωματωθεί στο κύριο κλαδί (main branch). Ο συνολικός κώδικας αποτελείται από περισσότερες από 7.500 σειρές κώδικα.

1.2 Ομάδα Ανάπτυξης

Η ομάδα ανάπτυξης της εφαρμογής Android σε Kotlin αποτελείται από τέσσερα μέλη:

Φοίβος Προεστακης ανέλαβε τον ρόλο του συντονιστή ομάδας και του δημιουργού της κύριας λειτουργίας της εφαρμογής. Εξασφάλισε ότι η βασική λογική της εφαρμογής λειτουργεί με τον ορθό τρόπο. Επιπλέον, ανέλαβε την βελτίωση διάφορων λειτουργιών της εφαρμογής, καθώς και την επίβλεψη της συνολικής ανάπτυξης του έργου.

Δημήτρης Δούκας ήταν υπεύθυνος για την ανάπτυξη του UI/UX. Επιπλέον, υλοποίησε τις οθόνες του προφίλ και της διαχείριση του, καθώς και των ρυθμίσεων της εφαρμογής, επιτρέποντας στους χρήστες να αλλάζουν τα χρώματα ή να διαφοροποιούν προσωπικές πληροφορίες όπως email, password, βάρος, ύψος & ηλικία.

Γιάννης Καρυωτάκης ανέλαβε την ευθύνη της ανάπτυξης των λειτουργιών backend και της διαχείρισης δεδομένων. Υλοποίησε τη βάση δεδομένων που υποστηρίζουν την εφαρμογή. Επιπλέον, διασφάλισε την ασφάλεια και την απόδοση των δεδομένων, καθώς και την ενσωμάτωση με εξωτερικές υπηρεσίες μέσω API, όπως το Google News, ενισχύοντας τις δυνατότητες της εφαρμογής.

Γιώργος Μαρκογιάννης υλοποίησε την προβολή και διαχείριση του ιστορικό προπονήσεων, διασφαλίζοντας ότι η εφαρμογή ενημερώνει τον χρήστη για την πορεία του.

Κεφάλαιο 2: Ανάλυση Απαιτήσεων

Στον παρόν κεφάλαιο αναλύονται οι απαιτήσεις της εφαρμογής, τόσο λειτουργικές όσο και μη λειτουργικές, προκειμένου να εξασφαλιστεί μια πλήρης αποτύπωση των αναγκών που καλύπτει η εφαρμογή για την επίτευξη των στόχων της.

2.1 Λειτουργικές Απαιτήσεις

Οι λειτουργικές απαιτήσεις καθορίζουν τις βασικές λειτουργίες που πρέπει να παρέχει η εφαρμογή στους χρήστες της. Καταρχάς, η εφαρμογή υποστηρίζει την διαχείριση χρηστών, επιτρέποντας στους νέους χρήστες να δημιουργούν λογαριασμούς και στους υπάρχοντες να συνδέονται μέσω email και κωδικού πρόσβασης. Επιπλέον, η δυνατότητα αλλαγής κωδικού πρόσβασης είναι ουσιώδης για την εξασφάλιση της πρόσβασης των χρηστών στα προσωπικά τους δεδομένα.

Μία από τις κύριες λειτουργίες της εφαρμογής είναι η καταγραφή προπονήσεων, η οποία επιτρέπει στους χρήστες να εισάγουν λεπτομερώς τις ασκήσεις τους, συμπεριλαμβανομένου του αριθμού επαναλήψεων και του βάρους. Η εφαρμογή επίσης παρέχει τη δυνατότητα επιλογής ασκήσεων από μια βάση ασκήσεων. Η εφαρμογή παρέχει ιστορικό γράφημα και ιστορικές αναφορές στις προπονήσεις που έχουν καταγραφεί, προκειμένου να παρακολουθούν την πρόοδό τους

Τέλος, η ενσωμάτωση αθλητικών ειδήσεων ενημερώνει τους χρήστες για τις τελευταίες τάσεις, ασκήσεις και συμβουλές από ειδικούς στον χώρο της γυμναστικής, διατηρώντας τους ενημερωμένους.

2.2 Μη Λειτουργικές Απαιτήσεις

Οι μη λειτουργικές απαιτήσεις καθορίζουν τις γενικές ιδιότητες και τις επιδόσεις της εφαρμογής που δεν σχετίζονται άμεσα με συγκεκριμένες λειτουργίες, αλλά είναι κρίσιμες για την απόδοση, την εμπειρία του χρήστη και τη συνολική ποιότητα της εφαρμογής. Για την παρούσα εφαρμογή, οι μη λειτουργικές απαιτήσεις διαμορφώθηκαν ως εξής:

Η εφαρμογή προσφέρει σχετικά υψηλή απόδοση, διασφαλίζοντας γρήγορη φόρτωση των οθονών σε λιγότερο από 300ms (πέραν της αρχικής) και αποτελεσματικό χειρισμό δεδομένων χωρίς καθυστερήσεις. Η ασφάλεια αποτέλεσε προτεραιότητα, με όλα τα προσωπικά και ευαίσθητα δεδομένα των χρηστών να κρυπτογραφούνται τόσο κατά την αποθήκευση όσο και κατά τη μετάδοση. Οι διαδικασίες εγγραφής και εισόδου προστατεύθηκαν από επιθέσεις όπως SQL Injection, εφαρμόζοντας τις καλές πρακτικές του οικοσυστήματος της Google. Οι κωδικοί πρόσβασης αποθηκεύτηκαν με ασφαλή κρυπτογράφηση και όχι σε απλό κείμενο.

Η χρηστικότητα της εφαρμογής ήταν υψηλή, προσφέροντας μια φιλική και ευανάγνωστη διεπαφή χρήστη που επέτρεπε στους χρήστες να πραγματοποιούν

εύκολα τις επιθυμητές ενέργειες. Η εφαρμογή σχεδιάστηκε ώστε να είναι προσβάσιμη σε χρήστες διαφόρων ηλικιών και επιπέδων τεχνολογικής εξοικείωσης, παρέχοντας απλά γραφικά στοιχεία. Επιπλέον η εφαρμογή σχεδιάστηκε ώστε να είναι ανθεκτική σε σφάλματα, διαχειριζόμενη τα σφάλματα χωρίς να διακόπτει τη λειτουργία της και παρέχοντας κατάλληλα μηνύματα σφάλματος και δυνατότητα επαναφοράς σε ασφαλή κατάσταση.

Η συμβατότητα ήταν κρίσιμη, με την εφαρμογή να λειτουργεί απρόσκοπτα σε διάφορες συσκευές Android με διαφορετικές αναλύσεις οθόνης και μεγέθη. Για να διασφαλίσουμε την καλύτερη δυνατή εμπειρία χρήστη, σχεδιάσαμε τα κεντρικά κουμπιά δράσης (action buttons) ώστε να παραμένουν πάντα ορατά ανεξαρτήτως της διάστασης της οθόνης, ενώ το υπόλοιπο περιεχόμενο οργανώθηκε σε scrolable διατάξεις.

Κεφάλαιο 3: Αρχιτεκτονικός Σχεδιασμός

Το παρόν κεφάλαιο εστιάζει στον αρχιτεκτονικό σχεδιασμό της εφαρμογής, περιγράφοντας αναλυτικά τη συνολική δομή και τα κύρια στοιχεία που τη συνθέτουν. Όπου χρειάζεται, παρατίθενται στιγμιότυπα του κώδικα που αποτυπώνουν τις αρχιτεκτονικές επιλογές.

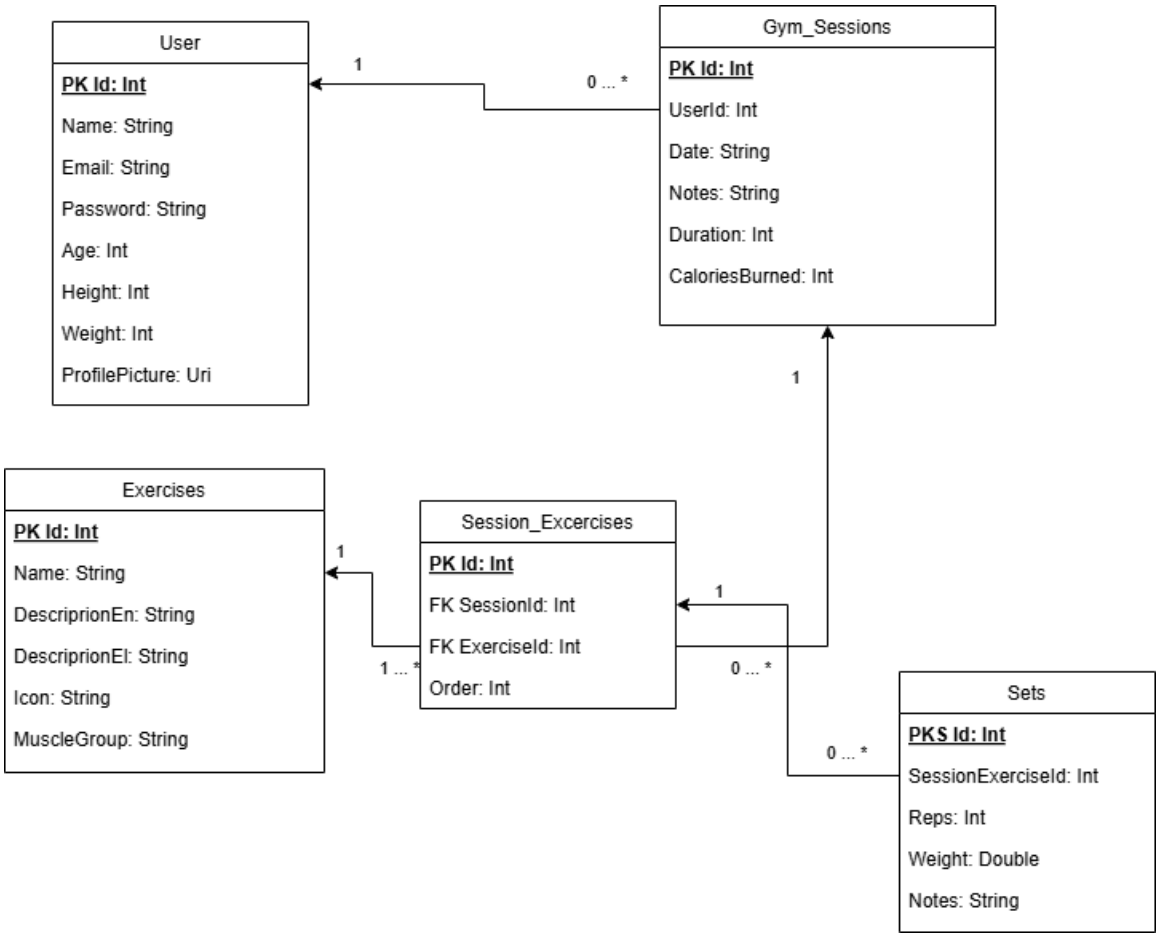
3.1 Γενική Περιγραφή της Αρχιτεκτονικής

Η αρχιτεκτονική της εφαρμογής σχεδιάστηκε με βάση το μοτίβο MVVM (Model-View-ViewModel), το οποίο εξασφαλίζει σαφή διαχωρισμό των ευθυνών, βελτιώνοντας τη συντηρησιμότητα και διευκολύνοντας την επεκτασιμότητα του κώδικα. Το **Model** διαχειρίζεται τα δεδομένα χρησιμοποιώντας τη Room Database για τοπική αποθήκευση και το Retrofit για επικοινωνία με εξωτερικά APIs. Το **View** υλοποιείται με το Jetpack Compose, παρέχοντας μια δυναμική γραφική διεπαφή που προσαρμόζεται σε διάφορα μεγέθη και αναλύσεις των οθονών. Το **ViewModel** λειτουργεί ως ενδιάμεσος μεταξύ του Model και του View, χρησιμοποιώντας LiveData και Coroutines για την παρακολούθηση και αποστολή δεδομένων σε πραγματικό χρόνο στο UI.

Για την πλοήγηση μεταξύ των οθονών, χρησιμοποιείται το Jetpack Navigation Component, εξασφαλίζοντας ομαλές μεταβάσεις και αποτελεσματική διαχείριση του ιστορικού πλοήγησης. Το Material Design εφαρμόζεται για μια συνεπή και ελκυστική αισθητική διεπαφής, ενώ το Android Studio παρέχει ένα ολοκληρωμένο περιβάλλον ανάπτυξης με όλα τα απαραίτητα εργαλεία για συγγραφή, δοκιμή και διόρθωση του κώδικα.

3.2 Διάγραμμα Βάσης Δεδομένων (Database Diagram)

Στο παρακάτω διάγραμμα απεικονίζεται η δομή και οι οντότητες της εφαρμογής. Συγκεκριμένα παρουσιάζονται τα γνωρίσματα, τα πρωτεύοντα και δευτερεύοντα κλειδιά και οι συσχετίσεις που έχει κάθε οντότητας.



Η εφαρμογή εν των πλείστων βασίζεται σε 5 διαφορετικές οντότητες, όπου κάθε μια από αυτή έχει τα δικά τις χαρακτηριστικά που την προσδιορίζουν. Αναφορικά μόνο οι οντότητες αυτές είναι:

- User
- Gym_Session
- Exercises
- Session_Exercises
- Sets

3.3 Διαχείριση Δεδομένων

3.3.1 Βάση Δεδομένων

Για την διαχείριση της βάσης δεδομένων χρησιμοποιήθηκε η βιβλιοθήκη Room ως στρώμα αφαίρεσης πάνω από την SQLite. Σύμφωνα με τις απαιτήσεις τις βιβλιοθήκης δημιουργήθηκαν οι οντότητες της βάσης με τις κατάλληλες επισημειώσεις και τους περιορισμούς κλειδιών. Ενδεικτικά, παρατίθενται κάποιες οντότητες.

```kotlin
@Entity(tableName = "users")
data class User(
    @PrimaryKey(autoGenerate = true) val id: Int = 0,
    val name: String,
    val email: String,
    val passwordHash: String,
    val age: Int?,
    val height: Int?,
    val weight: Int?,
    val profilePicture: Uri? = null
)
```

```kotlin
@Entity(
    tableName = "session_exercises",
    foreignKeys = [
        ForeignKey(
            entity = GymSession::class,
            parentColumns = ["id"],
            childColumns = ["sessionId"],
            onDelete = ForeignKey.CASCADE
        ),
        ForeignKey(
            entity = Exercise::class,
            parentColumns = ["id"],
            childColumns = ["exerciseId"],
            onDelete = ForeignKey.CASCADE
        )
    ]
)
data class SessionExercise(
    @PrimaryKey(autoGenerate = true) val id: Int = 0,
    val sessionId: Int,
    val exerciseId: Int,
    val order: Int
)
```

3.3.2 Αρχιτεκτονική διαχείρισης Δεδομένων

Η εφαρμογή ακολουθεί τις καλές πρακτικές στη διαχείριση των δεδομένων, βασισμένες σε μια **αρθρωτή (modular)** και **καθαρά διαχωρισμένη** αρχιτεκτονική, η οποία διασφαλίζει την ευκολία στη συντήρηση και την επεκτασιμότητα. Η ροή των δεδομένων οργανώνεται μέσα από μια ακολουθία επιπέδων: **Entity -> DAO -> Repository -> AppContainer -> MyApplication**, προσφέροντας έναν ξεκάθαρο διαχωρισμό των λειτουργιών και των ευθυνών.

3.3.2.1 Dependency Injection και Κεντρική Διαχείριση Πόρων

Η χρήση του **AppContainer** αποτελεί βασικό στοιχείο της αρχιτεκτονικής της εφαρμογής. Ο **AppContainer** λειτουργεί ως κεντρικός διαχειριστής εξαρτήσεων (Dependency Injection Container), εξασφαλίζοντας ότι οι εξαρτήσεις, όπως τα Repositories, τα Retrofit APIs και η Room Database, δημιουργούνται και διαχειρίζονται σε ένα ενιαίο σημείο. Αυτό αποτρέπει τον πολλαπλασιασμό αντικειμένων και την περιττή χρήση πόρων, ενώ παρέχει τις εξαρτήσεις σε κάθε μέρος της εφαρμογής μέσω του **MyApplication**, δημιουργώντας έναν κοινό χώρο κατάστασης (Global State Container) που εξυπηρετεί ολόκληρη την εφαρμογή.

3.3.2.2 Διαχωρισμός Λειτουργιών και Καθαρότητα Κώδικα

Η εφαρμογή ακολουθεί τις αρχές του **καθαρού κώδικα (clean code)** μέσω του σαφούς διαχωρισμού των επιπέδων:

1. **Entities**: Τα αντικείμενα των δεδομένων (π.χ., `User`, `SessionExercise`) ορίζουν τη δομή των δεδομένων που αποθηκεύονται και μεταφέρονται.
2. **DAO (Data Access Objects)**: Τα DAO αναλαμβάνουν την άμεση επικοινωνία με τη βάση δεδομένων, παρέχοντας λειτουργίες όπως εισαγωγή, ανάκτηση, ενημέρωση και διαγραφή.
3. **Repositories**: Τα Repositories λειτουργούν ως **ενδιάμεσο επίπεδο** μεταξύ των δεδομένων (DAO ή API) και του υπόλοιπου συστήματος. Αυτή η προσέγγιση παρέχει μια **αφαίρεση (abstraction)** που διευκολύνει την εναλλαγή πηγών δεδομένων (π.χ., από τοπική βάση σε απομακρυσμένο API).
4. **AppContainer**: Συγκεντρώνει όλες τις εξαρτήσεις σε ένα σημείο, διασφαλίζοντας ότι η δημιουργία των αντικειμένων είναι διαχειρίσιμη και αποδοτική.
5. **MyApplication**: Η εφαρμογή ορίζεται σε παγκόσμιο επίπεδο, ώστε να παρέχει πρόσβαση σε όλα τα στοιχεία του AppContainer μέσω μιας σαφώς ορισμένης δομής.

3.3.2.3 Αρθρωτότητα και Επεκτασιμότητα

Η αρθρωτή δομή επιτρέπει τη μεμονωμένη αναβάθμιση ή αντικατάσταση κάθε επιπέδου. Για παράδειγμα, η μετάβαση από το Room σε ένα διαφορετικό persistence framework ή η προσθήκη νέας λειτουργικότητας στο API γίνεται χωρίς να επηρεάζονται άλλα επίπεδα. Επιπλέον, η ενσωμάτωση νέων οθονών ή χαρακτηριστικών γίνεται γρήγορα, χάρη στη σαφή οργάνωση και τις υπάρχουσες αφαιρέσεις.

Συνολικά, η εφαρμογή αξιοποιεί σύγχρονες πρακτικές όπως το **Dependency Injection**, την **αρχή του διαχωρισμού ευθυνών (Separation of Concerns)** και τη

χρήση σαφών αφαιρέσεων, καθιστώντας την καλά δομημένη, επεκτάσιμη και εύκολα συντηρήσιμη.

```
class UserRepository(private val userDao: UserDao, private val context: Context) {

    private val sharedPreferences: SharedPreferences by lazy {
        context.getSharedPreferences("user_preferences", Context.MODE_PRIVATE)
    }

    Giannis Karyotakis
    suspend fun getUserByEmail(email: String): User? {
        return userDao.getUserByEmail(email)
    }

    Giannis Karyotakis
    suspend fun registerUser(user: User): Boolean {
        return if (userDao.doesUserExist(user.email)) {
            false // Email already exists
        } else {
            userDao.registerUser(user)
            saveLoggedInUserEmail(user.email)
            true
        }
    }
```

Figure 1 Υλοποίηση αποθετηρίου χρήστη ως επίπεδο αφαίρεσης πάνω απο το DAO

```kotlin
interface AppContainer {
    val userRepository: UserRepository
    val workoutRepository: WorkoutRepository
    val retrofitService: NewsApiService
    val themePreferences: ThemePreferences
}

Giannis Karyotakis +1
class AppContainerImpl(private val context: Context) : AppContainer {

    private  val BASE_URL = "https://newsapi.org/v2/"
    private val json = Json { this: JsonBuilder
        ignoreUnknownKeys = true
    }
    private val retrofit: Retrofit = Retrofit.Builder() Retrofit.Builder
        .baseUrl(BASE_URL) Retrofit.Builder
        .addConverterFactory(json.asConverterFactory(
            contentType = "application/json".toMediaType())
        ).build()
    override val retrofitService : NewsApiService by lazy {
        retrofit.create(NewsApiService::class.java) }

    override val userRepository: UserRepository by lazy {
        UserRepository.getInstance(
        AppDatabase.getInstance(context).userDao(),
        context
        )
```

Figure 2: Υλοποίηση App Container για το μοτίβο Dependency Injection

```kotlin
class MyApplication : Application() {
    lateinit var appContainer: AppContainer
    Giannis Karyotakis
    override fun onCreate() {
        super.onCreate()
        // Initialize AppContainer after Application is fully initialized
        appContainer = AppContainerImpl( context: this)
    }
}
```

Figure 3: Υλοποίηση κλάσης MyApplication για καθολική πρόσβαση στο container

3.3.3 Ανάκτηση και Ενημέρωση Δεδομένων από Διαδίκτυο

Η ανάκτηση δεδομένων από το διαδίκτυο υλοποιήθηκε με την βιβλιοθήκη Retrofit για την εύκολη σύνδεση με REST APIs . Εκμεταλλευτήκαμε το news api για την εύρεση ειδήσεων, στοχεύοντας στον χώρο του αθλητισμού. Για της ενσωμάτωση του news api ακολουθήθηκαν τα παρακάτω βήματα:

Αρχικά, δημιουργήθηκαν οι οντότητες που περιγράφουν κατάλληλα της απάντηση Json.

```kotlin
import kotlinx.serialization.Serializable

Giannis Karyotakis
@Serializable
data class NewsResponse(
    val status: String?,
    val totalResults: Int?,
    val articles: List<Article>
)

Giannis Karyotakis
@Serializable
data class Article(
    val source: Source?,
    val author: String? = null,
    val title: String,
    val description: String? = null,
    val url: String? = null,
    val urlToImage: String? = null,
    val publishedAt: String,
    val content: String? = null
)

Giannis Karyotakis
@Serializable
data class Source(
    val id: String? = null,
    val name: String?
)
```

Στην συνέχεια, διαμορφώθηκε η διεπαφή με την υλοποίηση της συνάρτησης.

```kotlin
interface NewsApiService {
    Giannis Karyotakis
    @GET("everything")
    suspend fun getNews(
        @Query("q") query: String,
        @Query("from") fromDate: String,
        @Query("sortBy") sortBy: String,
        @Query("apiKey") apiKey: String,
        @Query("pageSize") pageSize: Int,
        @Query("page") page: Int,
        @Query("language") language: String
    ): NewsResponse
}
```

Η αρχικοποίηση των απαραίτητων εξαρτήσεων και της υπηρεσίας γίνεται στο κιβώτιο της εφαρμογής που έχει ως σκοπό την “ένεση εξαρτήσεων” (dependency injection). Μέσω του κιβωτίου εξαρτήσεων της εφαρμογής (AppContainer), η υπηρεσία NewsApiService αρχικοποιείται μία φορά και επαναχρησιμοποιείται σε όλη την εφαρμογή, βελτιώνοντας την απόδοση και τη συντηρησιμότητα.

```kotlin
class AppContainerImpl(private val context: Context) : AppContainer {

    private  val BASE_URL = "https://newsapi.org/v2/"
    private val json = Json { this: JsonBuilder
        ignoreUnknownKeys = true
    }
    private val retrofit: Retrofit = Retrofit.Builder() Retrofit.Builder
        .baseUrl(BASE_URL) Retrofit.Builder
        .addConverterFactory(json.asConverterFactory(
            contentType = "application/json".toMediaType())
        ).build()
    override val retrofitService : NewsApiService by lazy {
        retrofit.create(NewsApiService::class.java) }
```

Στην κλάση NewsViewModel.kt χρησιμοποιούμε τις κατάλληλες παραμέτρους για να κάνουμε σωστά την αίτηση στο endpoint.

```
    @RequiresApi(Build.VERSION_CODES.O)
    private fun getNews() {
        viewModelScope.launch { this: CoroutineScope
            newsUiState = try {
                val response: NewsResponse = service.getNews(
                    query = "Sports",
                    fromDate = getDates(),
                    sortBy = "publishedAt",
                    apiKey = "e77cba1ddfbf40618da7d163d4d33946",
                    pageSize = 10,
                    page = 1,
                    language = "en"
                )
                NewsUiState.Success(response.articles)
            } catch (e: Exception) {
                    NewsUiState.Error(e.message ?: "An unexpected error occurred!")
            }
        }
    }
}
```

Τα δεδομένα που επιστρέφονται από την υπηρεσία NewsApiService εκτίθενται μέσω ενός mutableState στο ViewModel, το οποίο παρακολουθείται από το Composable News.kt, διασφαλίζοντας ότι η διεπαφή χρήστη ενημερώνεται δυναμικά. Τέλος, από το ViewModel τα δεδομένα καταλήγουν στο View News.kt και στα μάτια του χρήστη.

3.4 Διαχείριση Κατάστασης (State Management)

3.4.1 Διαχείρηση Κατάστασης στην σύνδεση View - ViewModel

Για την διαχείριση κατάστασης χρησιμοποιήθηκαν καλές πρακτικές στις οθόνες (Views) και στις κλάσεις διαχείρισης οθονών (View Models). Ενδεικτικά, λοιπόν, παρουσιάζονται παρακάτω μερικά παραδείγματα.

Όπως φαίνεται στο παράδειγμα του viewModel της αρχικής οθόνης, ορίζονται οι ιδιωτικές μεταβλητές που παρέχουν την κατάστασης στις οθόνες ως backing properties, ώστε να προστατεύονται τα δεδομένα της εφαρμογής από αλλαγές από εξωτερικές κλάσεις αλλά να επιτρέπεται τις εξωτερικές κλήσεις να έχουν πρόσβαση στην τιμή.

π.χ. το _userId περιέχει την κατάσταση, ενώ το userId παρέχει την δημόσια διεπαφή για το _userId

```
class HomeViewModel(
    private val workoutRepository: WorkoutRepository,
    private val userRepository: UserRepository
) : ViewModel() {

    private val _userId = MutableStateFlow( value: 0)
    val userId: StateFlow<Int?> = _userId.asStateFlow()

    private val _isWorkoutActive = MutableStateFlow( value: false)
    val isWorkoutActive: StateFlow<Boolean> = _isWorkoutActive.asStateFlow()

    private val _currentSessionId = MutableStateFlow<Int?>( value: null)
    val currentSessionId: StateFlow<Int?> = _currentSessionId.asStateFlow()

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

    private val _userSessions = MutableStateFlow<List<GymSession>>(emptyList())
    val userSessions: StateFlow<List<GymSession>> = _userSessions.asStateFlow()
```

Αντίστοιχα, στην οθόνη η κατάσταση ανατίθεται με την χρήση της μεθόδου collectAsState() χρησιμοποιώντας κάθε φορά την δημόσια διεπαφή της κατάστασης

```
@Composable
fun HomeScreen(
    navController: NavController,
    viewModel: HomeViewModel,
    currentStatusViewModel: CurrentStatusViewModel
) {
    val isWorkoutActive by viewModel.isWorkoutActive.collectAsState()
    val currentSessionId by viewModel.currentSessionId.collectAsState()
    val username by viewModel.username.collectAsState()
    val userId by viewModel.userId.collectAsState()
    val sessionList by viewModel.userSessions.collectAsState()
```

3.4.2 Διαχείρηση Κατάστασης στην εισαγωγή δεδομένων από χρήστη

Συνεχίζοντας, σε οθόνες όπου οι καταχωρήσεις του χρήστη έπρεπε να κάνουν επανασύνθεση στην εφαρμογή, χρησιμοποιήθηκαν οι συναρτήσεις remember και mutableStateOf. Το `mutableStateOf` δημιουργεί ένα αντικείμενο που είναι παρατηρήσιμο (observable) από το Compose. Αυτό σημαίνει ότι όταν αλλάζει η τιμή του, το Compose ανιχνεύει την αλλαγή και κάνει recomposition των composables που εξαρτώνται από αυτό το state. Η συνάρτηση remember χρησιμοποιείται, ώστε όταν ο κώδικας ενός composable εκτελείται ξανά (recomposes), οι τιμές δεν επανακαθορίζονται από το αρχικό state, αλλά παραμένουν όπως είχαν οριστεί.

```
@Composable
fun RegisterScreen(
    registerViewModel: RegisterViewModel,
    homeViewModel: HomeViewModel,
    myProfileViewModel: MyProfileViewModel,
    navController: NavController
) {
    // States for email, password, confirm password, and visibility
    var email by remember { mutableStateOf( value: "") }
    var password by remember { mutableStateOf( value: "") }
    var username by remember { mutableStateOf( value: "") }
    var confirmPassword by remember { mutableStateOf( value: "") }
    var isPasswordVisible by remember { mutableStateOf( value: false) }
    var isConfirmPasswordVisible by remember { mutableStateOf( value: false) }
    var showError by remember { mutableStateOf( value: false) }
    var showSuccessfulRegister by remember { mutableStateOf( value: false) }
    var showErrorRegister by remember { mutableStateOf( value: false) }

    val errorMessage by registerViewModel.errorMessage.collectAsState()
```

3.4.3 Ανύψωση Κατάστασης (State Hoisting)

Η ανύψωση κατάστασης (State Hoisting) αποτελεί μια βέλτιστη πρακτική στην διαχείριση της κατάστασης της εφαρμογής, η οποία αφορά την τη διαδικασία μεταφοράς της διαχείρισης μιας κατάστασης (state) σε ένα ανώτερο επίπεδο (π.χ. σε έναν γονικό composable), ώστε τα παιδιά components να λαμβάνουν την κατάσταση και τις αντίστοιχες λειτουργίες ενημέρωσης ως παραμέτρους αντί να διαχειρίζονται το δικό τους state. Στο παρακάτω παράδειγμα, στην οθόνη CurrentStatus η κατάσταση exercisesWithSets ορίζεται στο γονικό composable.

```
@Composable
fun CurrentStatus(
    sessionId: Int,
    viewModel: CurrentStatusViewModel,
    navController: NavController,
    onWorkoutTerminated: (Int, Int) -> Unit
) {
    val timer by viewModel.timerState.collectAsState()
    val calories by viewModel.caloriesState.collectAsState()
    val exercisesWithSets by viewModel.currentExercises.collectAsState()
    var setToEdit by remember { mutableStateOf<Set?>( value: null) }
    var setToAdd by remember { mutableStateOf<Int?>( value: null) }
    val error by viewModel.errorState.collectAsState()
```

Στην συνέχεια, η κατάσταση και οι συναρτήσεις διαχείρισης του, δίνονται στο παρακάτω στην ιεραρχία composable ως παράμετροι:

```
items(exercisesWithSets) { this: LazyItemScope exerciseWithSets ->
    ExerciseWithSetsCard(
        exerciseWithSets = exerciseWithSets,
        onRemoveSet = { setId ->
            viewModel.removeSetFromExercise(setId)
        },
        onDeleteExercise = { exerciseId ->
            viewModel.deleteExercise(
                exerciseId,
                language = context.resources.
                configuration.locales[0].language)
        },
        onEditSet = { set ->
            setToEdit = set
        },
        onAddSet = { sessionExerciseId ->
            setToAdd = sessionExerciseId
        }
    )
}
```

Με αυτό τον τρόπο, επιτυγχάνεται καλή άρθρωση στον κώδικα (modularity), διότι είναι ξεκάθαρη η λειτουργικότητά του.

Κεφάλαιο 4: Υλοποίηση

Η υλοποίηση της εφαρμογής πραγματοποιήθηκε με στόχο την παροχή ενός ολοκληρωμένου εργαλείου για τη διαχείριση προπονήσεων στο γυμναστήριο και την παρακολούθηση της φυσικής κατάστασης των χρηστών. Η εφαρμογή περιλάμβανε διάφορες λειτουργίες που αναλύονται παρακάτω.

4.1 Περιγραφή Λειτουργιών

Η εφαρμογή διέθετε δυνατότητες διαχείρισης χρηστών, επιτρέποντας στους νέους χρήστες να εγγράφονται και στους υπάρχοντες να συνδέονται με ασφαλή τρόπο μέσω email και κωδικού πρόσβασης. Οι χρήστες μπορούν να καταγράφουν λεπτομερώς τις προπονήσεις τους, συμπεριλαμβάνοντας πληροφορίες όπως αριθμός επαναλήψεων, βάρος, χρονική διάρκεια και είδος ασκήσεων.

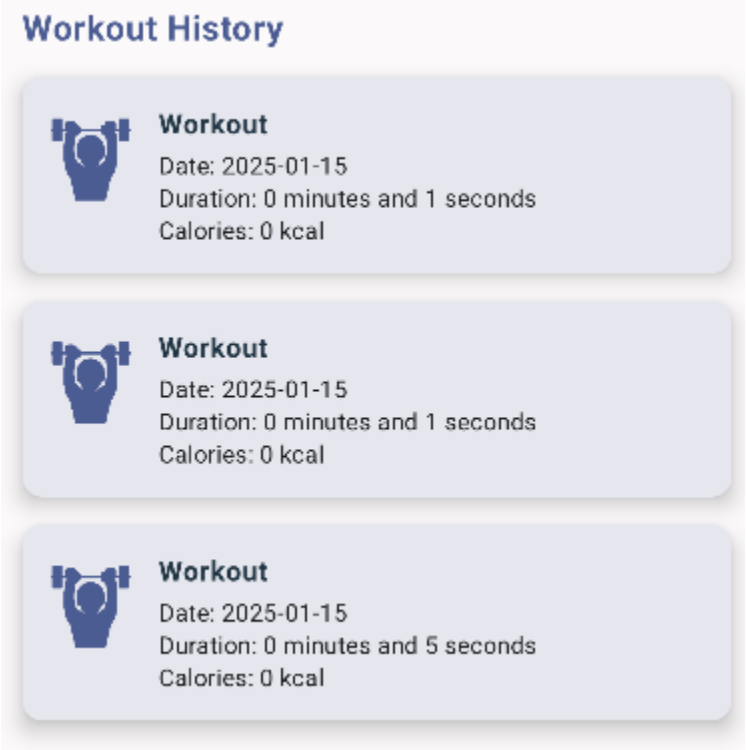
Η παρακολούθηση της προόδου των χρηστών επιτυγχάνεται μέσω γραφημάτων προόδου και αναφορών που απεικονίζουν τις επιδόσεις των προπονήσεων, όπως οι θερμίδες που καίγονται. Επιπλέον ο χρήστης πέρα από την καταγραφή και παρακολούθηση των προηγούμενων προπονήσεων, στην εφαρμογή έχει ενσωματωθεί μια σελίδα με τα νέα στον τομέα του αθλητισμού, προκειμένου να μένει και ενήμερος σε αυτόν τον τομέα.

4.2 Χρήση Κύριων Γραφικών Στοιχείων

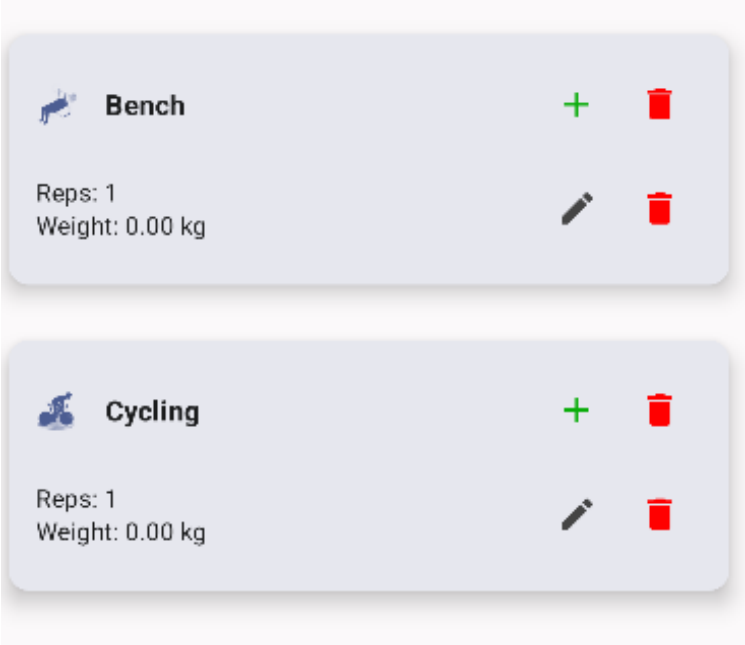
Η εφαρμογή αξιοποίησε βασικά γραφικά στοιχεία για τη δημιουργία μιας δυναμικής και φιλικής προς τον χρήστη διεπαφή. Κάθε στοιχείο ενσωματώθηκε με τρόπο που εξυπηρετεί τις ανάγκες του χρήστη και την αποδοτική λειτουργία της εφαρμογής.

4.2.1 Λίστες

Λίστες χρησιμοποιήθηκαν για την εμφάνιση δεδομένων που σχετίζονται με τις ασκήσεις. Συγκεκριμένα, η εφαρμογή εμφανίζει μια λίστα ασκήσεων που έχουν προστεθεί στο πρόγραμμα προπόνησης, περιλαμβάνοντας πληροφορίες όπως το όνομα της άσκησης, ο αριθμός των επαναλήψεων και το βάρος. Επιπλέον, οι χρήστες έχουν τη δυνατότητα να δουν τις πρόσφατες προπονήσεις τους μέσω λίστας προηγούμενων προπονήσεων, οργανωμένης σε χρονολογική σειρά, με συνοπτικά στατιστικά στοιχεία για κάθε προπόνηση. Για παράδειγμα, μια οθόνη εμφάνιζε μια προπόνηση ως "Workout - 10/01/2025: 45 λεπτά, 300 θερμίδες".



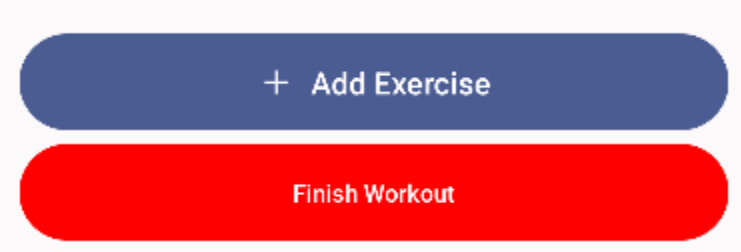
Αντίστοιχα στην οθόνη της ενεργής προπόνησης φαίνονται οι ασκήσεις που έχει προσθέσει ο χρήστης.



4.2.2 Κουμπιά

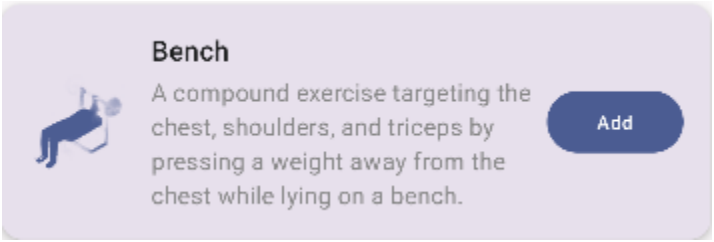
Τα κουμπιά αποτελούν ένα από τα πιο σημαντικά γραφικά στοιχεία, διευκολύνοντας την πλοήγηση και την αλληλεπίδραση των χρηστών με την εφαρμογή. Ένα

παράδειγμα χρήσης κουμπιών στην εφαρμογή είναι το εξής: το κουμπί "Add Exercise" οδηγεί στη φόρμα καταχώρισης νέας άσκησης, ενώ το κουμπί "Finish Workout" επιτρέπει στους χρήστες να τερματίζουν την συγκεκριμένη προπόνηση. Παρακάτω αποτυπώνονται τα δύο κουμπιά.



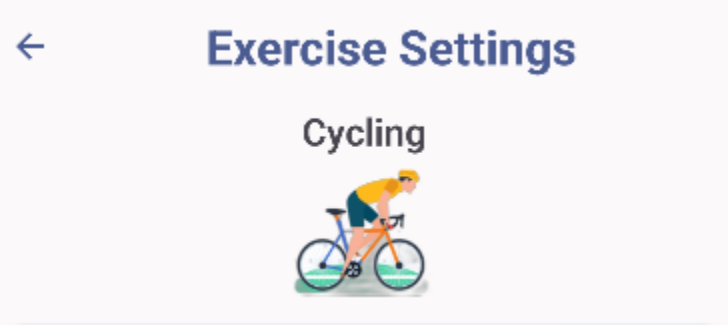
4.2.3 Κείμενο

Το κείμενο χρησιμοποιείται ευρέως στην εφαρμογή για την εμφάνιση πληροφοριών, την περιγραφή επιλογών και την καθοδήγηση των χρηστών. Ένα παράδειγμα χρήσης κειμένου στην εφαρμογή είναι η περιγραφή της κάθε άσκησης στην οθόνη επιλογής ασκήσεων.



4.2.4 Εικόνες

Οι εικόνες συμβάλουν στην αισθητική της εφαρμογής και ενισχύουν την παρουσίαση των δεδομένων με οπτικό τρόπο. Εικονίδια ασκήσεων συνοδεύουν κάθε άσκηση. Για παράδειγμα, στην οθόνη που προσθέτει τα σετ μιας συγκεκριμένης άσκησης στην κορυφή αυτής αποτυπώνεται εικόνα της συγκεκριμένης άσκησης που έχει επιλεγεί.

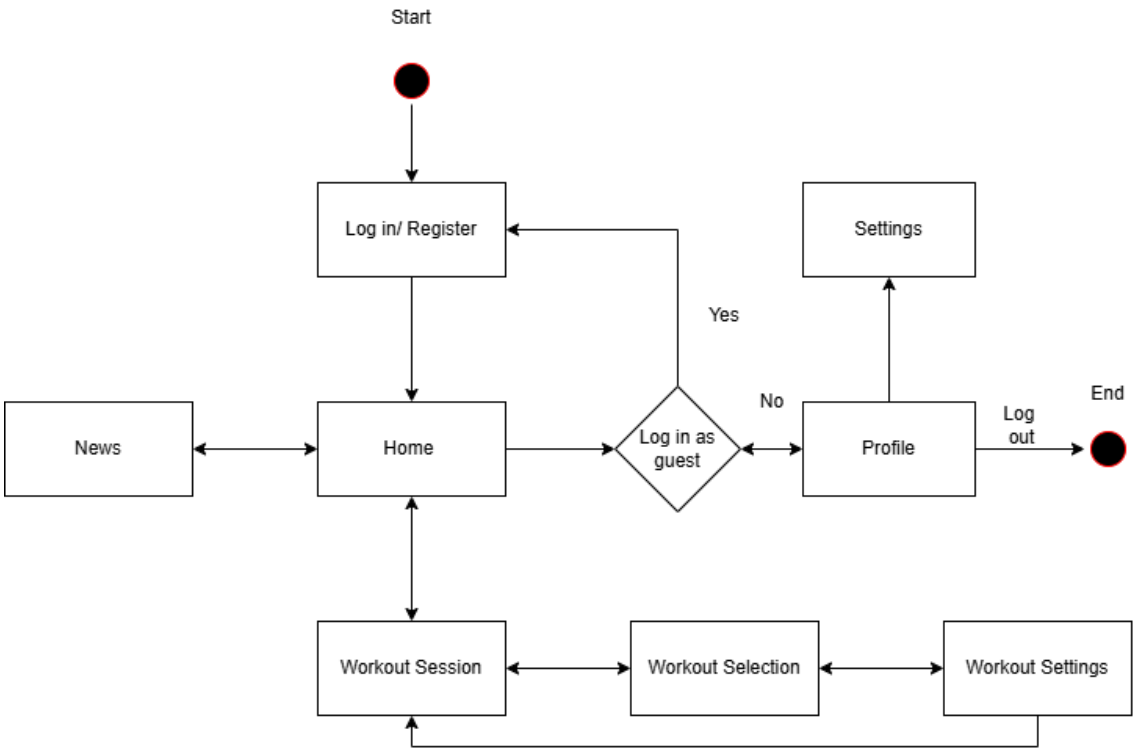


4.2.5 Βίντεο

Η ενσωμάτωση βίντεο δεν υλοποιήθηκε στην εφαρμογή. Εξαιτίας περιορισμών χρόνου και της ανάγκης να δοθεί προτεραιότητα σε πιο κρίσιμες λειτουργίες, αποφασίστηκε να μην ενσωματωθούν βίντεο στην τρέχουσα έκδοση.

4.3 Περιήγηση στην εφαρμογή

Στην παρούσα ενότητα περιγράφεται αρχικά η αναμενόμενη ροή κατά τη χρήση της εφαρμογής. Συγκεκριμένα, το παρακάτω διάγραμμα αναπαριστά την δραστηριότητα ενός χρήστη μέσα στην εφαρμογή μας. Προκειμένου να γίνει ακόμη πιο κατανοητό το συγκεκριμένο διάγραμμα, στη συνέχεια θα γίνει μια σύντομη περιγραφή των διαφορετικών δραστηριοτήτων του χρήστη στην εφαρμογή μας. Ο χρήστης προκειμένου να μεταβεί στην αρχική οθόνη, αρχικά θα τεθεί να δημιουργήσει έναν λογαριασμό (μέσω του register) είτε να συνδεθεί με τον υπάρχον με έναν ήδη υπάρχων, ή ακόμη να συνεχίσει την πλοήγησή του ως guest. Έχοντας συνδεθεί πλέον, ο χρήστης μπορεί στην αρχική οθόνη, είτε να ξεκινήσει μια νέα προπόνηση είτε να δει τις προηγούμενες προπονήσεις διαγραμματικά στο εύρος μιας εβδομάδας ή μέσω μιας λίστας. Ξεκινώντας μια νέα προπόνηση, (Workout Session) μπορεί να προσθέσει ποια άσκηση σκοπεύει να κάνει, μέσω φίλτρων ή μέσω της μπάρας αναζήτησης (Workout Selection), και στη συνέχεια να ορίσει τις επαναλήψεις και τα κιλά της άσκησης (WorkoutSettings). Ολοκληρώνοντας την προπόνηση του, θα μεταβεί στην αρχική οθόνη, στην οποία η προπόνηση που μόλις κατέγραψε θα έχει προστεθεί στο ιστορικό. Εναλλακτικές δραστηριότητες που μπορεί ο χρήστης να ακολουθήσει είναι να μεταβεί είτε στην οθόνη με τα νέα, σχετικά με τον αθλητισμό είτε στο προφίλ του. Στην περίπτωση που βρίσκεται στο προφίλ του, ο χρήστης έχει την επιλογή να κάνει αποσύνδεση από το λογαριασμό του ή να μεταβεί σε περαιτέρω ρυθμίσεις του προφίλ του, προκειμένου να διαφοροποιήσει κάποια στοιχεία του λογαριασμού του και να διαμορφώσει το UI της εφαρμογής με βάση την προτίμησή του. Στην περίπτωση όπου ο χρήστης έχει συνδεθεί ως guest όταν επιλέγει να μεταβεί στην οθόνη του προφίλ, θα καταλήξει στο login/ register. Η μετάβαση μεταξύ της αρχικής οθόνης του προφίλ και των νέων γίνεται μέσα του navbar. Στη συνέχεια θα ακολουθήσει μια πιο αναλυτική παρουσίαση και σχολιασμός για κάθε μία από τις οθόνες της εφαρμογής.

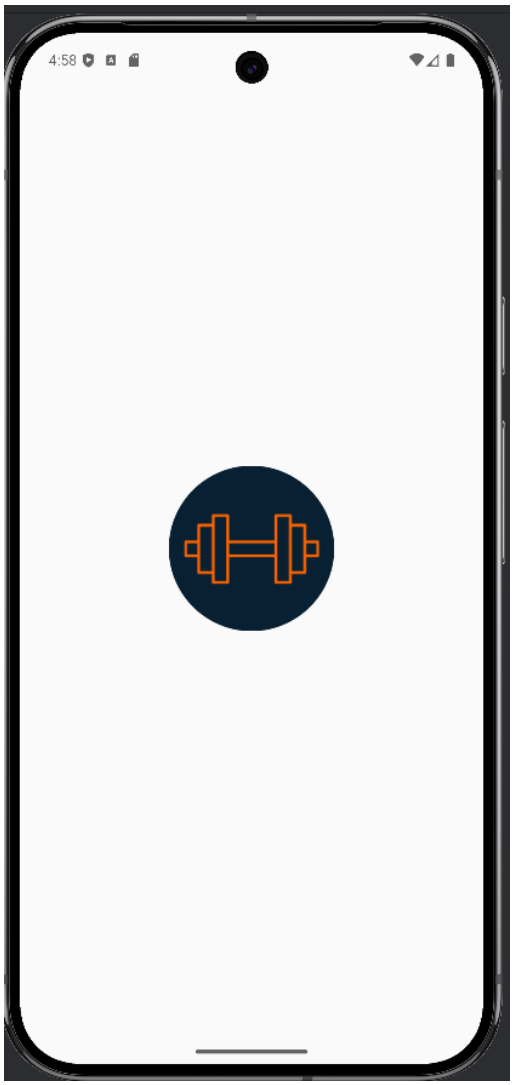


4.4 Περιγραφή Οθονών

Η συγκεκριμένη ενότητα εστιάζει στην παρουσίαση των οθονών και στην κατανόηση των βασικών λειτουργιών και χαρακτηριστικών των οθονών της εφαρμογής.

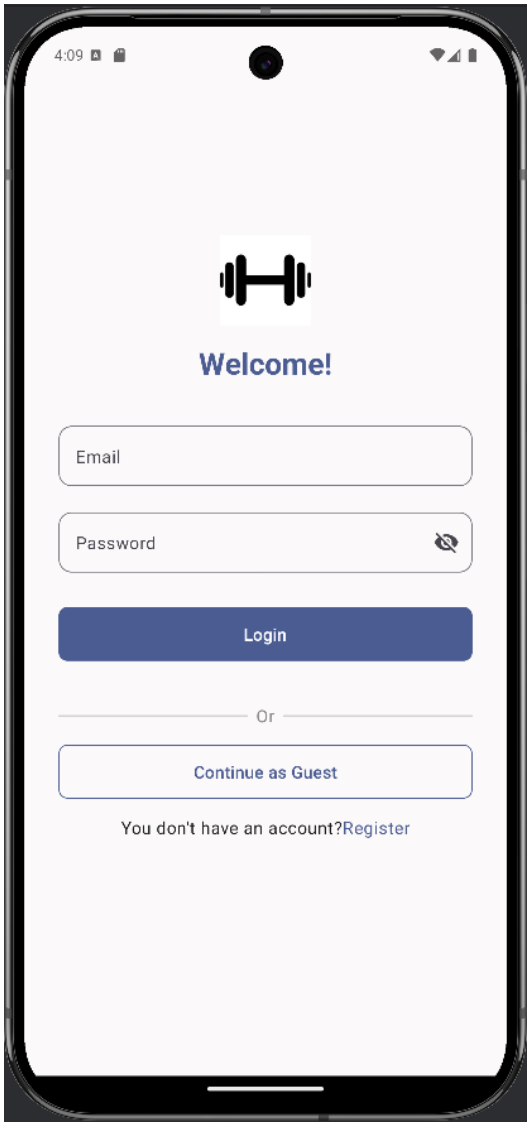
4.4.1 Οθόνης έναρξης (Splash Screen)

Όταν η εφαρμογή ανοίγει, εμφανίζεται για μερικά δευτερόλεπτα η οθόνη έναρξης (Splash Screen), όπως φαίνεται στην εικόνα, στην οποία προβάλλεται το λογότυπο της εφαρμογής. Μετά από ένα μικρό χρονικό διάστημα θα προβληθεί η οθόνη του login.



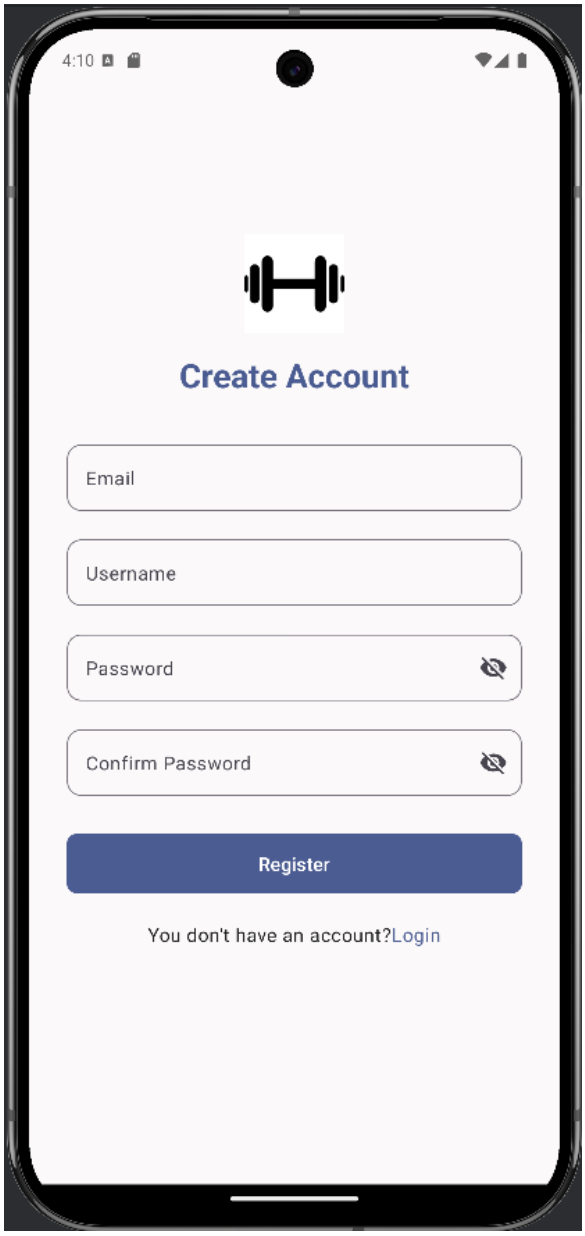
4.4.2 Οθόνη Login

Η πρώτη οθόνη της εφαρμογής είναι η αρχική οθόνη εισόδου (login screen) της εφαρμογής. Στο κεντρικό τμήμα της οθόνης, ο χρήστης καλείται να εισαγάγει τα στοιχεία του (email και κωδικό πρόσβασης) προκειμένου να αποκτήσει πρόσβαση στις λειτουργίες της εφαρμογής. Εάν εισάγει λανθασμένα τα στοιχεία του, η οθόνη θα του εμφανίσει ένα κατάλληλο μήνυμα και δε θα του επιτρέψει να συνεχίσει μέχρις ότου εισάγει σωστά τα στοιχεία του. Εάν δεν επιθυμεί να συνδεθεί, μπορεί να επιλέξει την επιλογή “Continue as Guest”. Στην περίπτωση που ο χρήστης δεν έχει ήδη κάποιον λογαριασμό, μπορεί να προχωρήσει σε εγγραφή μέσω του συνδέσμου “Register”.



4.4.3 Οθόνη Register

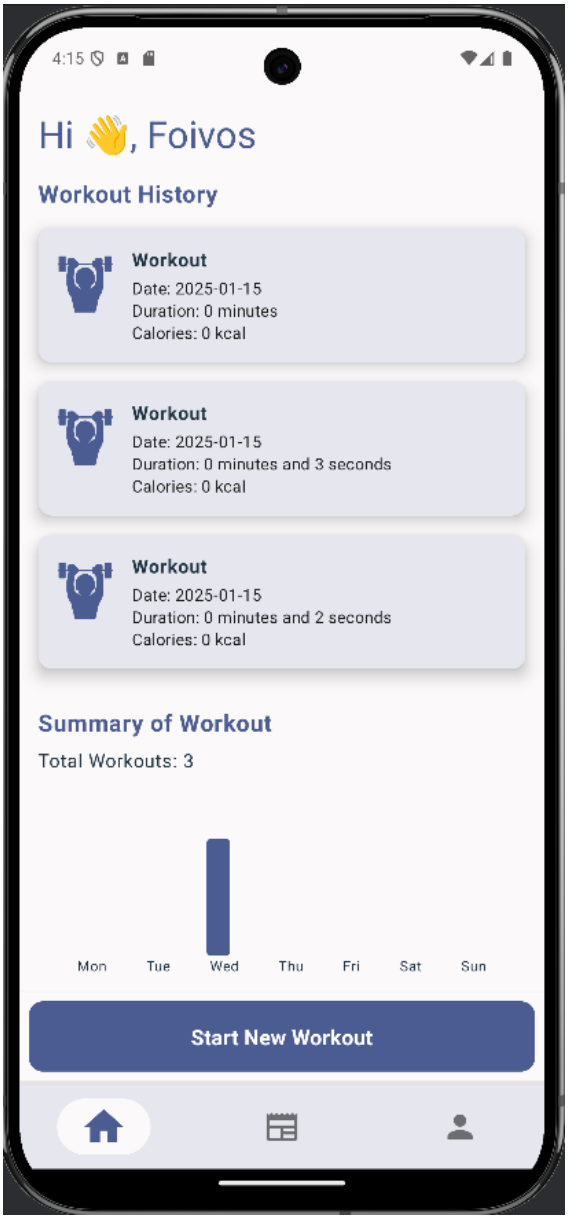
Η δεύτερη οθόνη της εφαρμογής είναι η οθόνη δημιουργίας λογαριασμού (register screen). Εδώ, ο χρήστης καλείται να συμπληρώσει τα προσωπικά του στοιχεία, όπως τη διεύθυνση email, το επιθυμητό όνομα χρήστη και τον κωδικό πρόσβασης, ενώ υπάρχει και πεδίο επιβεβαίωσης του κωδικού για πρόσθετη ασφάλεια. Το κουμπί “Register” επιτρέπει την καταχώρηση του νέου λογαριασμού, ενώ ο σύνδεσμος “Login” οδηγεί πίσω στην οθόνη εισόδου για όσους διαθέτουν ήδη λογαριασμό. Προκειμένου να δημιουργήσει ο χρήστης νέο λογαριασμό χρειάζεται να εισάγει ένα ρεαλιστικό email, το όνομα χρήστη να είναι μεγαλύτερο από 3 χαρακτήρες και έναν κωδικό, ο οποίος θα είναι τουλάχιστον 8 χαρακτήρες, περιλαμβάνοντας τουλάχιστον ένα χαρακτήρα, τουλάχιστον ένα νούμερο. Σε περίπτωση που δεν ισχύει κάποια από τις αναφερόμενες συνθήκες, η εγγραφή δε μπορεί να ολοκληρωθεί και η εφαρμογή θα εμφανίσει ένα μήνυμα με κόκκινους χαρακτήρες με το λάθος του χρήστη.



4.4.4 Οθόνη Home

Η τρίτη οθόνη της εφαρμογής είναι η κεντρική οθόνη το οποίο αποτελεί το home. Εδώ, ο χρήστης υποδέχεται ένα φιλικό μήνυμα καλωσορίσματος με το όνομά του, ενώ ακριβώς από κάτω εμφανίζονται οι πρόσφατες προπονήσεις. Για καθεμία αναγράφονται η ημερομηνία , η διάρκεια και οι θερμίδες που κάηκαν . Επιπλέον, υπάρχει μια σύνοψη των συνολικών προπονήσεων με ένα γράφημα που απεικονίζει τη δραστηριότητα κάθε ημέρας της εβδομάδας. Μέσω του κουμπιού “Start New Workout”, ο χρήστης μπορεί εύκολα να ξεκινήσει μια καινούρια προπόνηση. Στο κάτω μέρος της οθόνης υπάρχει και ένα navbar προκειμένου να βοηθήσει τον χρήστη

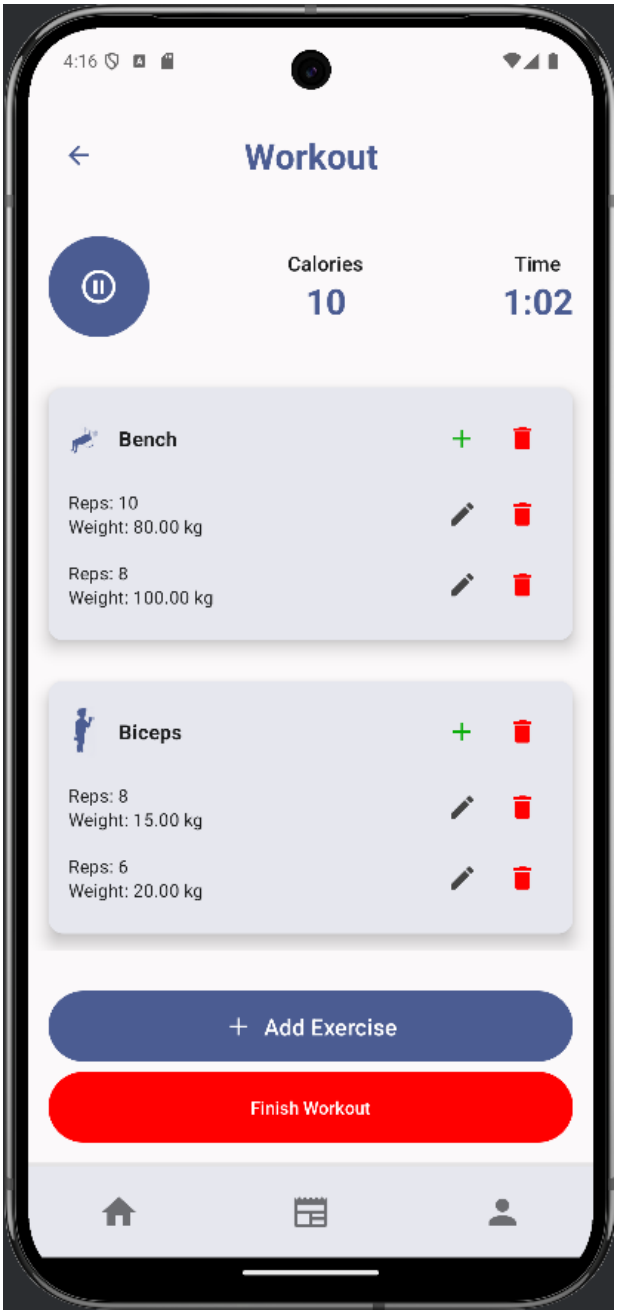
στην πλοήγησή του σε άλλες λειτουργίες της εφαρμογής αλλά και να γνωρίζει σε ποιο σημείο με την ένδειξη του σκιαγραφημένου εικονιδίου.



4.4.5 Οθόνη CurrentStatus

Η τέταρτη οθόνη της εφαρμογής είναι η οθόνη προπόνησης. Στο επάνω μέρος εμφανίζονται συνοπτικά οι θερμίδες και ο χρόνος προπόνησης. Πάνω αριστερά υπάρχει ένα κουμπί με το οποίο ο χρήστης μπορεί να σταματήσει και να συνεχίσει τον χρόνο της προπόνησης. Κάθε προπόνηση χωρίζεται σε ενότητες ασκήσεων, όπου ο χρήστης βλέπει τις λεπτομέρειες των επαναλήψεων (Reps) και του βάρους (Weight), τις οποίες τις έχει καταγράψει προηγουμένως στην οθόνη “SetRepsScreen”. Παρέχονται κουμπιά για την επεξεργασία ή τη διαγραφή κάθε άσκησης, καθώς και

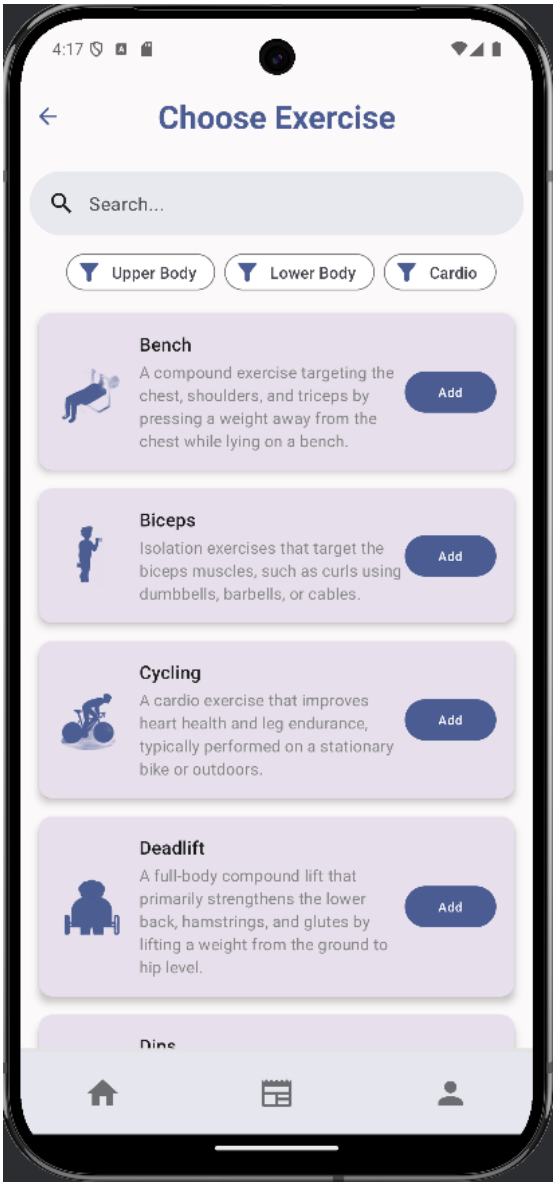
ένα πλήκτρο “+ Add Exercise” για την προσθήκη νέων ασκήσεων. Ολοκληρώνοντας τη προπόνησή του, ο χρήστης μπορεί να επιλέξει “Finish Workout” για να τερματίσει την προπόνηση. Με το που ολοκληρωθεί η προπόνηση, ο χρήστης πλοηγείται πίσω στην αρχική , με την τελευταία προπόνηση να έχει ενταχθεί στο ιστορικό των προπονήσεών του.



4.4.6 Οθόνη ExercisePickerScreen

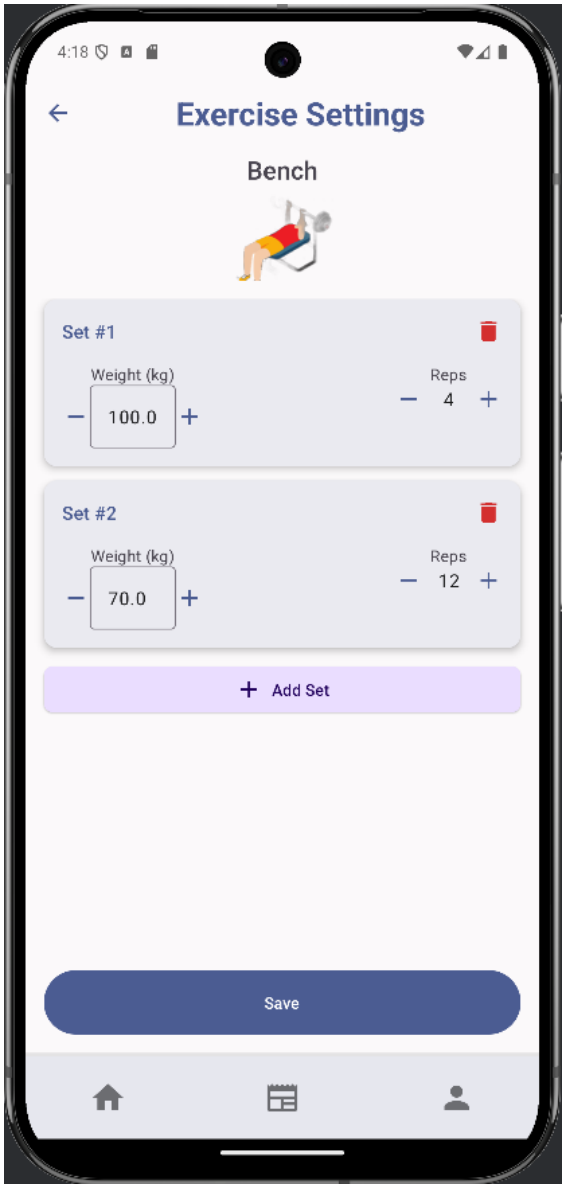
Η πέμπτη οθόνη της εφαρμογής είναι η οθόνη επιλογής άσκησης. Εδώ, ο χρήστης μπορεί να φιλτράρει τις ασκήσεις ανάμεσα σε “Upper Body”, “Lower Body” ή

“Cardio”, ενώ υπάρχει και πεδίο αναζήτησης για γρήγορη εύρεση μιας συγκεκριμένης άσκησης. Για κάθε διαθέσιμη άσκηση παρουσιάζεται μια σύντομη περιγραφή, ώστε ο χρήστης να κατανοήσει γρήγορα σε ποιους μύες εστιάζει η άσκηση ή τι είδους προπόνηση προσφέρει. Πατώντας το κουμπί “Add”, ο χρήστη θα μεταβεί στην επόμενη σελίδα προκειμένου να ορίσει τις επαναλήψεις και τα κιλά που θα κάνει. Στην περίπτωση όπου ο χρήστης δεν επιθυμεί τελικά να διαλέξει άλλη άσκηση γυμναστικής, μπορεί πέρα από το navigation back button, να πατήσει το πάνω κουμπί που βρίσκεται αριστερά πάνω, όπου θα βρεθεί στην προηγούμενη οθόνη.



4.4.7 Οθόνη SetRepsScreen

Η έκτη οθόνη της εφαρμογής είναι η οθόνη με τις περεταίρω επιλογές για μια άσκηση. Σε αυτήν την οθόνη, στο πάνω μέρος προβάλλεται το όνομα και η εικόνα της άσκησης, την οποία ο χρήστης διάλεξε. Σε αυτή την οθόνη ο χρήστης καθορίζει τον αριθμό των σετ (Set #1, Set #2 κ.λπ.), τα κιλά και τον αριθμό επαναλήψεων για κάθε σετ. Πατώντας το "+ Add Set" προστίθενται νέα σετ, ενώ υπάρχει και δυνατότητα διαγραφής ανεπιθύμητων σετ. Με την ολοκλήρωση των αλλαγών, επιλέγεται το κουμπί "Save" αποθηκεύονται τα σετς και ο χρήστης οδηγείται στην οθόνη "CurrentStatus", προκειμένου είτε να συνεχίσει είτε να τερματίσει την προπόνηση. Στην περίπτωση όπου ο χρήστης επιθυμεί να διάλεξη άλλη άσκηση, μπορεί πέρα από το navigation back button, να πατήσει το πάνω κουμπί που βρίσκεται αριστερά πάνω, όπου θα βρεθεί στην προηγούμενη οθόνη.

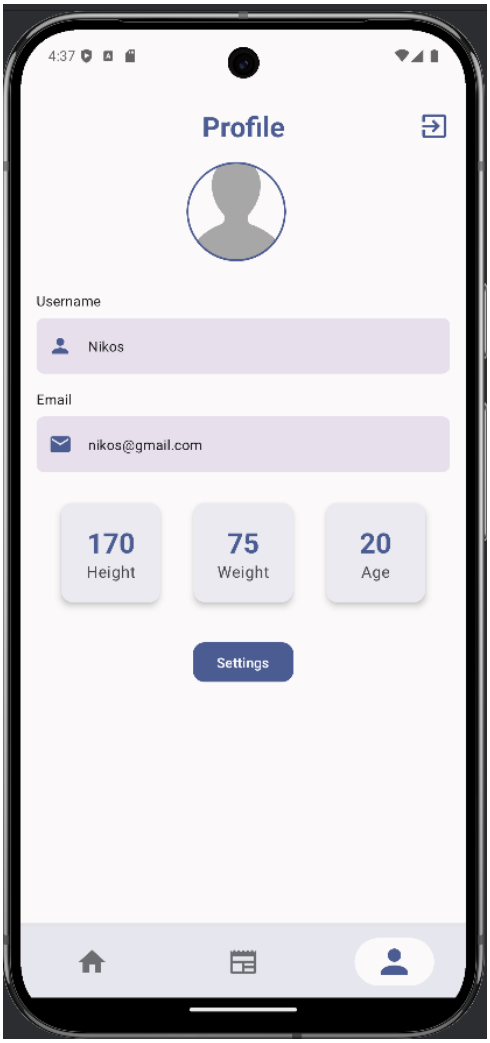


4.4.8 Οθόνη News

Η έβδομη οθόνη της εφαρμογής είναι η οθόνη ειδήσεων. Σε αυτήν, ο χρήστης συναντά μια ροή (feed) άρθρων και ενημερώσεων από τον χώρο του αθλητισμού. Τα νέα αυτά τραβιούνται απευθείας από το διαδίκτυο, συγκεκριμένα του News API, προκειμένου να περιέχει πιο πρόσφατο περιεχόμενο. Κάθε νέα είδηση συνοδεύεται από σχετική εικόνα και μια σύντομη περιγραφή, επιτρέποντας στον χρήστη να ενημερωθεί γρήγορα για θέματα σχετικά με την αθλητική επικαιρότητα.

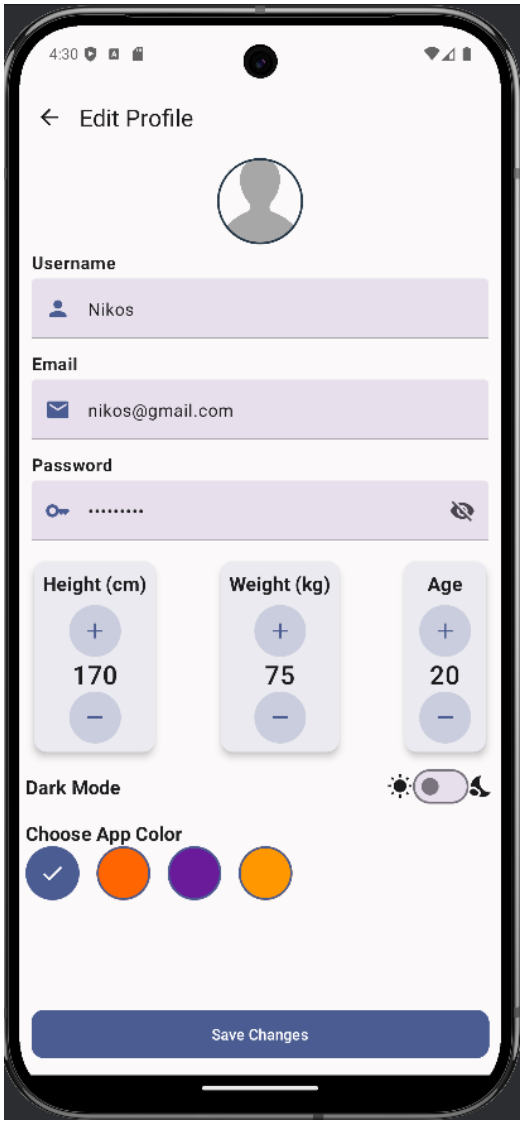
4.4.9 Οθόνη MyProfile

Η όγδοη οθόνη της εφαρμογής είναι η οθόνη προφίλ. Σε αυτήν, εμφανίζονται το όνομα χρήστη, το email και τα βασικά χαρακτηριστικά του προφίλ, τα οποία είναι ύψος, βάρος και ηλικία. Μέσω των διαθέσιμων επιλογών (ύψος, βάρος, ηλικία), ο χρήστης μπορεί να εμπλουτίσει τις πληροφορίες του προφίλ του. Επιπλέον, υπάρχει ένα κεντρικό κουμπί “Settings” για την πρόσβαση σε περαιτέρω ρυθμίσεις προκειμένου για κάνει ο χρήστης κάποια περαιτέρω διαφοροποίηση των στοιχείων του ή της εμφάνισης της εφαρμογής. Ένα ακόμη χαρακτηριστικό στοιχείο το οποίο περιέχει η συγκεκριμένη οθόνη είναι ότι πάνω δεξιά περιέχει ένα κουμπί εξόδου προκειμένου εάν επιθυμεί ο χρήστης να αποσυνδεθεί από τον λογαριασμό του. Βασική προϋπόθεση προκειμένου ο χρήστης να μεταβεί σε αυτή την οθόνη είναι να έχεις συνδεθεί με έναν λογαριασμό. Σε περίπτωση όπου ο χρήστης έχει συνδεθεί ως guest, πατώντας στην επιλογή του προφίλ , θα του εμφανίσει ένα μήνυμα ρωτώντας τον εάν επιθυμεί να συνδεθεί με κάποιον λογαριασμό και μέσω του κουμπιού “Login” θα τον οδηγήσει στην οθόνη του login.



4.4.10 Οθόνη ProfileSettings

Η ένατη οθόνη της εφαρμογής είναι η οθόνη επεξεργασίας προφίλ. Σε αυτήν την οθόνη, ο χρήστης μπορεί να επεξεργαστεί τις βασικές του πληροφορίες, όπως το όνομα χρήστη , το email, καθώς και τον κωδικό πρόσβασης. Επιπλέον ο χρήστης μπορεί να προσαρμόσει το ύψος, το βάρος και την ηλικία με βάση την τρέχουσα στοιχεία του, προκειμένου το προφίλ του να είναι συγχρονισμένο με την πραγματικότητα. Πέρα από τη διαφοροποίηση των προσωπικών στοιχείων, ο χρήστης μπορεί να κάνει προσαρμογές στην εμφάνιση της εφαρμογής, επιλέγοντας εάν επιθυμεί η εφαρμογή να είναι σε dark ή light mode καθώς και να διαλέξει ένα χρώμα από τα τέσσερα χρώματα της παλέτας. Με άλλα λόγια μέσα από τις ρυθμίσεις του προφίλ ,ο χρήστης μπορεί να διαλέξει ανάμεσα σε 8 διαφορετικούς επιλογές για την εμφάνιση της εφαρμογής. Ότι αλλαγή πραγματοποιήσει ο χρήστης στο τέλος χρειάζεται να πατήσει το κουμπί που βρίσκεται στο κάτω μέρος “Save Changes”.



Κεφάλαιο 5: Προκλήσεις

Κατά την ανάπτυξη της εφαρμογής, προέκυψαν αρκετές προκλήσεις που έπρεπε να αντιμετωπιστούν για να εξασφαλιστεί η σωστή λειτουργικότητα και η διατήρηση μιας καθαρής αρχιτεκτονικής. Οι δυσκολίες αυτές εντοπίστηκαν κυρίως στον σχεδιασμό της αρχιτεκτονικής, την ενσωμάτωση των εξαρτήσεων και τη σύνδεση με εξωτερικές υπηρεσίες, όπως το News API.

Η διαχείριση των εξαρτήσεων αποτέλεσε ένα από τα μεγαλύτερα εμπόδια κατά την ανάπτυξη. Αρχικά, ήταν δύσκολο να κατανοήσουμε πώς πρέπει να δομηθεί η αρχιτεκτονική μιας εφαρμογής με υψηλά επίπεδα αφαίρεσης και modularity. Η απόφαση να υιοθετήσουμε μια προσέγγιση βασισμένη στο MVVM (Model-View-ViewModel) με καθαρό διαχωρισμό επιπέδων, όπως entities, DAOs, repositories και ViewModels, απαιτούσε ιδιαίτερη κατανόηση των ροών δεδομένων και του τρόπου που αυτές πρέπει να διαχειρίζονται. Η ανάγκη για έναν καθαρό μηχανισμό διαχείρισης εξαρτήσεων οδήγησε στη δημιουργία του AppContainer, το οποίο εξυπηρετεί ως κεντρικός διαχειριστής των εξαρτήσεων της εφαρμογής. Ωστόσο, η χειροκίνητη υλοποίηση του Dependency Injection χωρίς τη χρήση βιβλιοθηκών όπως το Hilt ήταν απαιτητική. Η πρόκληση δεν περιοριζόταν μόνο στην αρχικοποίηση των εξαρτήσεων, όπως το Retrofit, το Room Database και τα Repositories, αλλά και στη διασφάλιση ότι τα ViewModel λαμβάνουν τις εξαρτήσεις τους σωστά μέσω του ViewModelProviderFactory. Η διαδικασία αυτή απαιτούσε αρκετές δοκιμές και επαναλήψεις για να λειτουργήσει σωστά και να διατηρηθεί η ευκολία συντήρησης της εφαρμογής.

Η χρήση του News API για την ανάκτηση αθλητικών ειδήσεων αποτέλεσε μια ακόμη σημαντική πρόκληση. Ο κύριος στόχος ήταν η ομαλή ενσωμάτωσή του στο υπάρχον σύστημα, ώστε τα δεδομένα να μπορούν να μεταφερθούν από το API στο UI χωρίς να διαταραχθεί η αρχιτεκτονική της εφαρμογής. Αυτό απαιτούσε την κατανόηση της δομής του API, η οποία περιλάμβανε την ανάλυση της JSON απόκρισης και τη δημιουργία των κατάλληλων οντοτήτων, μια διαδικασία πιο περίπλοκη από το αναμενόμενο, καθώς έπρεπε να αντιστοιχίσουμε σωστά τις δομές δεδομένων στις ανάγκες της εφαρμογής. Επιπλέον, η διαχείριση αιτήσεων και σφαλμάτων ήταν απαραίτητη για την αξιοπιστία της εφαρμογής, περιλαμβάνοντας τη σωστή υλοποίηση της διεπαφής του Retrofit και τη διαχείριση πιθανών σφαλμάτων, όπως η αποτυχία σύνδεσης ή η επιστροφή μη έγκυρων δεδομένων. Τέλος, η εναρμόνιση με το ViewModel μέσω του Repository απαιτούσε προσεκτική σχεδίαση, ώστε τα δεδομένα να εκτίθενται με συνέπεια μέσω StateFlow, χωρίς να επηρεάζεται η απόδοση ή η εμπειρία χρήστη.