Data Layer II

Room Database, Retrofit (HTTP)

prof. dr Bratislav Predić
dipl. inž. Nevena Tufegdžić

Razvoj mobilnih aplikacija i servisa
Računarstvo i informatika

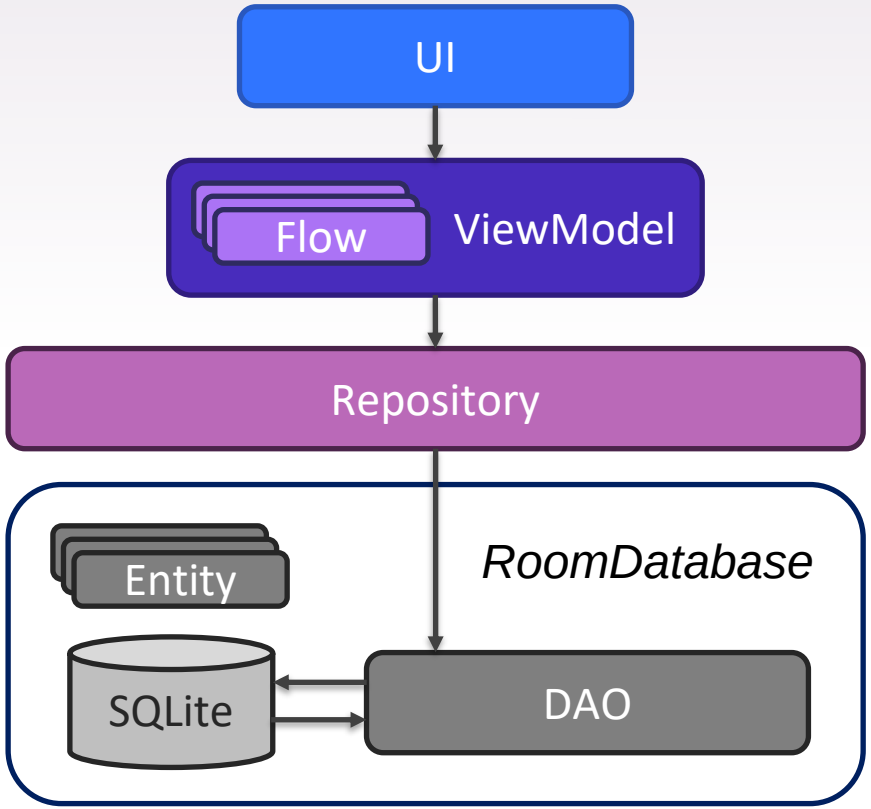
2024.

Room Database

prof. dr Bratislav Predić
dipl. inž. Nevena Tufegdžić

Data Layer II
Razvoj mobilnih aplikacija i servisa

Room Database arhitektura



- Room je apstrakcija nad SQLite bazom
- Koristi Data Access Objects (DAO) kako bi pristupala bazi
- Bazu modeluju Entity klase
- Aplikacija pristupa bazi pomoću Repository klase
- ViewModel sadrži Flows za podatke iz baze

Potrebni dependencies

- Build.gradle (Module: app):

```
plugins {
    alias(libs.plugins.androidApplication)
    alias(libs.plugins.jetbrainsKotlinAndroid)
    id ("kotlin-kapt")
}
```

Potrebni dependencies

- Build.gradle (Module: app):

```
implementation("androidx.room:room-ktx:2.6.1")
kapt("androidx.room:room-compiler:2.6.1")
androidTestImplementation("androidx.room:room-testing:2.6.1")
```

Entity

- Predstavlja objekat koji treba perzistirati u bazi
- Svaki entitet odgovara jednoj tabeli u bazi
  - Svaka instanca entiteta odgovara jednom redu tabele (torka)
  - Property entiteta predstavlja kolonu (atribut) u tabeli u bazi
- Entiteti služe za definisanje šeme baze podataka
- Koristi se anotacija *@Entity*

Entity - primer

```
@Entity
data class User(
    @PrimaryKey val id: Int,

    val firstName: String?,
    val lastName: String?
)
```

Entity

- Room podrazumeva ime klase kao ime tabele, ukoliko mu se ne kaže drugačije
- Na isti način, podrazumeva se ime property-ja kao ime atributa, osim ako se drugačije ne navede
- Za primarni ključ koristi se anotacija *@PrimaryKey*
  - Može se podesiti da baza automatski generiše vrednost primarnog ključa
- Imena tabela i atributa u SQLite su case sensitive

Entity - primer

```
@Entity(tableName = "users")
data class User (
    @PrimaryKey(autoGenerate = true) val id: Int,
    @ColumnInfo(name = "first_name") val firstName: String?,
    @ColumnInfo(name = "last_name") val lastName: String?
)
```

Data Access Object

```
@Dao
interface UserDao {
    @Insert
    fun insertAll(vararg users: User)

    @Delete
    fun delete(user: User)

    @Query("SELECT * FROM user")
    fun getAll(): List<User>
}
```

- DAO je interfejs pomoću koga se pristupa bazi – sadrži definicije metoda za vršenje upita nad bazom
- Anotacija *@Dao*

Pisanje DAO metoda

- Svaka metoda anotirana je jednim od dve vrste anotatora:
  - *@Query* anotator – služi za pisanje generalnih upita ka bazi
  - Specijalni anotatori – *@Insert, @Update, @Delete*
- @Query anotator zahteva parametar – to je SQL upit

```
@Query("SELECT * FROM user")
fun loadAllUsers(): Array<User>
```

Pisanje DAO metoda

- Insert:

```
@Insert(onConflict = OnConflictStrategy.IGNORE)
suspend fun insert(word: Word)
```

- Update:

```
@Update
suspend fun updateUser(user: User)
```

- Delete:

```
@Delete
suspend fun deleteUser(user: User)
```

Definisanje relacija

- Room ne podržava čuvanje referenci na druge objekte
- Relacije smeju da se definišu na dva načina:
  - Korišćenjem tabele relacije
    - Kreira se data klasa koja predstavlja entitet veze – čuva ključeve od obe strane veze
  - Multimap query
    - DAO metoda koja vraća mapu sa jednim entitetom (nosilac veze) i listom ostalih entiteta (zavisna strana veze)

Tabela relacije

```
@Dao
interface UserBookDao {
    @Query(
        "SELECT user.name AS userName, book.name AS bookName " +
        "FROM user, book " +
        "WHERE user.id = book.user_id"
    )
    fun loadUserAndBookNames(): LiveData<List<UserBook>>
}

data class UserBook(val userName: String?, val bookName: String?)
```

Multimap

```
@Query(
    "SELECT * FROM user" +
    "JOIN book ON user.id = book.user_id"
)
fun loadUserAndBookNames(): Map<User, List<Book>>
```

Database klasa

```
@Database(entities = arrayOf(User::class), version = 1, exportSchema = false)
abstract class UserDatabase : RoomDatabase(){
    abstract fun userDao() : userDao

    companion object {
        @Volatile
        private var INSTANCE: UserDatabase? = null

        fun getDatabase(context: Context, scope: CoroutineScope): UserDatabase {
            ...
```

Database klasa

```
    ...
fun getDatabase(context: Context, scope: CoroutineScope): UserDatabase {
    return INSTANCE ?: synchronized(this) {
        val instance = Room.databaseBuilder(
            context.applicationContext,
            UserDatabase::class.java,
            "user_database"
        ).build()
        INSTANCE = instance
        instance
    }
}
```

Pristup HTTP servisima - Retrofit

prof. dr Bratislav Predić
dipl. inž. Nevena Tufegdžić

Data Layer II
Razvoj mobilnih aplikacija i servisa

Retrofit

- Biblioteka za komunikaciju sa REST servisima
- Omogućava pozive REST funkcija sa nekog web servera i obradu response objekata
- Podržava rad sa RequestBody i ResponseBody objektima
- REST servis se modeluje pomoću interfejsa u Kotlinu, čije se funkcije mogu pozivati u aplikaciji

Uključivanje Retrofit-a u projekat

- Biblioteka se uključuje dodavanjem u Gradle build fajl na nivou modula – *build.gradle(Module:app)*

```
implementation "com.squareup.retrofit2:retrofit:2.9.0"
```

- Proveriti da li build.gradle na nivou projekta ima sledeće:

```
repositories {
    google()
    mavenCentral()
}
```

Response klasa

- Templejtska klasa koja sadrži informacije o HTTP odgovoru

Response<T>	
code()	Integer status code
body()	Telo response-a ukoliko je zahtev uspešno prošao
message()	Poruka u okviru response-a

... neke od metoda klase

Konverzija odgovora servera

- Retrofit zahteva JSON konverter kako bi mogao da vrši konverziju odgovora servera u objekte u Kotlinu, i obrnuto
- Neki JSON konverteri:
  - Scalar converter
  - Gson
  - **Moshi**
  - Protobuf
  - Jackson

Konverzija odgovora servera

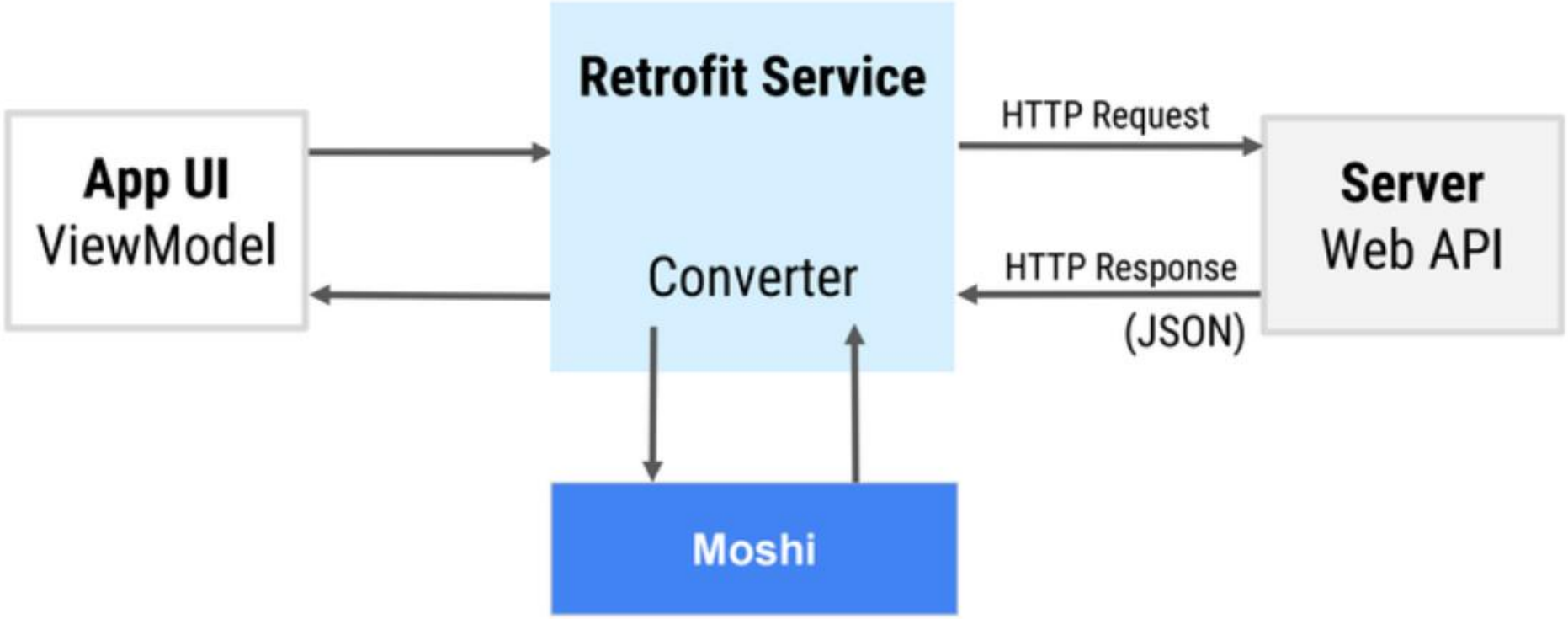
- Konverter je potrebno uključiti u projekat u *build.gradle* na nivou modula:

```
implementation 'com.squareup.moshi:moshi-kotlin:1.13.0’
implementation 'com.squareup.retrofit2:converter-moshi:2.9.0'
```

- ... umesto:

```
implementation "com.squareup.retrofit2:retrofit:2.9.0"
```

Funkcionisanje sistema



Povezivanje sa HTTP serverom

- Retrofit modeluje HTTP server preko interfejsa
- Potrebno je definisati interfejs sa svim HTTP metodama koje se mogu pozvati sa servera

Metod	Endpoint
GET	/students
POST	/addStudent

```
interface StudentApiService {
    @GET("students")
    suspend fun getStudents() : Array<Student>

    @Headers("Content-Type: application/json")
    @POST("addStudent")
    suspend fun addStudent(@Body student: Student)
        : Response<Student>
}
```

GET

- Za definisanje GET funkcije koristi se anotacija ***@GET***
- Anotacija prima putanju ka endpoint-u koji izvršava funkciju koja se modeluje u interfejsu
- Funkcija treba da vraća ono što GET metoda vraća sa servera

```
@GET("students")
suspend fun getStudents() : Array<Student>
```

POST

- Za definisanje POST funkcije koristi se anotacija ***@POST***, koja takođe prima putanju ka endpoint-u pomoću kog se pristupa funkciji na serveru

```
@Headers("Content-Type: application/json")
@POST("addStudent")
suspend fun addStudent(@Body student: Student) :
    Response<Student>
```

POST

- Anotacija ***@Headers*** služi za definisanje hedera zahteva
- U ovom slučaju to je u svrhu definisanja tipa sadržaja
- Anotacija ***@Body*** služi za definisanje da se parametar prosleđuje kao body u zahtevu

```
@Headers("Content-Type: application/json")
@POST("addStudent")
suspend fun addStudent(@Body student: Student) :
    Response<Student>
```

Retrofit servis

Pristup localhost-u iz emulatora

```
private const val BASE_URL = "http://10.0.2.2:3000/"

private val moshi = Moshi.Builder()
    .add(KotlinJsonAdapterFactory())
    .build()

private val retrofit = Retrofit.Builder()
    .addConverterFactory(MoshiConverterFactory.create(moshi))
    .baseUrl(BASE_URL)
    .build()
```

Retrofit servis

```
object StudentApi {
    val retrofitService: StudentApiService by lazy {
        retrofit.create(StudentApiService::class.java)
    }
}
```

- Koristi se Singleton pattern, jer je na nivou aplikacije dovoljna samo jedna konekcija sa serverom
- Definisana retrofit konstanta se koristi za kreiranje retrofitService objekta

Korišćenje Retrofit servisa

```
private fun getStudents() {
    viewModelScope.launch {
        try {
            val listResult = StudentApi.retrofitService.getStudents()
            _status.value =
                  "GET results: fetched ${listResult.size} students."
        } catch (e: Exception) {
            _status.value = "Failure: ${e.message}"
        }
    }
}
```

Korišćenje Retrofit servisa

```
private fun postStudent(student: Student) {
    viewModelScope.launch {
        try {
            val result = StudentApi.retrofitService.addStudent(student)
            _status.value = result.body()?.name
        } catch (e: Exception) {
            _status.value = "Failure: ${e.message}"
        }
    }
}
```

Definisanje klase za serijalizaciju

- Kako bi se vršila serijalizacija i deserijalizacija Kotlin objekata u JSON, potrebno je definisati data klasu za to
- Klasa sadrži sve atribute koji se očekuju u JSON objektu
- Ukoliko JSON objekat treba da nazove atribut na drugačiji način koristi se anotacija ***@Json(name = „ime_atributa“)***

```
data class Student(
    @Json(name = "id") val id: String,
    @Json(name = "name") val name: String
    )
```

Problem

- Nedostaju permisije za korišćenje Interneta u aplikaciji!
- Pre nego što aplikacija može da komunicira sa serverom, treba osigurati da se u *AndroidManifest.xml* fajlu nalazi deklaracija permisije:

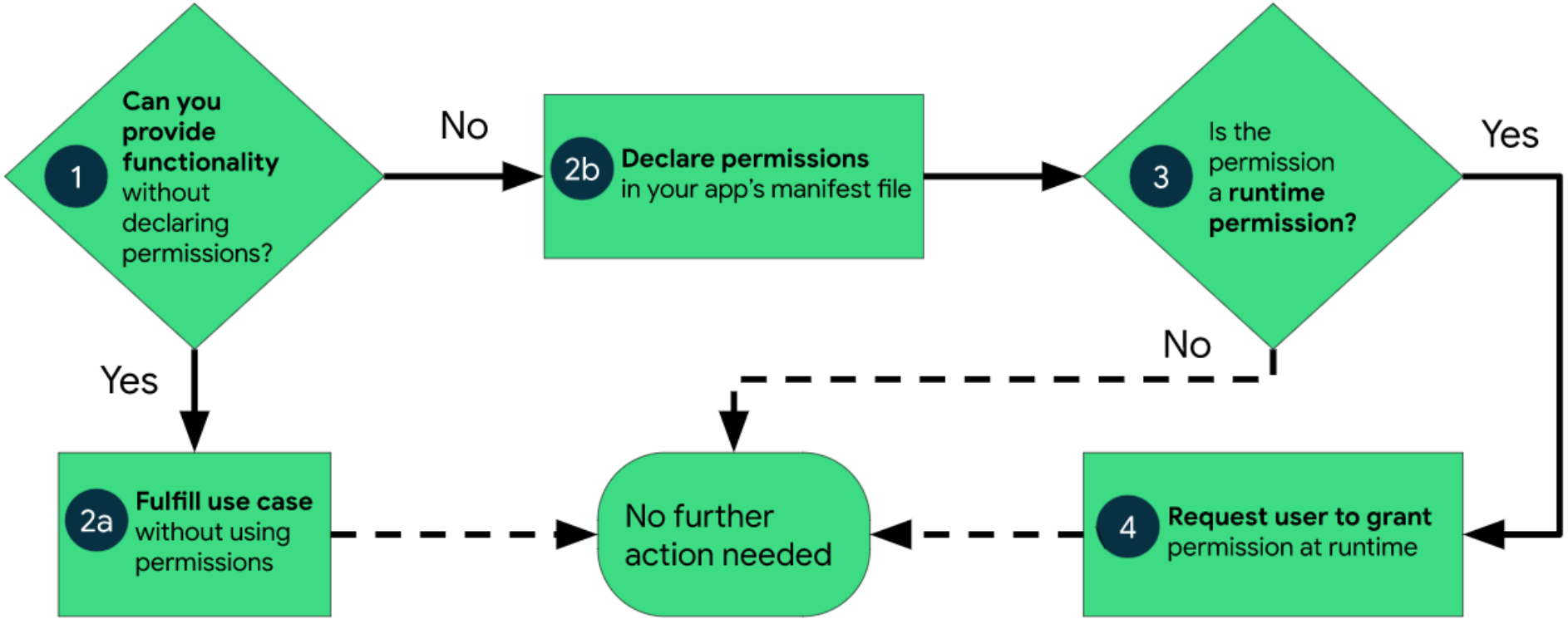
```
<manifest
xmlns:android="http://schemas.android.com/apk/res/android">

    <uses-permission android:name="android.permission.INTERNET" />

    <application
   ...
```

Android permisije

Protection Level	Granted when?	Must prompt before use?	Examples
Normal	Install time	No	`ACCESS_WIFI_STATE`, `BLUETOOTH`, `VIBRATE`, `INTERNET`
Signature	Install time	No	N/A
Dangerous	Runtime	Yes	`GET_ACCOUNTS`, `CAMERA`, `CALL_PHONE`

Puna lista: https://developer.android.com/reference/android/Manifest.permission

Pravila za permisije



Pravila za permisije

- Ukoliko su permisije zahtevane tokom izvršenja aplikacije, treba pitati korisnika za permisiju
- Ako korisnik inicijalno odbije, treba prikazati prompt koji ga obaveštava zbog čega su potrebne
- Ako korisnik ponovo ne dozvoli permisiju, aplikacija ne sme da pukne
- Obezbediti da aplikacija i dalje radi, sa limitiranom funkcionalnošću

Literatura

- Room with Jetpack Compose
- Room Database - Android Developer
- Room Database Codelab
- Retrofit Documentation
- Retrofit Codelab
- Glide Documentation

Hvala na pažnji!