Specificarea si dezvoltarea unei aplicatii mobile numită AppTracker de tip scrapper pentru colectarea unor informații pentru aplicații mobile de pe APKMirror/F-Droid

Necula Vlad-Nicolae

Lupei Dacian-Cristian

1. Scopul aplicației

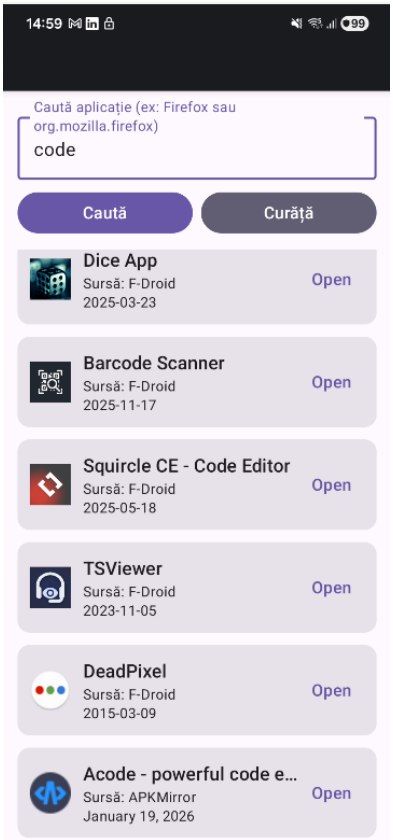
Aplicația **AppTracker** este un sistem de tip **web scraper și data aggregator** dezvoltat pentru platforma Android, care colectează și centralizează informații despre aplicații mobile Android din surse publice online. Aplicația are ca scop oferirea unei interfețe unificate pentru căutarea și analizarea aplicațiilor disponibile pe mai multe platforme de distribuție.

Sursele de date utilizate sunt:

- **F-Droid**, prin consumarea API-ului public index-v2.json
- **APKMirror**, prin analiză și parsare a paginilor HTML

Scopul principal al aplicației este:

- să permită căutarea aplicațiilor după **nume** sau **nume de pachet**;
- să agrege rezultate din mai multe surse într-o singură listă;
- să ofere utilizatorului informații relevante precum: nume aplicație, pachet, versiune, dezvoltator, dată de lansare, icon și link de descărcare;
- să permită evaluarea aplicației pe un set de căutări predefinite, din perspectiva **calității datelor**, **robusteții mecanismului de colectare** și **timpului de răspuns**.



2. Context și surse de date

- lista completă de pachete (packages: Map<String, FdroidPackageEntry>);
- pentru fiecare pachet: metadate precum nume localizat, descriere, autor, licență, icon;
- lista versiunilor disponibile, cu informații despre versionName, versionCode și data adăugării (added).

În cadrul aplicației **AppTracker**, datele din F-Droid sunt obținute **fără scraping HTML**, folosind exclusiv API-ul JSON oficial, prin intermediul bibliotecilor **Retrofit** și **Moshi**.

Implementarea accesului la F-Droid se află în fișierul: FdroidApiService.kt

2.2. APKMirror

APKMirror este un catalog online de aplicații Android care **nu oferă un API public oficial** pentru acces programatic. Din acest motiv, aplicația AppTracker utilizează **scraping HTML** pentru a colecta datele necesare.

Aplicația construiește URL-uri de căutare de forma:

- https://www.apkmirror.com/?s=<query>&post_type=app

Pentru fiecare pagină de rezultate, sunt extrase următoarele informații:

- titlul aplicației;
- link-ul către pagina aplicației sau către pagina de release;
- icon-ul aplicației (dacă este disponibil);
- data ultimei actualizări afișată pe card.

Scraping-ul este implementat în fișierul:

- ApkMirrorScraper.kt

Biblioteca utilizată pentru parsarea HTML este **Jsoup**, iar pentru respectarea sursei sunt aplicate:

- un **User-Agent real de browser mobil**;
- un **crawl delay** între request-uri.

```
val doc = Jsoup.connect(appPageUrl)
    .userAgent( userAgent = USER_AGENT)
    .header( name = "Accept-Language", value = ACCEPT_LANG)
    .header( name = "Accept", value = "text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8")
    .timeout( millis = 20_000)
    .followRedirects( followRedirects = true)
    .get()
```

3. Cerințe funcționale

3.1. Căutare aplicații

Utilizatorul introduce un termen de căutare (nume aplicație sau nume de pachet) în interfața grafică. La apăsarea butonului „Caută”, aplicația interoghează simultan sursele F-Droid și APKMirror folosind termenul introdus.

Logica de căutare este declanșată din:

- SearchScreen.kt
- SearchViewModel.kt

```
fun search() {
    val q = _state.value.query.trim()
    if (q.isEmpty()) {
        _state.update { it.copy(error = "Introdu un termen de căutare.") }
        return
    }

    searchJob?.cancel()
    searchJob = viewModelScope.launch {
        _state.update { it.copy(loading = true, error = null, selected = null, selectedLoading = false) }

        runCatching { repo.searchLite( term = q, limit = 20) }
            .onSuccess { list ->
                _state.update { it.copy(loading = false, results = list, error = null) }
            }
            .onFailure { e ->
                _state.update { it.copy(loading = false, error = e.message ?: "Eroare necunoscută") }
            }
    }
}
```

3.2. Integrare F-Droid

Pentru F-Droid, aplicația:

1. descarcă fișierul index-v2.json;
2. parcurge structura packages;
3. identifică pachetele care conțin termenul de căutare în:
    - numele pachetului sau
    - numele aplicației;
4. extrage pentru fiecare rezultat:
    - packageName
    - appName
    - versionName, versionCode
    - releaseDate
    - developer
    - iconUrl
    - downloadUrl

Această logică este implementată în:

- AppRepositoryImpl.kt, metoda searchLite()

```
fdroidResults += AppInfo(
    source = "F-Droid",
    packageName = pkg,
    appName = appName,
    versionName = versionName ?: latestFromIndex?.versionName,
    versionCode = versionCode ?: latestFromIndex?.versionCode?.toInt(),
    releaseDate = releaseDate,
    developer = meta.authorName,
    downloadUrl = "https://f-droid.org/en/packages/$pkg/",
    iconUrl = iconUrl,

    // ✅ nice extras already in index-v2
    summary = pickLocalized( map = meta.summary),
    license = meta.license,
    sourceCodeUrl = meta.sourceCode
)
```

3.3. Integrare APKMirror

Pentru APKMirror, aplicația:

1. construiește URL-ul de căutare pe baza termenului introdus;
2. descarcă pagina HTML folosind Jsoup;

3. selectează zona principală de conținut (div#content);
4. extrage cardurile de aplicații (div.appRow);
5. pentru fiecare card, extrage:
    - numele aplicației;
    - link-ul către pagina APK-ului;
    - icon-ul aplicației.

Datele sunt mapate într-un obiect AppInfo, cu source = "APKMirror".

```
val mainContent = doc.selectFirst( cssQuery = "div#content") ?: doc

val rows = mainContent.select( cssQuery = "div[class*=appRow]")

val apps = mutableListOf<AppInfo>()

for (row in rows) {

    val titleEl =
        row.selectFirst( cssQuery = ".appRowTitle")
            ?: row.selectFirst( cssQuery = "h5, h4, h3")
            ?: row.selectFirst( cssQuery = "a")

    val titleText = titleEl?.text()?.trim().orEmpty()
    if (titleText.isBlank()) continue
```

```
apps += AppInfo(
    source = "APKMirror",
    packageName = "",
    appName = appName,
    versionName = versionName,
    developer = developer,
    downloadUrl = pageUrl,
    iconUrl = iconUrl,
    lastUpdated = lastUpdated
)
```

3.4. Agregare rezultate

Rezultatele provenite din F-Droid și APKMirror sunt:

- combinate într-o singură listă;
- deduplicate pe baza (source, packageName) sau (source, downloadUrl).

Aplicația aplică o regulă de afișare:

- **10 rezultate din F-Droid**
- **10 rezultate din APKMirror**

Agregarea este realizată în:

- AppRepositoryImpl.kt

```
val targetFdroid = minOf( a = 10,  b = limit)
val targetMirror = minOf( a = 10,  b = maxOf( a = 0,  b = limit - targetFdroid))

val fetchFdroid = targetFdroid * 3
val fetchMirror = targetMirror * 3
```

3.5. Afișare rezultate

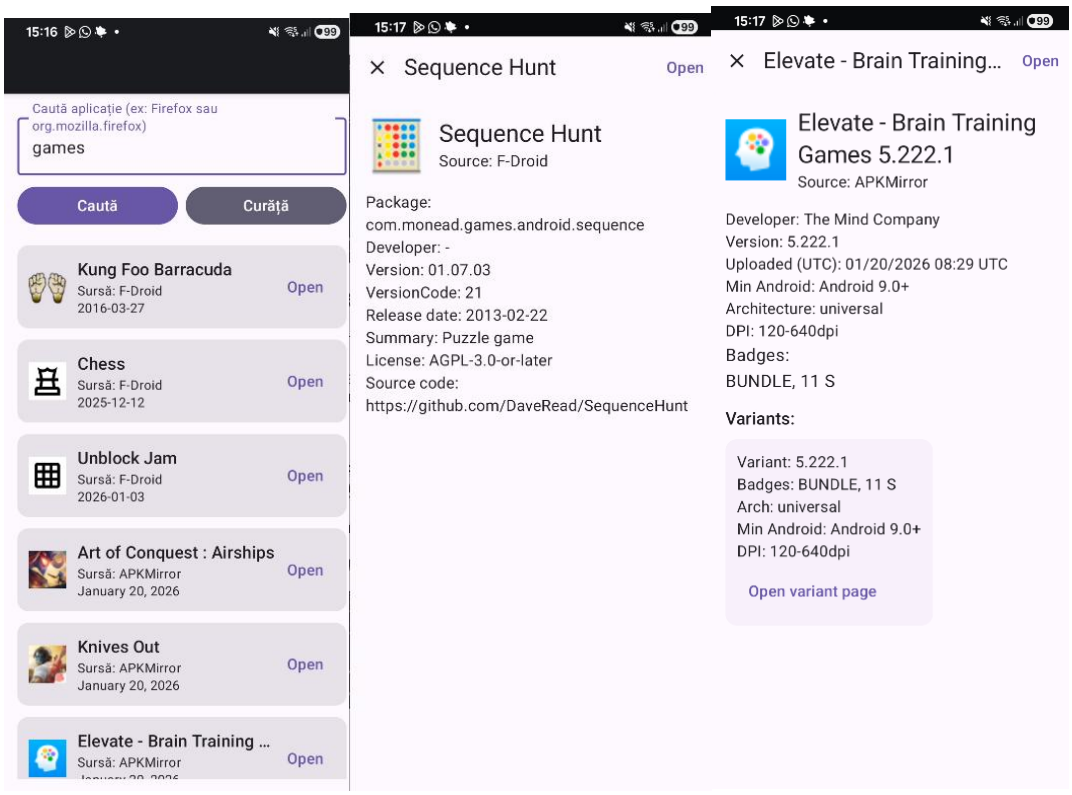
Rezultatele sunt afișate într-o listă scrollabilă (LazyColumn) folosind Jetpack Compose. Fiecare element conține:

- nume aplicație;
- sursa (F-Droid / APKMirror);
- versiune sau dată;
- icon;
- buton de deschidere în browser.

La selectarea unui element, se deschide un **dialog fullscreen** cu detalii suplimentare.

Componente UI:

- AppCard.kt
- FullScreenDetailsDialog



4. Cerințe nefuncționale

4.1. Performanță

Operațiile de rețea și parsare sunt executate pe `Dispatchers.IO`, pentru a evita blocarea interfeței grafice. Timpul de răspuns este influențat de latența rețelei și dimensiunea fișierului `index-v2.json`.

```
override suspend fun searchLite(term: String, limit: Int): List<AppInfo> =
    withContext( context = Dispatchers.IO) {

        val query = term.trim()
        if (query.isEmpty()) return@withContext emptyList()

        val lower = query.lowercase()

        val targetFdroid = minOf( a = 10,  b = limit)
        val targetMirror = minOf( a = 10,  b = maxOf( a = 0,  b = limit - targetFdroid))

        val fetchFdroid = targetFdroid * 3
        val fetchMirror = targetMirror * 3

        val fdroidResults = mutableListOf<AppInfo>()
        val mirrorResults = mutableListOf<AppInfo>()
```

4.2. Fiabilitate

Erorile de rețea sau de parsing sunt tratate folosind `runCatching`, fără a duce la crash-ul aplicației. În cazul în care o sursă nu este disponibilă, aplicația continuă să afișeze rezultatele din cealaltă sursă.

```
runCatching {
    val app = fdroid.getApp( packageId = pkg)
    val suggested = app.suggestedVersionCode
    val chosen = app.packages.firstOrNull { it.versionCode != null && it.versionCode == suggested }
        ?: app.packages.filter { it.versionCode != null }.maxByOrNull { it.versionCode!! }

    chosenApiVersionCode = chosen?.versionCode
    versionName = chosen?.versionName
    versionCode = chosen?.versionCode?.toInt()
}.onFailure {
    versionName = latestFromIndex?.versionName
    versionCode = latestFromIndex?.versionCode?.toInt()
    Log.w( tag = "FDROID",  msg = "getApp($pkg) failed, fallback to index-v2")
}
```

4.3. Logare și debugging

Aplicația utilizează:

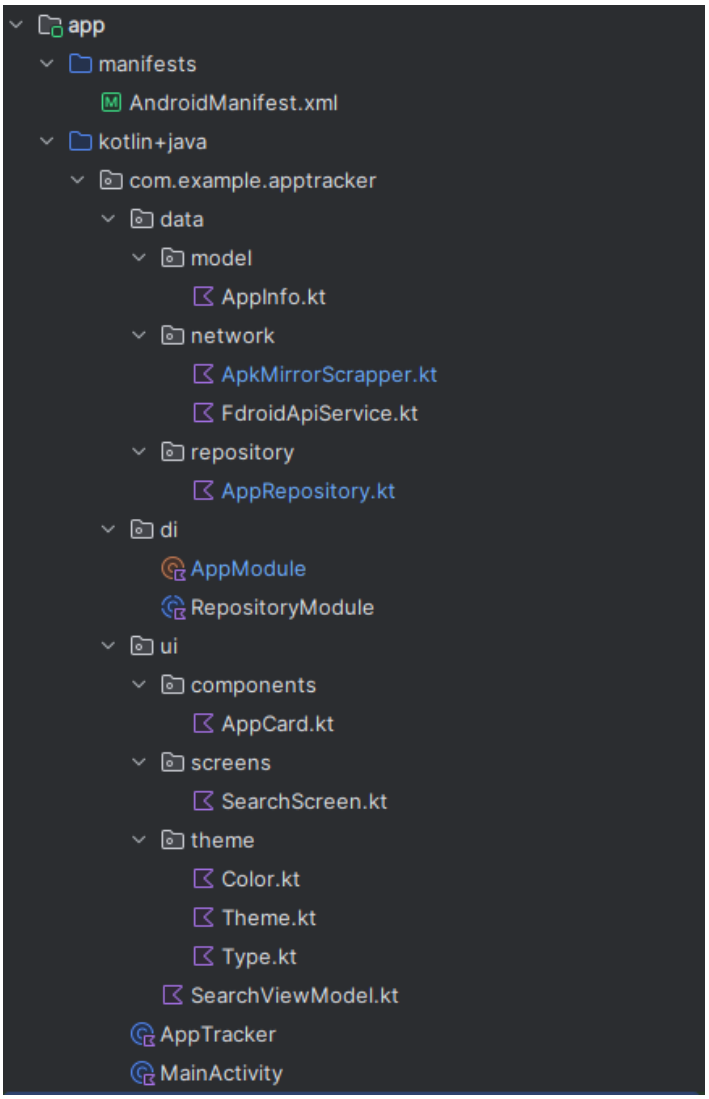
- `Log.d` și `Log.e` pentru urmărirea execuției;
- `HttpLoggingInterceptor` pentru debugging HTTP în build-ul de debug.

```
@Provides
@Singleton
fun provideOkHttpClient(): OkHttpClient {
    val logging = HttpLoggingInterceptor().apply {
        level = HttpLoggingInterceptor.Level.BODY // Logs all HTTP requests/responses
    }
    return OkHttpClient.Builder()
        .addInterceptor( interceptor = logging)
        .retryOnConnectionFailure( retryOnConnectionFailure = true)
        .build()
}
```

5. Arhitectura sistemului

Aplicația este structurată pe layere:

1. **UI Layer – Jetpack Compose**
    - SearchScreen
    - SearchViewModel
2. **Repository Layer**
    - AppRepository
    - AppRepositoryImpl
3. **Data / Network Layer**
    - FdroidApiService
    - ApkMirrorScraper
    - modele de date F-Droid și AppInfo
4. **Dependency Injection Layer (Hilt)**
    - AppModule
    - RepositoryModule
    - AppTracker



6. Modelul de date

6.1. Model intern comun – AppInfo

Pentru a permite agregarea datelor provenite din surse diferite (F-Droid și APKMirror), aplicația utilizează un **model intern comun**, denumit `AppInfo`. Acest model asigură o reprezentare unitară a aplicațiilor, indiferent de sursa de proveniență.

Modelul este definit în fișierul:

- `AppInfo.kt`

Rolul principalelor câmpuri:

- `source` – indică sursa aplicației („F-Droid” sau „APKMirror”);
- `packageName` – identificatorul unic al aplicației (unde este disponibil);
- `appName` – numele afișat utilizatorului;
- `versionName` / `versionCode` – informații despre versiune;
- `releaseDate` – data publicării sau actualizării;
- `developer` – autorul aplicației;
- `downloadUrl` – link către sursa originală;
- `iconUrl` – URL-ul icon-ului aplicației.

```
data class ApkMirrorVariant(
    val variantName: String? = null,
    val badges: List<String> = emptyList(),
    val architecture: String? = null,
    val minAndroid: String? = null,
    val dpi: String? = null,
    val variantPageUrl: String? = null
)

29 Usages
data class AppInfo(
    val source: String,
    val packageName: String,
    val appName: String,
    val versionName: String? = null,
    val versionCode: Int? = null,

    // used for all sources
    val releaseDate: String? = null,

    val developer: String? = null,
    val downloadUrl: String? = null,
    val iconUrl: String? = null,

    // APKMirror list
    val lastUpdated: String? = null,
```

```
// APKMirror details (release page)
val fileSize: String? = null,
val downloads: String? = null,
val uploadedUtc: String? = null,
val minAndroid: String? = null,
val architecture: String? = null,
val dpi: String? = null,
val badges: List<String> = emptyList(),
val variants: List<ApkMirrorVariant> = emptyList

// F-Droid nice metadata
val summary: String? = null,
val license: String? = null,
val sourceCodeUrl: String? = null,

val minSdk: Int? = null,
val targetSdk: Int? = null,
val categories: List<String>? = null,
val antiFeatures: List<String>? = null,
val permissions: List<String>? = null
```

6.2. Modele de date pentru F-Droid

Pentru maparea structurii JSON furnizate de F-Droid, aplicația definește mai multe modele de date, toate localizate în layer-ul `data/network`.

Principalele clase sunt:

- `FdroidIndex` – obiectul root al fișierului `index-v2.json`;
- `FdroidPackageEntry` – conține metadatele și lista versiunilor;

- `FdroidMetadata` – metadate precum nume, descriere, autor, licență;
- `FdroidVersion` – informații despre versiuni (versionName, versionCode, added).

Aceste modele permit parsarea automată a JSON-ului folosind Moshi.

7. Mecanismul de colectare pe surse

Aplicația AppTracker colectează date din surse diferite folosind mecanisme adaptate fiecărei platforme.

7.1. Colectarea datelor din F-Droid

Datele sunt obținute prin consumarea API-ului public `index-v2.json`. Informațiile sunt mapate în modele interne, filtrate după termenul de căutare și transformate în obiecte `AppInfo`. Pentru optimizarea performanței, index-ul este cache-uit local.

7.2. Colectarea datelor din APKMirror

În lipsa unui API public, datele sunt colectate prin scraping HTML folosind biblioteca Jsoup. Aplicația descarcă pagina de rezultate, extrage cardurile de aplicații și mapează informațiile relevante în modelul comun `AppInfo`, aplicând mecanisme de deduplicare.

8. Set de căutări predefinite

Pentru evaluarea funcționalității aplicației AppTracker a fost utilizat un set de termeni de căutare predefiniți, reprezentativi pentru aplicații populare din ecosistemul Android:

- telegram
- signal
- firefox
- newpipe
- vlc
- aurora store

Pentru fiecare termen au fost analizate:

- numărul de rezultate obținute din F-Droid și APKMirror;
- completitudinea datelor afișate (versiune, dezvoltator, icon);
- eventuale erori apărute în procesul de colectare.

9. Evaluarea aplicației

Evaluarea aplicației s-a realizat pe baza următoarelor criterii:

Corectitudinea rezultatelor

Informațiile afișate în aplicație corespund cu cele disponibile în sursele originale (F-Droid și APKMirror).

Completența datelor

Majoritatea aplicațiilor includ câmpurile esențiale: nume, versiune, dezvoltator și icon.

Robustețea la erori

Aplicația continuă să funcționeze chiar dacă una dintre surse nu este disponibilă.

Timpul de răspuns

Rezultatele sunt afișate într-un timp rezonabil, dependent de latența rețelei și dimensiunea datelor procesate.