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

FitTracker Android

Aplicație Mobilă pentru Fitness

Grupa 10LF333

Studenți:
Tudose Alexandru Vasile
Vîlcu Andreea

Iunie 2025

Cuprins

1 Prezentarea proiectului, ce își propune, ce probleme rezolvă

1.1 Descrierea Proiectului

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

1.2 Problemele Rezolvate

Aplicația adresează următoarele probleme comune în domeniul fitness-ului:

- **Dezorganizarea antrenamentelor:** Utilizatorii pot organiza și gestiona exercițiile într-o bază de date structurată cu funcționalități avansate de căutare și filtrare
- **Lipsa informațiilor nutriționale:** Integrarea cu API-uri externe pentru date nutriționale actualizate în timp real
- **Urmărirea progresului:** Sistem complet de logging pentru antrenamente și progres personal cu analytics
- **Accesul limitat la exerciții:** Bază de date locală cu exerciții predefinite + posibilitatea de a adăuga exerciții custom cu detalii complete
- **Lipsa motivației:** Sistem de realizări și statistici pentru încurajarea continuității cu gamification elements

1.3 Obiectivele Aplicației

- Oferirea unei platforme centralizate pentru gestionarea activităților fizice
- Implementarea unei arhitecturi moderne Android folosind MVVM pattern cu Dependency Injection
- Integrarea seamless între datele locale și sursele externe de informații prin Repository Pattern
- Crearea unei interfețe intuitive și user-friendly folosind Material Design 3
- Asigurarea securității datelor utilizatorului prin autentificare și criptare
- Implementarea unei arhitecturi scalabile și maintainable

2 Tehnologiile folosite

2.1 Tehnologii Core

- **Limbaj de programare:** Kotlin
- **Platform:** Android SDK (API level 24-35)
- **Arhitectură:** MVVM (Model-View-ViewModel)
- **Build System:** Gradle cu Kotlin DSL
- **Dependency Injection:** Manual prin Application class

2.2 Database și Storage

- **Baza de date locală:** Room Database (SQLite)
- **Type Converters:** Pentru entități complexe (enum-uri, liste)
- **Storage preferences:** SharedPreferences pentru setări utilizator
- **Coroutines:** Pentru operațiuni asincrone database cu Flow

2.3 Networking și API

- **HTTP Client:** Retrofit + OkHttp
- **JSON Parsing:** Gson Converter
- **API Integration:** API Ninjas (Exercises & Nutrition)
- **Logging:** HttpLoggingInterceptor pentru debugging
- **Error Handling:** Result wrapper pattern pentru API calls

2.4 UI și Navigation

- **Design System:** Material Design 3
- **Navigation:** Android Navigation Component cu Safe Args
- **UI Components:** Material Components (CardView, Chips, FAB)
- **Layouts:** ConstraintLayout, RecyclerView cu DiffUtil
- **ViewBinding:** Pentru type-safe view access

2.5 Arhitectură și Pattern

- **Design Pattern:** Repository Pattern pentru abstractizarea datelor
- **State Management:** StateFlow și LiveData pentru reactive programming
- **Dependency Injection:** Manual prin FitTrackerApplication class
- **Lifecycle Management:** ViewModel și Lifecycle-aware components

```
dependencies {
    // Core Android
    implementation("androidx.core:core-ktx:1.16.0")
    implementation("androidx.appcompat:appcompat:1.7.0")
    implementation("com.google.android.material:material:1.12.0")

    // Navigation
    implementation("androidx.navigation:navigation-fragment-ktx:2.7.7")
    implementation("androidx.navigation:navigation-ui-ktx:2.7.7")

    // Room Database
    implementation("androidx.room:room-runtime:2.6.1")
    implementation("androidx.room:room-ktx:2.6.1")
    kapt("androidx.room:room-compiler:2.6.1")

    // Networking
    implementation("com.squareup.retrofit2:retrofit:2.9.0")
    implementation("com.squareup.retrofit2:converter-gson:2.9.0")
    implementation("com.squareup.okhttp3:logging-interceptor:4.12.0")

    // Coroutines
    implementation("org.jetbrains.kotlinx:kotlinx-coroutines-android
    :1.7.3")
}
```

Listing 1: Configurarea dependențelor în build.gradle.kts

```
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 apiRepository by lazy { ApiRepository() }
    val userPreferencesManager by lazy { UserPreferencesManager(this) }
}
```

Listing 2: FitTrackerApplication - Dependency Injection

3 Baza de date: diagrama bazei de date + scurtă prezentare a tabelelor și a relațiilor dintre ele

3.1 Diagrama Bazei de Date

Entitate	Chei	Relații
Users	PK: id	1:N cu WorkoutLogs, UserProgress
Exercises	PK: id	1:N cu ExerciseLogs
Workouts	PK: id, FK: userId	1:N cu WorkoutLogs
WorkoutLogs	PK: id, FK: userId, workoutId	1:N cu ExerciseLogs
ExerciseLogs	PK: id, FK: workoutLogId, exerciseId	1:N cu ExerciseSets
ExerciseSets	PK: id, FK: exerciseLogId	N:1 cu ExerciseLogs
UserProgress	PK: id, FK: userId	N:1 cu Users

3.2 Prezentarea Tabelelor

3.2.1 Users

Stochează informațiile utilizatorilor înregistrați cu securitate crescută:

```
@Entity(tableName = "users")
data class UserEntity(
    @PrimaryKey val id: String,
    val username: String,
    val email: String,
    val passwordHash: String, // SHA-256 hash for security
    val age: Int? = null,
    val height: Float? = null,
    val weight: Float? = null,
    val gender: Gender? = null,
    val fitnessLevel: FitnessLevel = FitnessLevel.BEGINNER,
    val isLoggedIn: Boolean = false,
    val createdAt: Long = System.currentTimeMillis()
)
```

Listing 3: UserEntity - Tabelul utilizatorilor

```
@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?>

    @Update
    suspend fun updateUser(user: UserEntity)
}
```

Listing 4: UserDao - Operații database pentru utilizatori

3.2.2 Exercises

Catalogul de exerciții disponibile cu suport pentru exerciții custom:

```
@Entity(tableName = "exercises")
data class ExerciseEntity(
    @PrimaryKey val id: String,
```

```
    val name: String,
    val description: String,
    val instructions: String,
    val type: ExerciseType,
    val muscleGroups: String, // comma-separated pentru flexibilitate
    val difficulty: DifficultyLevel,
    val equipmentRequired: String,
    val isCustom: Boolean = false,
    val createdBy: String? = null,
    val imageUrl: String? = null
)
```

Listing 5: ExerciseEntity - Catalogul exercițiilor

3.2.3 WorkoutLogs

Înregistrarea antrenamentelor completate cu metrici detaliate:

```
@Entity(tableName = "workout_logs")
data class WorkoutLogEntity(
    @PrimaryKey val id: String,
    val userId: String,
    val workoutId: String?,
    val startTime: Long,
    val endTime: Long,
    val duration: Int, // in minute
    val caloriesBurned: Int? = null,
    val perceivedEffort: Int? = null, // scala 1-10
    val mood: Mood? = null,
    val notes: String? = null
)
```

Listing 6: WorkoutLogEntity - Antrenamente completate

3.3 Relațiile dintre Tabele

- **Users ↔ WorkoutLogs:** Un utilizator poate avea multiple antrenamente (1:N)
- **WorkoutLogs ↔ ExerciseLogs:** Un antrenament conține multiple exerciții (1:N)
- **Exercises ↔ ExerciseLogs:** Un exercițiu poate fi folosit în multiple antrenamente (1:N)
- **ExerciseLogs ↔ ExerciseSets:** Un exercițiu din antrenament poate avea multiple seturi (1:N)
- **Users ↔ UserProgress:** Un utilizator poate avea multiple înregistrări de progres (1:N)

3.4 Repository Pattern Implementation

```
class ExerciseRepository(private val exerciseDao: ExerciseDao) {
    fun getAllExercises(): Flow<List<ExerciseEntity>> = exerciseDao.
   getAllExercises()

```

```
    suspend fun createCustomExercise(
        name: String, description: String, instructions: String,
        type: ExerciseType, muscleGroups: List<MuscleGroup>
    ): Result<ExerciseEntity> {
        return try {
            val exercise = ExerciseEntity(
                id = UUID.randomUUID().toString(),
                name = name,
                description = description,
                instructions = instructions,
                type = type,
                muscleGroups = muscleGroups.joinToString(","),
                difficulty = DifficultyLevel.BEGINNER,
                equipmentRequired = "",
                isCustom = true
            )
            exerciseDao.insertExercise(exercise)
            Result.success(exercise)
        } catch (e: Exception) {
            Result.failure(e)
        }
    }
}
```

Listing 7: ExerciseRepository - Repository pattern pentru exerciții

4 Prezentarea API-ului: endpoint-urile + scurtă descriere a operațiunilor

4.1 API-uri Externe Integrate

Aplicația consumă două API-uri externe prin platforma API Ninjas pentru date actualizate:

4.1.1 Exercises API

Base URL: `https://api.api-ninjas.com/v1/exercises`

- **GET /exercises?muscle={muscle}** - Obține exerciții pe grupe musculare
- **GET /exercises?type={type}** - Obține exerciții pe tipuri (cardio, strength, etc.)
- **GET /exercises?difficulty={level}** - Filtrează pe nivel de dificultate

```
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>>

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

Listing 8: ExerciseApiService - Interfața pentru API exerciții

4.1.2 Nutrition API

Base URL: `https://api.api-ninjas.com/v1/nutrition`

- **GET /nutrition?query={food_name}** - Obține informații nutriționale detaliate

```
interface NutritionApiService {
    @GET("nutrition")
    suspend fun getNutritionInfo(
        @Query("query") foodName: String,
        @Header("X-Api-Key") apiKey: String
    ): Response<List<NutritionApiModel>>
}
```

Listing 9: NutritionApiService - Interfața pentru API nutriție

4.2 Configurarea Retrofit

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

    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)
                .addHeader("Accept", "application/json")
                .build()
            chain.proceed(newRequest)
        }
        .connectTimeout(30, TimeUnit.SECONDS)
        .readTimeout(30, TimeUnit.SECONDS)
        .build()

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

```

```
    val exerciseApiService: ExerciseApiService = retrofitNinja.create()
    val nutritionApiService: NutritionApiService = retrofitNinja.create
    ()
}
```

Listing 10: ApiManager - Configurare Retrofit avansată

4.3 Operațiuni CRUD Locale

Aplicația implementează operațiuni CRUD complete pentru entitățile locale:

4.3.1 User Operations

- **CREATE:** Înregistrare utilizator nou cu validare completă
- **READ:** Autentificare și obținere profil cu observare reactivă
- **UPDATE:** Modificare informații profil în timp real
- **DELETE:** Ștergere cont utilizator cu cleanup complet

4.3.2 Exercise Operations

- **CREATE:** Adăugare exerciții custom cu validare
- **READ:** Listare, căutare și filtrare exerciții avansată
- **UPDATE:** Modificare exerciții custom
- **DELETE:** Ștergere exerciții custom cu verificări

4.3.3 Workout Operations

- **CREATE:** Crearea unui nou antrenament cu template-uri
- **READ:** Vizualizarea antrenamentelor și istoricului detaliat
- **UPDATE:** Modificarea antrenamentelor în progres
- **DELETE:** Ștergerea antrenamentelor cu arhivare

```
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
    ()} - ${response.message()}"))
                }
            } catch (e: Exception) {
```

```
                Result.failure(e)
            }
        }
    }

    suspend fun getNutritionInfo(foodName: String): Result<List<
    NutritionApiModel>> {
        return withContext(Dispatchers.IO) {
            try {
                val response = ApiManager.nutritionApiService.
    getNutritionInfo(foodName, ApiManager.API_KEY)
                if (response.isSuccessful) {
                    Result.success(response.body() ?: emptyList())
                } else {
                    Result.failure(Exception("Nutrition API Error: ${
    response.code()}"))
                }
            } catch (e: Exception) {
                Result.failure(e)
            }
        }
    }
}
```

Listing 11: ApiRepository - Repository pentru API-uri externe

5 Prezentare despre cum poate fi utilizată aplicația: tipuri de utilizatori, ce vede fiecare, autentificare etc

5.1 Fluxul de Autentificare

Aplicația implementează un sistem complet de autentificare în două etape cu securitate avansată:

5.1.1 Procesul de Login

1. Utilizatorul introduce email-ul și parola
2. Sistemul validează credențialele în baza de date locală
3. Parola este hash-uită folosind SHA-256 pentru securitate
4. La autentificare reușită, utilizatorul este redirectat către ecranul principal
5. State management prin AuthViewModel cu StateFlow

5.1.2 Procesul de Înregistrare (2 Pași)

Pasul 1: Email și Parolă

- Validarea formatului email-ului cu regex pattern
- Verificarea unicității email-ului în baza de date

- Validarea lungimii parolei (minim 6 caractere)
- Real-time validation cu error messages

Pasul 2: Informații Personale

- Nume și prenume (obligatoriu)
- Vârsta (opțional pentru personalizare)
- Finalizarea înregistrării și redirectarea către login

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

    fun register(email: String, password: String, firstName: String,
                 lastName: String, age: Int?) {
        viewModelScope.launch {
            _uiState.value = _uiState.value.copy(isLoading = true, error
    = null)

            // Validate input
            if (!isValidEmail(email)) {
                _uiState.value = _uiState.value.copy(
                    isLoading = false,
                    error = "Email format invalid"
                )
                return@launch
            }

            val result = userRepository.registerUser(
                email, password, firstName, lastName, age
            )

            _uiState.value = if (result.isSuccess) {
                _uiState.value.copy(isLoading = false,
   registrationSuccess = true)
            } else {
                _uiState.value.copy(
                    isLoading = false,
                    error = result.exceptionOrNull()?.message ?: "
   Registration failed"
                )
            }
        }
    }

    private fun isValidEmail(email: String): Boolean {
        return Patterns.EMAIL_ADDRESS.matcher(email).matches()
    }
}
```

Listing 12: AuthViewModel - Gestionarea avansată a autentificării

5.2 Navigation Component Implementation

```
<navigation android:id="@+id/auth_nav_graph"
            app:startDestination="@id/loginFragment">

    <fragment android:id="@+id/loginFragment"
              android:name="com.example.fittracker.ui.auth.LoginFragment
    ">
        <action android:id="@+id/action_login_to_register_step1"
                app:destination="@id/registerStep1Fragment"
                app:enterAnim="@anim/slide_in_right"
                app:exitAnim="@anim/slide_out_left" />
        <action android:id="@+id/action_login_to_profile"
                app:destination="@id/profile_nav_graph">
            <argument android:name="userId" app:argType="string" />
        </action>
    </fragment>

    <include app:graph="@navigation/profile_nav_graph" />
</navigation>
```

Listing 13: Navigation Graph - Structura avansată a navigării

5.3 Tipuri de Utilizatori

5.3.1 Utilizatori Neautentificați

- Accesul limitat doar la ecranele de login și înregistrare
- Nu pot accesa funcționalitățile aplicației
- Interface simplificată cu focus pe onboarding

5.3.2 Utilizatori Autentificați

Utilizatorii autentificați au acces complet la toate funcționalitățile avansate:
Ecranul Principal (Profile):

- Dashboard cu statistici personalizate
- Acces rapid la funcționalitățile principale prin CardView
- Butoane pentru: Exerciții, Date Externe, Antrenamente, Progres, Setări
- Welcome message personalizat cu numele utilizatorului

Secțiunea Exerciții:

- Vizualizarea catalogului de exerciții locale cu RecyclerView optimizat
- Căutare în timp real și filtrare pe tip, grup muscular, dificultate
- Adăugarea de exerciții personalizate cu form validation
- Vizualizarea detaliilor fiecărui exercițiu cu imagini și instrucțiuni

Date Externe:

- Accesarea exercițiilor din API-uri externe cu loading states
- Căutarea informațiilor nutriționale cu autocomplete
- Vizualizarea citatelor motivaționale
- Cache-uirea datelor pentru offline usage

Setări Avansate:

- Configurarea aspectului (Dark Mode) cu SharedPreferences
- Gestionarea unităților de măsură (Metric/Imperial)
- Setări pentru antrenamente (timp de odihnă, obiective săptămânale)
- Resetarea setărilor cu dialog de confirmare

5.4 SharedPreferences Management

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

    companion object {
        private const val PREFS_NAME = "FitTrackerPrefs"
        private const val KEY_DARK_MODE = "dark_mode"
        private const val KEY_UNITS = "units"
        private const val KEY_WEEKLY_GOAL = "weekly_goal"
        private const val KEY_REST_TIME = "default_rest_time"
    }

    fun setDarkMode(enabled: Boolean) {
        sharedPreferences.edit().putBoolean(KEY_DARK_MODE, enabled).
    apply()
    }

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

    fun setUnits(units: Units) {
        sharedPreferences.edit().putString(KEY_UNITS, units.name).apply
    ()
    }

    fun getUnits(): Units {
        val unitName = sharedPreferences.getString(KEY_UNITS, Units.
    METRIC.name)
        return Units.valueOf(unitName ?: Units.METRIC.name)
    }

    fun setWeeklyGoal(goal: Int) {
        sharedPreferences.edit().putInt(KEY_WEEKLY_GOAL, goal).apply()
    }

```

```
    fun getWeeklyGoal(): Int = sharedPreferences.getInt(KEY_WEEKLY_GOAL, 3)
}
```

Listing 14: UserPreferencesManager - Gestionarea setărilor

5.5 RecyclerView și Adaptere Avansate

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

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): ExerciseViewHolder {
        val binding = ItemExerciseBinding.inflate(
            LayoutInflater.from(parent.context), parent, false
        )
        return ExerciseViewHolder(binding, onExerciseClick, onEditClick)
    }

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

    class ExerciseViewHolder(
        private val binding: ItemExerciseBinding,
        private val onExerciseClick: (ExerciseEntity) -> Unit,
        private val onEditClick: (ExerciseEntity) -> Unit
    ) : RecyclerView.ViewHolder(binding.root) {

        fun bind(exercise: ExerciseEntity) {
            binding.apply {
                textViewExerciseName.text = exercise.name
                textViewMuscleGroups.text = exercise.muscleGroups
                textViewDifficulty.text = exercise.difficulty.name

                // Show edit button only for custom exercises
                buttonEdit.isVisible = exercise.isCustom

                root.setOnClickListener { onExerciseClick(exercise) }
                buttonEdit.setOnClickListener { onEditClick(exercise) }

                // Set difficulty color
                chipDifficulty.setChipBackgroundColorResource(
                    when (exercise.difficulty) {
                        DifficultyLevel.BEGINNER -> R.color.green_500
                        DifficultyLevel.INTERMEDIATE -> R.color.orange_500
                        DifficultyLevel.EXPERT -> R.color.red_500
                    }
                )
            }
        }
    }
```

```
}

class ExerciseDiffCallback : DiffUtil.ItemCallback<ExerciseEntity>() {
    override fun areItemsTheSame(oldItem: ExerciseEntity, newItem:
    ExerciseEntity): Boolean {
        return oldItem.id == newItem.id
    }

    override fun areContentsTheSame(oldItem: ExerciseEntity, newItem:
    ExerciseEntity): Boolean {
        return oldItem == newItem
    }
}
```

Listing 15: ExerciseAdapter - Adapter optimizat cu DiffUtil

5.6 Shape Drawables și Material Design

```
<shape xmlns:android="http://schemas.android.com/apk/res/android"
    android:shape="rectangle">
    <gradient
        android:startColor="#9C27B0"
        android:endColor="#6A1B9A"
        android:angle="45"
        android:type="linear" />
    <corners android:radius="12dp" />
    <stroke android:width="1dp" android:color="#4A148C" />
    <padding
        android:left="16dp"
        android:right="16dp"
        android:top="12dp"
        android:bottom="12dp" />
</shape>
```

Listing 16: button_gradient.xml - Gradient pentru butoane

```
<shape xmlns:android="http://schemas.android.com/apk/res/android"
    android:shape="rectangle">
    <solid android:color="?attr/colorSurface" />
    <corners android:radius="16dp"/>
    <stroke
        android:width="1dp"
        android:color="?attr/colorOutline"/>
    <size
        android:height="120dp"
        android:width="match_parent"/>
</shape>
```

Listing 17: rounded_background.xml - Background pentru carduri

5.7 Gestionarea Stării Utilizatorului

- **Persistența login-ului:** Starea de autentificare salvată în Room Database
- **Logout automată:** La închiderea aplicației sau logout manual cu cleanup

- **Securitate:** Parolele hash-uite cu SHA-256, nu plain text storage
- **Sesiune unică:** Un singur utilizator logat simultan cu validation
- **State Management:** StateFlow pentru reactive updates în UI

6 Concluzii și contribuții

6.1 Împărțirea Task-urilor

6.1.1 Tudose Alexandru Vasile

- Implementarea arhitecturii MVVM și structurarea modulară a proiectului
- Dezvoltarea sistemului complex de autentificare cu SecurityManager
- Implementarea completă a bazei de date Room cu toate entitățile și DAO-urile
- Crearea repository pattern-ului și integrarea seamless cu ViewModels
- Implementarea sistemului de navigație cu Safe Args și deep linking
- Dezvoltarea funcționalității avansate de exerciții locale și custom cu validări
- Implementarea sistemului de logging și error handling

6.1.2 Vîlcu Andreea

- Designul complet al interfețelor utilizator cu Material Design 3
- Implementarea integrărilor complexe cu API-uri externe (Exercises & Nutrition)
- Dezvoltarea adapterelor optimizate pentru RecyclerView-uri cu DiffUtil
- Crearea shape drawable-urilor și sistemului complet de teme (Dark/Light Mode)
- Implementarea avansată a sistemului de setări cu SharedPreferences
- Testarea comprehensivă a aplicației și debugging la nivel de producție
- Optimizarea performanței UI și UX design patterns

6.1.3 Colaborarea

- Planificarea detaliată a arhitecturii și design patterns
- Code review reciproc rigoros pentru calitatea și consistența codului
- Testarea cross-feature extensivă pentru integrare perfectă
- Documentarea comună și detaliată a întregului proiect
- Pair programming pentru componentele critice

6.2 Provocări Întâmpinate și Soluții

- **Gestionarea relațiilor complexe în Room:** Soluționat prin simplificarea structurii, folosirea de Foreign Key constraints validate și implementarea de TypeConverters pentru date complexe
- **Integrarea API-urilor externe:** Gestionarea comprehensivă a erorilor de rețea, implementarea de retry mechanisms și parsing robust al răspunsurilor inconsistente
- **Navigation Component:** Învățarea și implementarea Safe Args, gestionarea complexă a back stack-ului și implementarea de deep linking
- **State Management:** Coordonarea perfectă între ViewModels și UI state folosind StateFlow, LiveData și Coroutines
- **Performance Optimization:** Implementarea de lazy loading, database indexing și UI optimizations cu DiffUtil

6.3 Cunoștințe Dobândite

6.3.1 Aspecte Tehnice Avansate

- Înțelegerea profundă și practică a arhitecturii MVVM în ecosystemul Android
- Experiență hands-on cu Room Database, TypeConverters și database migrations
- Utilizarea avansată a Coroutines pentru operațiuni asincrone și Flow pentru reactive programming
- Implementarea pattern-urilor Repository, Dependency Injection și Observer
- Integrarea complexă a API-urilor REST cu Retrofit, OkHttp și custom interceptors
- Implementarea security best practices pentru authentication și data protection

6.3.2 Best Practices și Optimization

- Structurarea scalabilă și maintainable a proiectelor Android enterprise-level
- Gestionarea expertă a lifecycle-ului Android și preventia memory leaks
- Implementarea comprehensivă a security measures pentru production apps
- Optimizarea avansată a performanței UI cu RecyclerView, DiffUtil și ViewBinding
- Error handling robust și user experience optimization
- Testing strategies și debugging techniques pentru aplicații complexe

6.4 Dezvoltări Viitoare și Scalabilitate

- **Cloud Sync:** Implementarea sincronizării datelor cu Firebase/AWS backend
- **Push Notifications:** Sistema de reminder-uri inteligente pentru antrenamente
- **Social Features:** Partajarea progresului, competiții între utilizatori și community features
- **Advanced Analytics:** Grafice interactive și statistici ML-powered pentru progres
- **Wearable Integration:** Conectarea seamless cu smartwatch-uri și fitness trackers
- **AI Recommendations:** Algoritmi de machine learning pentru sugestii personalizate de antrenamente
- **Offline-First Architecture:** Sync bidirectional și conflict resolution

6.5 Impact și Valoare Demonstrată

Proiectul demonstrează:

- Capacitatea completă de a implementa aplicații Android production-ready și scalabile
- Înțelegerea profundă a principiilor moderne de dezvoltare mobile și best practices
- Abilitatea expertă de a integra multiple surse de date (locale și externe) într-o arhitectură coerentă
- Competențe avansate în designul UX/UI modern și respectarea guideline-urilor Material Design 3
- Experiență valoroasă în colaborarea pe proiecte software complexe și management de cod
- Implementarea security measures și performance optimizations la nivel enterprise

7 Link GIT către codul proiectului

7.1 Repository GitHub

URL: https://github.com/andreeavilcu/FitTracker_Android.git

7.2 Structura Detaliată a Repository-ului

```
FitTracker_Android/
|-- app/
|   |-- src/main/java/com/example/fittracker_android/
|   |   |-- data/
|   |   |   |-- api/              # API services si modele externe
|   |   |   |   |-- models/       # Data models pentru API responses
```

```
|   |   |   |   |-- services/     # Retrofit service interfaces
|   |   |   |   `-- ApiManager.kt # Configurare Retrofit si OkHttp
|   |   |   |-- local/            # Room database si DAO-uri
|   |   |   |   |-- dao/          # Data Access Objects
|   |   |   |   |-- entities/     # Room entities
|   |   |   |   |-- converters/   # Type converters pentru Room
|   |   |   |   `-- AppDatabase.kt # Database configuration
|   |   |   |-- model/            # Data classes si domain models
|   |   |   |   |-- enums/        # Enum classes pentru tipuri
|   |   |   |   `-- dto/          # Data Transfer Objects
|   |   |   `-- repository/       # Repository pattern implementation
|   |   |       |-- UserRepository.kt
|   |   |       |-- ExerciseRepository.kt
|   |   |       `-- ApiRepository.kt
|   |   |-- ui/
|   |   |   |-- auth/             # Autentificare si navigare
|   |   |   |   |-- LoginFragment.kt
|   |   |   |   |-- RegisterStep1Fragment.kt
|   |   |   |   |-- RegisterStep2Fragment.kt
|   |   |   |   `-- AuthViewModel.kt
|   |   |   |-- exercises/        # Gestionarea exercitiilor
|   |   |   |   |-- ExercisesFragment.kt
|   |   |   |   |-- ExerciseDetailFragment.kt
|   |   |   |   |-- adapters/     # RecyclerView adapters
|   |   |   |   `-- ExerciseViewModel.kt
|   |   |   |-- external/         # Date externe din API-uri
|   |   |   |   |-- ExternalExercisesFragment.kt
|   |   |   |   |-- NutritionFragment.kt
|   |   |   |   `-- ExternalDataViewModel.kt
|   |   |   |-- profile/          # Profil utilizator
|   |   |   |   |-- ProfileFragment.kt
|   |   |   |   `-- ProfileViewModel.kt
|   |   |   |-- settings/         # Setari aplicatie
|   |   |   |   |-- SettingsFragment.kt
|   |   |   |   |-- UserPreferencesManager.kt
|   |   |   |   `-- SettingsViewModel.kt
|   |   |   `-- common/           # Componente UI comune
|   |   |       |-- BaseFragment.kt
|   |   |       `-- ViewBindingDelegate.kt
|   |   |-- utils/                # Utility classes
|   |   |   |-- Constants.kt
|   |   |   |-- Extensions.kt
|   |   |   `-- SecurityUtils.kt
|   |   `-- FitTrackerApplication.kt  # Application class cu DI
|   |-- src/main/res/
|   |   |-- drawable/             # Shape drawables si vectori
|   |   |   |-- button_gradient.xml
|   |   |   |-- rounded_background.xml
```

```
|   |   |   `-- ic_*.xml          # Vector icons
|   |   |-- layout/               # Layout-uri XML
|   |   |   |-- activity_main.xml
|   |   |   |-- fragment_*.xml    # Fragment layouts
|   |   |   `-- item_*.xml        # RecyclerView item layouts
|   |   |-- navigation/           # Navigation graphs
|   |   |   |-- auth_nav_graph.xml
|   |   |   `-- profile_nav_graph.xml
|   |   |-- values/               # Resources
|   |   |   |-- strings.xml
|   |   |   |-- colors.xml
|   |   |   |-- themes.xml        # Light/Dark themes
|   |   |   `-- dimens.xml
|   |   `-- values-night/         # Dark theme specific
|   `-- build.gradle.kts          # Module dependencies
|-- gradle/
|   `-- wrapper/                  # Gradle wrapper files
|-- .gitignore                    # Git ignore rules
|-- README.md                     # Project documentation
|-- build.gradle.kts              # Project level build script
`-- gradle.properties             # Gradle properties
```

7.3 Instrucțiuni Detaliate de Instalare

1. **Clone repository:** `git clone https://github.com/andreeavilcu/FitTracker_Android.git`
2. **Setup development environment:**
   - Deschide proiectul în Android Studio Arctic Fox sau mai nou
   - Asigură-te că ai Android SDK API 24+ instalat
   - Sync Gradle files și resolve dependencies
3. **Configurare API keys:**
   - Înregistrează-te pe API Ninjas
   - Obține API key gratuit
   - Adaugă key-ul în `ApiManager.kt` sau în `local.properties`
4. **Build și test:**
   - Build proiectul: `./gradlew build`
   - Run pe emulator sau device fizic
   - Testează toate flow-urile principale

7.4 Cerințe Sistem și Dependencies

- **Development Environment:** Android Studio Arctic Fox sau mai nou
- **Programming Language:** Kotlin 2.0.21+

- **Target Platform:** Android SDK API 24+ (Android 7.0) - API 35
- **Build System:** Gradle 8.11.1 cu Kotlin DSL
- **Internet Connection:** Necesară pentru API-uri externe și initial setup
- **Minimum RAM:** 4GB pentru development, 2GB pentru runtime
- **Storage:** Minim 1GB available space

Key Dependencies cu versiuni:

- Room Database: 2.6.1
- Navigation Component: 2.7.7
- Retrofit: 2.9.0
- OkHttp: 4.12.0
- Coroutines: 1.7.3
- Material Components: 1.12.0
- Lifecycle Components: 2.8.0