Universitatea *Transilvania* din Brașov
Facultatea de Matematică și Informatică
Departamentul de Matematică și Informatică

FitTracker Android
Aplicație Mobilă pentru Fitness

Grupa 10LF333

Autori:
Tudose Alexandru Vasile
Vîlcu Andreea

June 2025

Cuprins

1 Introducere

1.1 Descrierea Aplicației

FitTracker este o aplicație nativă pentru platforma Android, dezvoltată în limbajul Kotlin, care are ca scop facilitarea monitorizării activităților fizice și a gestionării antrenamentelor. Aplicația oferă funcționalități precum organizarea exercițiilor, urmărirea progresului utilizatorului și accesul la informații nutriționale provenite din surse externe.

Aplicația prezintă o arhitecturaă MVVM (Model-View-ViewModel) și se distinge prin integrarea seamless între datele locale și sursele externe de informații.

1.2 Obiectivele Proiectului

- Implementarea unei arhitecturi moderne Android folosind MVVM pattern
- Integrarea unei baze de date locale folosind Room
- Implementarea autentificării utilizatorilor
- Conectarea la surse externe prin API-uri pentru preluarea datelor despre exerciții și nutriție
- Crearea unei interfețe intuitive și user-friendly
- Gestionarea setărilor utilizator prin SharedPreferences

1.3 Tehnologii Utilizate

- **Limbaj:** Kotlin
- **Framework:** Android SDK (API level 24-35)
- **Arhitectură:** MVVM (Model-View-ViewModel)
- **Baza de date:** Room Database
- **Networking:** Retrofit + OkHttp
- **Navigation:** Android Navigation Component
- **UI:** Material Design Components
- **Storage:** SharedPreferences
- **Coroutines:** Pentru operațiile asincrone

2 Arhitectura Aplicației

2.1 Structura Generală

Aplicația respectă principiile arhitecturii MVVM și este organizată în următoarele module principale:

- **UI Layer:** Fragmente, Adaptere, ViewModels
- **Domain Layer:** Modele de date și logica de business
- **Data Layer:** Repository pattern, DAO-uri, API services

2.2 Dependency Injection

Pentru injecția de dependențe, aplicația folosește un approach manual prin clasa **FitTrackerApplication**, care gestionează instanțele de repository-uri și database.

```
class FitTrackerApplication : Application() {
    private val applicationScope = CoroutineScope(SupervisorJob() +
    Dispatchers.IO)

    val database by lazy { AppDatabase.getDatabase(this) }
    val userRepository by lazy { UserRepository(database.userDao()) }
    val exerciseRepository by lazy { ExerciseRepository(database.
    exerciseDao()) }
    val userPreferencesManager by lazy { UserPreferencesManager(this) }
}
```

Listing 1: FitTrackerApplication - Dependency Injection

3 Componente Principale

3.1 Autentificarea Utilizatorilor

Sistemul de autentificare este implementat folosind un flow în două etape:

3.1.1 Login Fragment

- Validare email și parolă
- Integrare cu AuthViewModel pentru gestionarea stării
- Hash-uirea parolelor pentru securitate

3.1.2 Register Flow (2 Step-uri)

- **Step 1:** Email și parolă
- **Step 2:** Informații personale (nume, prenume, vârstă)

```
class AuthViewModel(private val userRepository: UserRepository) :
    ViewModel() {
    private val _uiState = MutableStateFlow(AuthUiState())
    val uiState: StateFlow<AuthUiState> = _uiState

    fun login(email: String, password: String) {
        viewModelScope.launch {
            _uiState.value = _uiState.value.copy(isLoading = true)
            val result = userRepository.loginUser(email, password)
            // Handle result...
        }
    }
}
```

Listing 2: AuthViewModel - Logica de autentificare

3.2 Navigation Component

Aplicația folosește Android Navigation Component cu două graphuri principale:

- **auth_nav_graph:** Pentru flow-ul de autentificare
- **profile_nav_graph:** Pentru funcționalitățile principale

Navigation se face prin Safe Args pentru type-safety:

```
<action
    android:id="@+id/action_login_to_profile"
    app:destination="@id/profile_nav_graph">
    <argument
        android:name="email"
        app:argType="string" />
</action>
```

Listing 3: Navigation Graph Example

4 Baza de Date Locală

4.1 Room Database

Aplicația folosește Room pentru persistența datelor locale, cu următoarele entități principale:

- **UserEntity:** Informații utilizatori
- **ExerciseEntity:** Exerciții disponibile
- **WorkoutEntity:** Antrenamente planificate
- **WorkoutLogEntity:** Antrenamente completate
- **UserProgressEntity:** Progresul utilizatorului

```
@Entity(tableName = "users")
data class UserEntity(
    @PrimaryKey val id: String,
    val username: String,
    val email: String,
    val passwordHash: String,
    val age: Int? = null,
    val fitnessLevel: FitnessLevel = FitnessLevel.BEGINNER,
    val isLoggedIn: Boolean = false
)
```

Listing 4: UserEntity - Exemplu de entitate Room

4.2 DAO (Data Access Object)

Fiecare entitate are propriul DAO pentru operațiile CRUD:

```
@Dao
interface UserDao {
    @Insert
    suspend fun insertUser(user: UserEntity)

    @Query("SELECT * FROM users WHERE email = :email LIMIT 1")
    suspend fun getUserByEmail(email: String): UserEntity?

    @Query("SELECT * FROM users WHERE isLoggedIn = 1 LIMIT 1")
    fun observeLoggedInUser(): Flow<UserEntity?>
}
```

Listing 5: UserDao - Operații database

4.3 Repository Pattern

Repository-urile abstractizează accesul la date și oferă o interfață curată pentru ViewModels:

```
class UserRepository(private val userDao: UserDao) {
    fun observeLoggedInUser(): Flow<UserEntity?> = userDao.
   observeLoggedInUser()

    suspend fun registerUser(email: String, password: String,
                             firstName: String, lastName: String, age: Int
   ?): Result<UserEntity> {
        // Implementation...
    }
}
```

Listing 6: UserRepository

5 SharedPreferences

Pentru gestionarea setărilor utilizator, aplicația implementează `UserPreferencesManager`:

```
class UserPreferencesManager(context: Context) {
    private val sharedPreferences = context.getSharedPreferences(
    PREFS_NAME, Context.MODE_PRIVATE)

    fun setDarkMode(enabled: Boolean) {
        putBoolean(KEY_DARK_MODE, enabled)
    }

    fun getDarkMode(): Boolean = getBoolean(KEY_DARK_MODE, false)

    fun setUnits(units: Units) {
        putString(KEY_UNITS, units.name)
    }
}
```

Listing 7: UserPreferencesManager

Setările includ:

- Dark Mode
- Unități de măsură (Metric/Imperial)
- Setări notificări
- Timp de odihnă implicit
- Obiective săptămânale

6 HTTP Requests și API Integration

6.1 API-uri Externe Utilizate

Aplicația consumă două API-uri externe prin API Ninjas:

1. **Exercises API:** Pentru exerciții externe
2. **Nutrition API:** Pentru informații nutriționale

6.2 Retrofit Configuration

```
object ApiManager {
    private const val BASE_URL_NINJA = "https://api.api-ninjas.com/v1/"
    const val API_KEY = "your-api-key"

    private val okHttpClient = OkHttpClient.Builder()
        .addInterceptor(HttpLoggingInterceptor().apply {
            level = HttpLoggingInterceptor.Level.BODY
        })
        .addInterceptor { chain ->
            val originalRequest = chain.request()
            val newRequest = originalRequest.newBuilder()
                .addHeader("X-Api-Key", API_KEY)
                .build()
            chain.proceed(newRequest)
```

```
        }
        .build()

    private val retrofitNinja = Retrofit.Builder()
        .baseUrl(BASE_URL_NINJA)
        .client(okHttpClient)
        .addConverterFactory(GsonConverterFactory.create())
        .build()
}
```

Listing 8: ApiManager - Configurare Retrofit

6.3 API Services

```
interface ExerciseApiService {
    @GET("exercises")
    suspend fun getExercisesByMuscle(
        @Query("muscle") muscle: String,
        @Query("offset") offset: Int = 0
    ): Response<List<ExerciseApiModel>>

    @GET("exercises")
    suspend fun getExercisesByType(
        @Query("type") type: String
    ): Response<List<ExerciseApiModel>>
}
```

Listing 9: ExerciseApiService

6.4 Repository pentru API

```
class ApiRepository {
    suspend fun getExercisesByMuscle(muscle: String): Result<List<
    ExerciseApiModel>> {
        return withContext(Dispatchers.IO) {
            try {
                val response = ApiManager.exerciseApiService.
    getExercisesByMuscle(muscle)
                if (response.isSuccessful) {
                    Result.success(response.body() ?: emptyList())
                } else {
                    Result.failure(Exception("API Error: ${response.code
    ()}"))
                }
            } catch (e: Exception) {
                Result.failure(e)
            }
        }
    }
}
```

Listing 10: ApiRepository

7 RecyclerView și Adaptere

Aplicația folosește multiple RecyclerView-uri cu adaptere custom:

7.1 ExerciseAdapter

```
class ExerciseAdapter(
    private val onExerciseClick: (ExerciseEntity) -> Unit
) : ListAdapter<ExerciseEntity, ExerciseAdapter.ExerciseViewHolder>(
    ExerciseDiffCallback()) {

    override fun onBindViewHolder(holder: ExerciseViewHolder, position:
    Int) {
        holder.bind(getItem(position))
    }

    class ExerciseViewHolder(itemView: View, private val onExerciseClick
    : (ExerciseEntity) -> Unit)
        : RecyclerView.ViewHolder(itemView) {
        // Binding logic...
    }
}
```

Listing 11: ExerciseAdapter - Adapter pentru exerciții locale

7.2 Adaptere Multiple

- **ExerciseAdapter:** Pentru exercițiile locale
- **ExternalExerciseAdapter:** Pentru exercițiile din API
- **NutritionAdapter:** Pentru informațiile nutriționale

8 Shape Drawables

Aplicația folosește multiple shape drawables pentru un design modern:

8.1 Button Gradient

```
<shape xmlns:android="http://schemas.android.com/apk/res/android"
    android:shape="rectangle">
    <gradient
        android:startColor="#9C27B0"
        android:endColor="#6A1B9A"
        android:angle="45" />
    <corners android:radius="12dp" />
    <stroke android:width="1dp" android:color="#4A148C" />
</shape>
```

Listing 12: button_gradient.xml

8.2 Rounded Background

```
<shape xmlns:android="http://schemas.android.com/apk/res/android"
    android:shape="rectangle">
    <solid android:color="#2196F3" />
    <corners android:radius="16dp"/>
    <stroke android:width="1dp" android:color="#1976D2"/>
</shape>
```

Listing 13: rounded_background.xml

9 Interfața Utilizator

9.1 Material Design

Aplicația folosește Material Design 3 cu:

- CardView pentru listări
- Chips pentru filtrare
- FloatingActionButton pentru acțiuni rapide
- Material Toolbar cu navigation
- TextInputLayout pentru input-uri

9.2 Teme și Culori

- Tema principală: Purple (#9C27B0)
- Support pentru Dark Mode prin SharedPreferences
- Gradient backgrounds pentru butoane importante

10 Funcționalități Implementate

10.1 Core Features

1. **Autentificare completă** cu register în 2 pași
2. **Gestionarea exercițiilor** - browse, search, filter, add custom
3. **Integrare API externe** pentru exerciții și nutriție
4. **Setări utilizator** complexe prin SharedPreferences
5. **Baza de date locală** cu multiple entități și relații

10.2 Advanced Features

- Real-time search și filtering
- Custom exercise creation
- External API data browsing
- Settings management cu UI intuitiv
- Navigation cu Safe Args
- Error handling comprehensive

11 Managementul Erorilor și Testare

11.1 Error Handling

- Try-catch blocks pentru operațiile database
- Result wrapper pentru API calls
- User-friendly error messages
- Loading states în UI

11.2 Crash Prevention

- Null safety prin Kotlin
- Validare input utilizator
- Safe navigation cu Navigation Component
- Proper lifecycle management

12 Performanță și Optimizări

12.1 Database Optimizations

- Indexuri pe coloane frecvent utilizate
- Coroutines pentru operații asincrone
- Flow pentru observarea datelor
- Repository pattern pentru caching

12.2 UI Optimizations

- RecyclerView cu DiffUtil pentru eficiență
- ViewBinding pentru type safety
- Lazy initialization pentru repositories
- Proper lifecycle awareness

13 Dezvoltări ulterioare

Aplicația poate fi extinsă cu:

- Sincronizare cloud
- Notificări push
- Sharing social
- Analytics și tracking avansat

14 Anexe

14.1 Structura Proiectului

```
app/
|-- src/main/java/com/example/fittracker_android/
|   |-- data/
|   |   |-- api/
|   |   |-- local/
|   |   |-- model/
|   |   +-- repository/
|   |-- ui/
|   |   |-- auth/
|   |   |-- exercises/
|   |   +-- settings/
|   +-- FitTrackerApplication.kt
|-- src/main/res/
|   |-- drawable/
|   |-- layout/
|   |-- navigation/
|   +-- values/
+-- build.gradle.kts
```

14.2 Dependencies

Principalele dependențe utilizate:

- Room: 2.6.1

- Navigation: 2.7.7
- Retrofit: 2.9.0
- Coroutines: 1.7.3
- Material Components: 1.12.0