Predavanje br. 11

Sadržaj

Couroutines

Umrežavanje

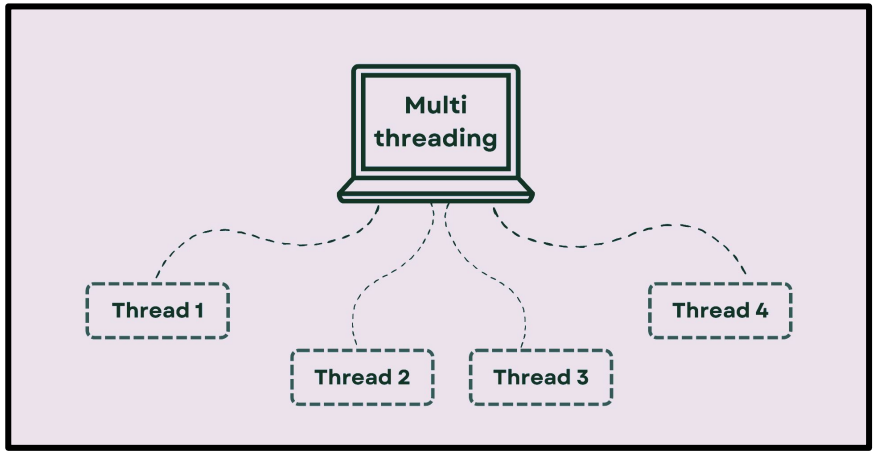
Učitavanje slika sa Internet-a

Coroutines

Multithreading (Idea)

Multithreading je proces istovremenog izvršavanja više sekvenci koda.
(npr. thread korisničkog interfejsa, thread networking, thread pristupa lokalnim podacima itd.)

Ovo je važno za networking jer je potrebno vrijeme za učitavanje stvari s interneta!



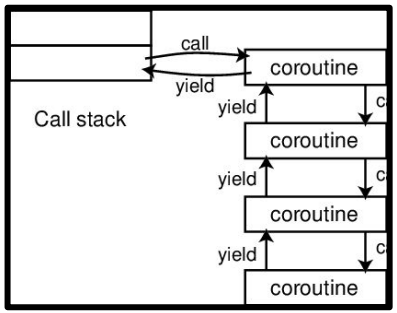
Ne želimo da se naša aplikacija samo zamrzne kada se nešto učitava, pa je pokrećemo u drugom thread-u!

Asinhronizacija u Kotlinu

Kotlin ima vrlo jedinstven način upravljanja više thread-ova / asinhronim kodom.

(Vrlo različit od Jave, Pythona, itd.)

To su **Coroutines**



Asinhronizacija u Kotlinu

Na vašu sreću, nije potrebno da znate kako u pozadini oni rade, jer su vrlo jednostavne za koristiti!!!

Ideja:
Kad god vam je potrebno da nešto radi asinhrono, samo to pokrenite na korutini, a Kotlin će se pobrinuti za SVE ostalo.

Analogija

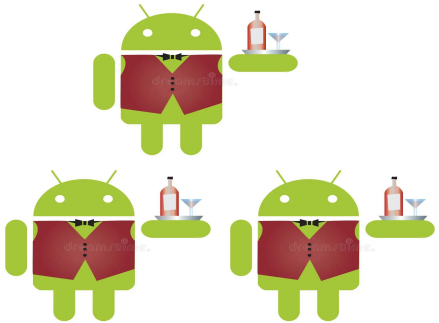
Coroutine (sto) “napravi narudžbu” i **thread** (konobar) odlazi u kuhinju da prenese narudžbu.

Konobar **ne čeka** u kuhinji da hrana bude gotova; on poslužuje ostale stolove i uzima više narudžbi.

Kada je hrana spremna, bilo koji slobodan konobar je može donijeti za pravi sto.

Taskovi u pozadini

Thread-ovi



Couroutine

suspend fun

Svaka funkcija koja se može pauzirati (npr. mrežni poziv) mora biti označena sa **`suspend`**

Funkcija suspenzije ne blokira ⟶ ona predaje kontrolu dok se posao ne završi

Suspend funkcije se mogu pozivati samo unutar **coroutines** ili drugih suspend funkcija

```
/** Given an [id], returns the image associated with it. */
suspend fun getImageFromId(id: String): ImageBitmap {
    delay( timeMillis: 500)
    return imageMap[id] ?: ImageBitmap( width: 1,  height: 1)
}
```

Primjer suspend funkcije

suspend fun

Kada deklarišete u Kotlinu

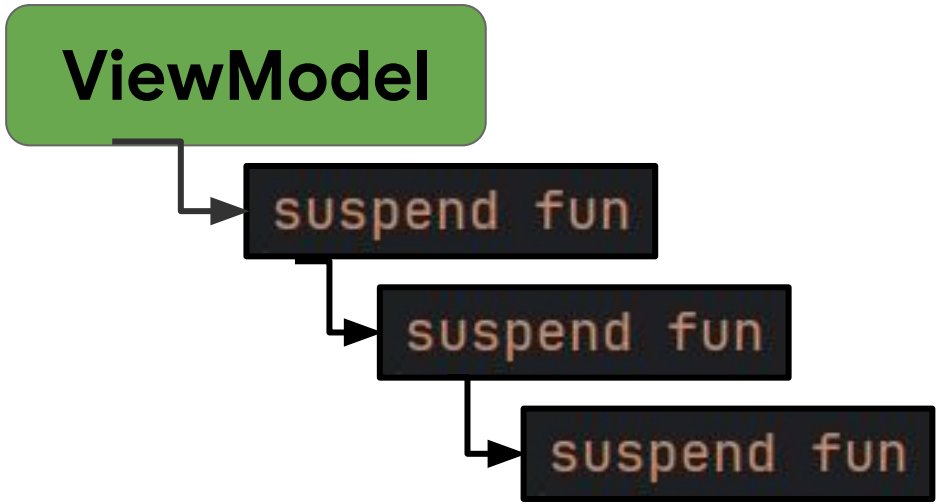
```
suspend fun
```

To znači da kažete

“Znam da ova funkcija ima asinhronog koda, ne brini!”

suspend fun

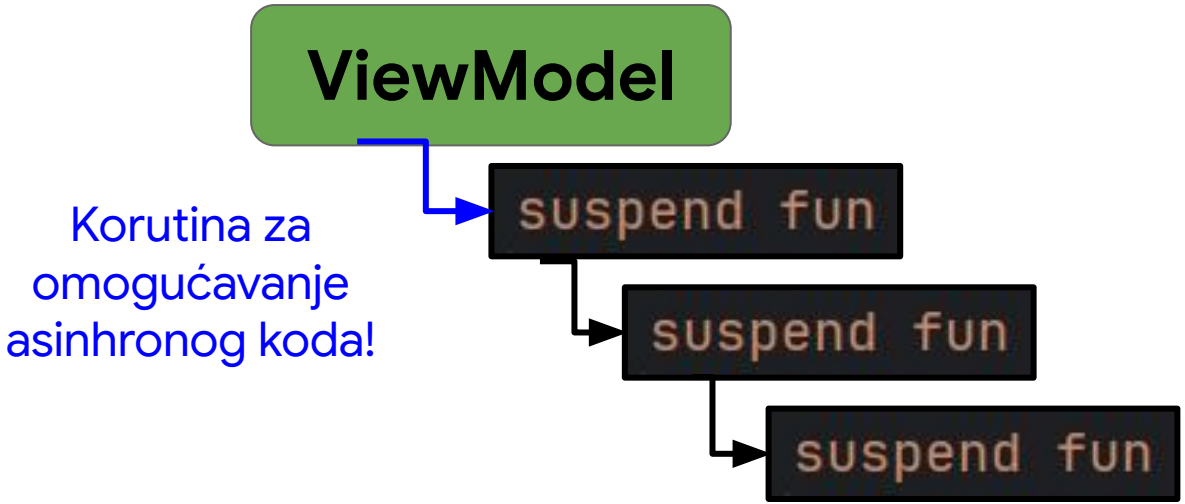
Ideja je, međutim, da ćete u nekom trenutku morati pokrenuti suspend fun na svom redovnom (neblokirajućem) kodu:



suspend fun

Ovdje ćete pokrenuti **Coroutine**!

Izvršavanje tijela korutine će trajati neko vrijeme i radit će u pozadini.



Pokretanje Coroutines

Postoji nekoliko načina da se pokrenu:

1. U ViewModel: `viewModelScope`

```
viewModelScope.launch {
    delay( timeMillis: 50)
    refreshImages()
}
```

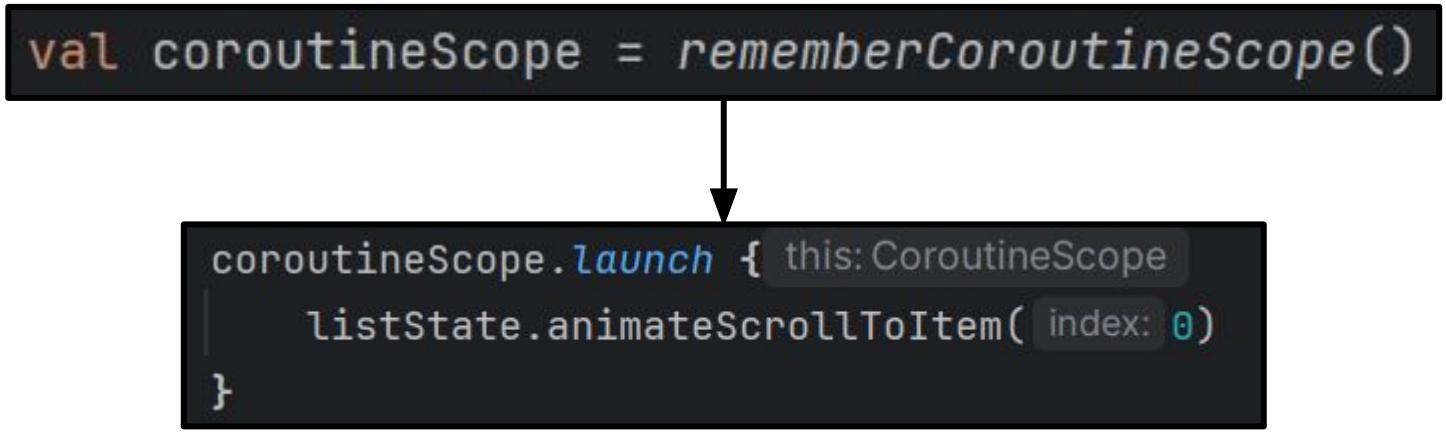
Ovaj kod čeka 50 ms, pa onda pokreće funkciju refreshImages()

2. Bilo gdje drugo: `CoroutineScope(Dispatchers.IO)`

```
CoroutineScope(Dispatchers.IO).launch {
    saveImage(bitmap.asImageBitmap())
}
```

Pokretanje Coroutines

3. U UI: `rememberCoroutineScope()`



Pokretanje Coroutines

U bilo kojem slučaju:
Kada imate definiran coroutine scope, samo pozovite `.launch{}`, i kod unutar `{}` će raditi asinhrono.

```
viewModelScope.launch {
    delay( timeMillis: 50)
    refreshImages()
}
```

```
coroutineScope.launch { this: CoroutineScope
    listState.animateScrollToItem( index: 0)
}
```

```
CoroutineScope(Dispatchers.IO).launch {
    saveImage(bitmap.asImageBitmap())
}
```

Retrofit & JSON

Backend odgovori

Backend platforme obično funkcionišu tako što omogućavaju više **endpoint-a.**

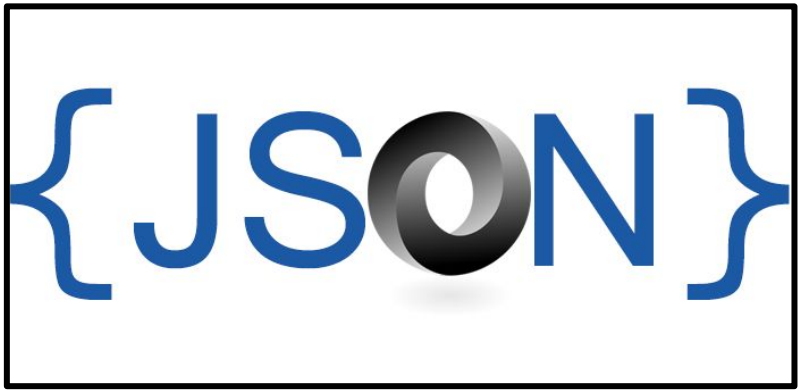
(svaki endpoint odgovara jednom URL kojem vaša aplikacija može pristupiti.)

Backend odgovori

Kada god pošaljete zahtjev prema endpoint-u, vaša aplikacija će čekati odgovor od backend servera.

Ovaj odgovor obično je u formatu **JSON**.

(JSON je u osnovi jedno veliko strukturirano stablo podataka)



JSON primjer

Cats API

The Cats API provides detailed, qualitative information on every recognized cat breed.

API Ninjas

Kada pozovete ovaj endpoint

`https://api.api-ninjas.com/v1/cats`

Dobijate nazad

JSON


```
[
  {
    "length": "12 to 16 inches",
    "origin": "Southeast Asia",
    "image_link": "https://api-ninjas.com/images/
    "family_friendly": 3,
    "shedding": 3,
    "general_health": 2,
    "playfulness": 5,
    "children_friendly": 5,
    "grooming": 3,
    "intelligence": 5,
    "other_pets_friendly": 5,
    "min_weight": 6,
    "max_weight": 10,
    "min_life_expectancy": 9,
    "max_life_expectancy": 15,
    "name": "Abyssinian"
  }
]
```

Retrofit

Pisanje vlastitog Kotlin/Java koda za obradu odgovora i parsiranje svih tih podataka je nepraktično.

Zato koristimo biblioteku koja to radi umjesto nas:

Retrofit!

Sa Retrofit, možemo slati zahtjeve prema **endpoint-ima** i konvertovati odgovore u naše vlastite klase sa minimalnim trudom.

Retrofit

Kako je Retrofit 3rd party library, moramo dodati je u dependencies...

```
plugins {
        …
        kotlin("plugin.serialization") version "2.2.21"
        …
}

…

dependencies {
        …
        implementation("com.squareup.retrofit2:retrofit:3.0.0")
        implementation("com.squareup.okhttp3:okhttp:5.1.0")
        implementation("com.squareup.retrofit2:converter-kotlinx-serialization:3.0.0")
        implementation("org.jetbrains.kotlinx:kotlinx-serialization-json:1.9.0")
        …
}
```

Slojevi Retrofit-a

Retrofit radi u nekoliko slojeva kako bi definirao, konvertirao i pogodio krajnje tačke.

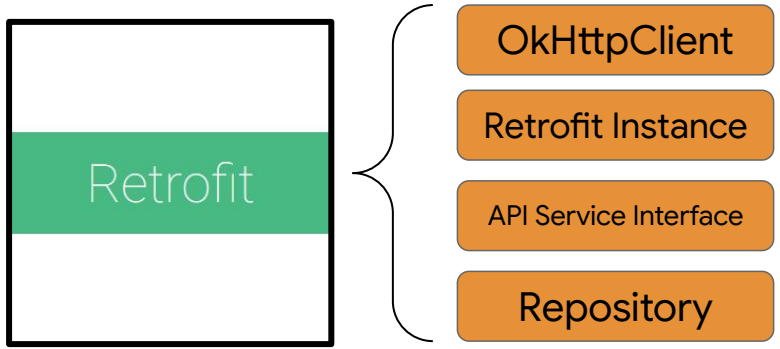
Svi ovi slojevi postoje unutar sloja modela, ali na samom vrhu.

(tj. druge stvari u sloju modela će zapravo koristiti ove slojeve za upućivanje mrežnih poziva!)

Slojevi Retrofit-a

Kako bi koristili Retrofit:

1. Kreirajte instancu OkHttpClient za rukovanje stvarnim mrežnim operacijama
2. Kreirajte instancu Retrofit za povezivanje HTTP klijenta i vašeg definiranog API interfejsa
3. Definirajte API strukturu kao interfejs i kreirajte instancu servisa
4. Ubacite servis u repozitorij (sljedeći put)



Sloj 1: OkHttpClient

Kreirajte OkHttpClient koji je odgovoran za slanje HTTP zahtjeva preko mreže i primanje *sirovih* odgovora.

Također možete dodati presretače i autentifikatore za dodatne funkcionalnosti poput automatskog dodavanja tokena za autorizaciju u zaglavlja zahtjeva.

Sloj 2: Retrofit Instanca

Kreirajte instancu Retrofita koja će spojiti Http klijenta i API interfejs koristeći osnovni URL API krajnje tačke.

Converter factory nam omogućava da konvertujemo JSON-ove u Kotlin objekte.

```
1 Usage
@Provides
@Singleton
fun provideRetrofit(okHttpClient: OkHttpClient): Retrofit {
    val json = Json {
        ignoreUnknownKeys = true
        isLenient = true
        coerceInputValues = true
    }
    return Retrofit.Builder()
        .addConverterFactory( factory = json.asConverterFactory( contentType = "application/json".toMediaType()))
        .baseUrl( baseUrl = "https://dummyjson.com/")
        .client(okHttpClient)
        .build()
}
```

Sloj 3: API Service Interface

Definišite tijelo odgovora ili zahtjeva koje ćete primiti ili poslati kao klasu podataka.

@SerialName omogućava nam da koristimo naša vlastita imena za polja umjesto njihovih originalnih iz JSON-a.

```
1 Usage
@Serializable
data class Recipe(
    val id: Int,
    val name: String,
    val ingredients: List<String>,
    val instructions: List<String>,
    @SerialName( value = "prepTimeMinutes")
    val prepTime: Int,
    @SerialName( value = "cookTimeMinutes")
    val cookTime: Int,
    val servings: Int,
    val difficulty: String,
    val cuisine: String,
    @SerialName( value = "caloriesPerServing")
    val calories: Int,
    val tags: List<String>,
    @SerialName( value = "userId")
    val user: Int,
    val image: String,
    val rating: Double,
    @SerialName( value = "reviewCount")
    val reviews: Int,
    @SerialName( value = "mealType")
    val mealTypes: List<String>
)
```

Sloj 3: API Service Interface

Definirajte metode koje predstavljaju specifične API krajnje tačke

Imajte na umu da je omotač Response<T> opcionalan, ali vam daje pristup više HTTP informacija poput statusnih kodova.

```
14 Usages
interface RecipesApiService {
    1 Usage
    @GET( value = "recipes")
    suspend fun getAllRecipes(): Response<RecipeListResponse>

    1 Usage
    @GET( value = "recipes/search")
    suspend fun searchRecipes(
        @Query( value = "q") query: String
    ): Response<RecipeListResponse>

    @POST( value = "recipes/add")
    suspend fun addRecipe(
        @Body recipe: Recipe
    ): Response<Recipe>
}
```

Sloj 3: API Service Interface

Kreirajte instancu API servisa koristeći Retrofit instancu i interfejs koji ste upravo definisali.

```
1 Usage
@Provides
@Singleton
fun provideRecipesApiService(retrofit: Retrofit): RecipesApiService {
    return retrofit.create(RecipesApiService::class.java)
}
```

Učitavanje slika: Coil

Za učitavanje URL-ova slika možete koristiti biblioteku Coil.

Coil pruža UI komponente za prikazivanje slike jednostavno iz njenog URL-a.

```
AsyncImage(
    model = "https://example.com/image.jpg",
    contentDescription = null,
)
```

```
implementation("io.coil-kt.coil3:coil-compose:3.3.0")
implementation("io.coil-kt.coil3:coil-network-okhttp:3.3.0")
```

Dodatno: dozvole

Da biste pristupili nekim funkcijama u vašoj aplikaciji, ponekad ćete možda morati dodati dozvole u `AndroidManifest.xml`.

Stvari poput:
Internet
Lokacije
Biblioteka slika/videozapisa
zahtjeva dozvole.

```
<!-- Need internet perms (lec6) -->
<uses-permission android:name="android.permission.INTERNET" />
```