Predavanje br. 9

Razvoj mobilnih aplikacija i servisa

Sadržaj

- Životni ciklus aktivnosti
- Arhitektura aplikacije
- ViewModel
- Demo

Activity Life Cycle

Promjena konfiguracije

Kotlin Playground

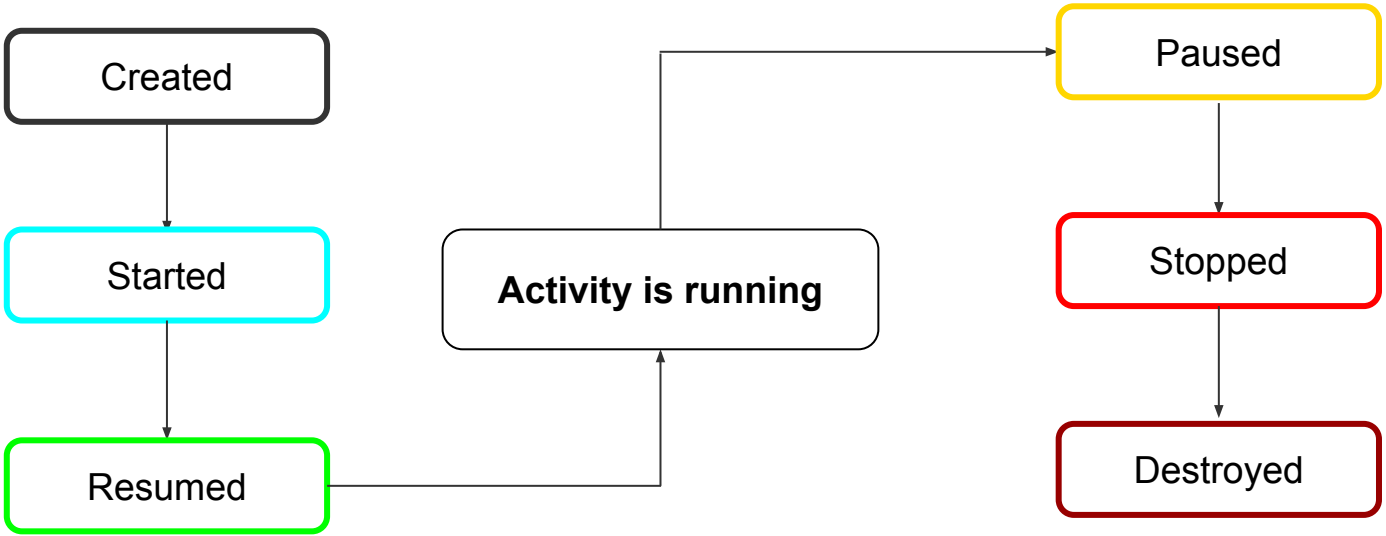
```
fun main() {
    println("Hello, world!")
}
```

Android aplikacija

```
class MainActivity : ComponentActivity() {

    override fun onCreate(...) {
        super.onCreate(savedInstanceState)
        setContent {
            HelloWorldTheme {
                ...
            }
        }
    }
    ...
```

Activity - životni ciklus

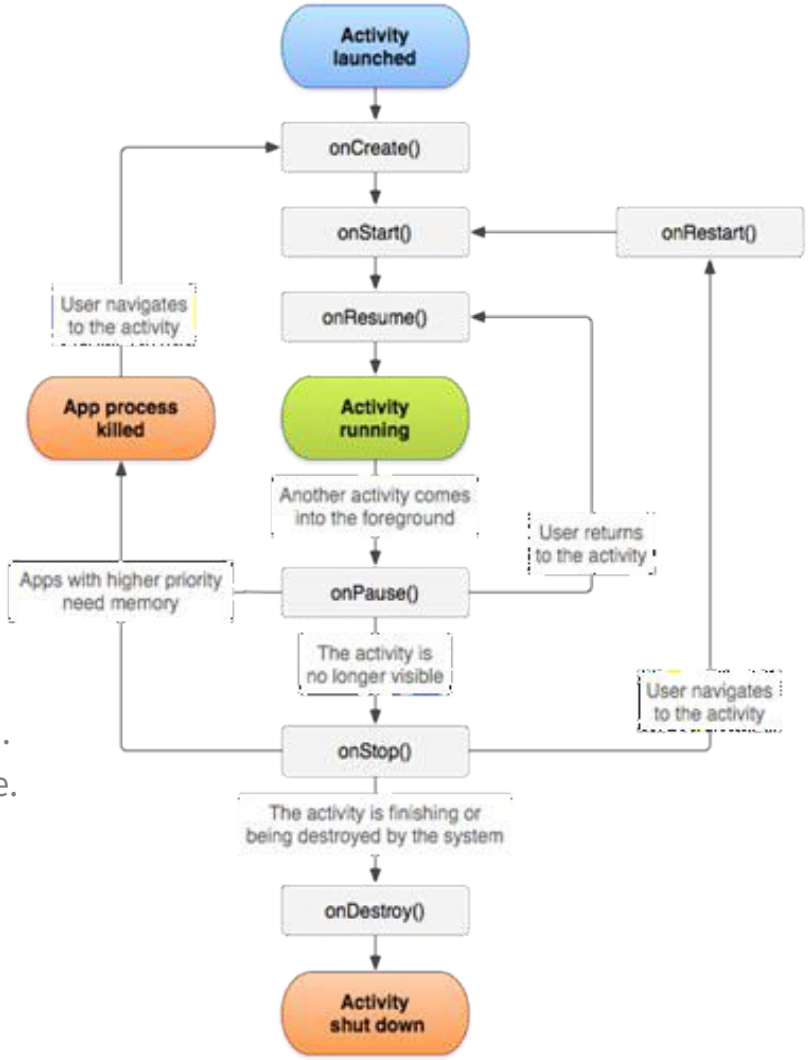


Activity Life Cycle

- onCreate()
- onStart()
- onResume()
- onPause()
- onStop()
- onRestart()
- onDestroy()

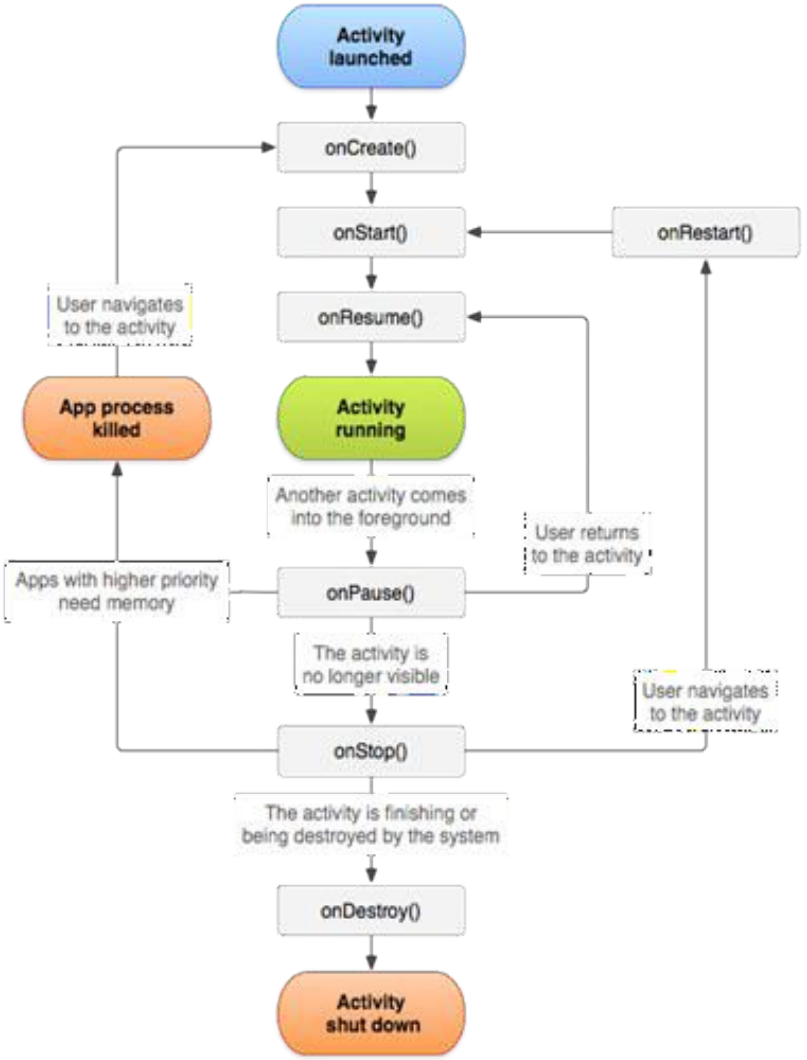
Activity Life Cycle

- onCreate()
  - Poziva se kada se Activity prvi put kreira.
  - Inicijalizacija aplikacije (ViewModel, repozitoriji).
  - Učitaj UI pomoću setContent {}.
- onStart()
  - Activity postaje vidljiva korisniku.
  - Pokreni lagane UI zadatke ako je potrebno.
- onResume()
  - Activity je u fokusu i korisnik može imati interakciju.
  - Nastavi senzore, animacije ili druge aktivne procese.



Activity Life Cycle

- onPause()
  - Activity djelimično gubi fokus (npr. otvaranje dijaloga ili prelazak u drugu aplikaciju).
  - Pauziraj animacije, zaustavi osjetljive procese.
- onStop()
  - Activity više nije vidljiva.
  - Zaustavi “teške” operacije (lokacija, kamera...).
  - Po potrebi sačuvaj privremeno stanje.
- onRestart()
  - Poziva se prije povratka iz onStop().
  - Ponovo pokreni ono što je pauzirano u onStop()
- onDestroy()
  - Konačno čišćenje prije uništavanja Activity-ja.

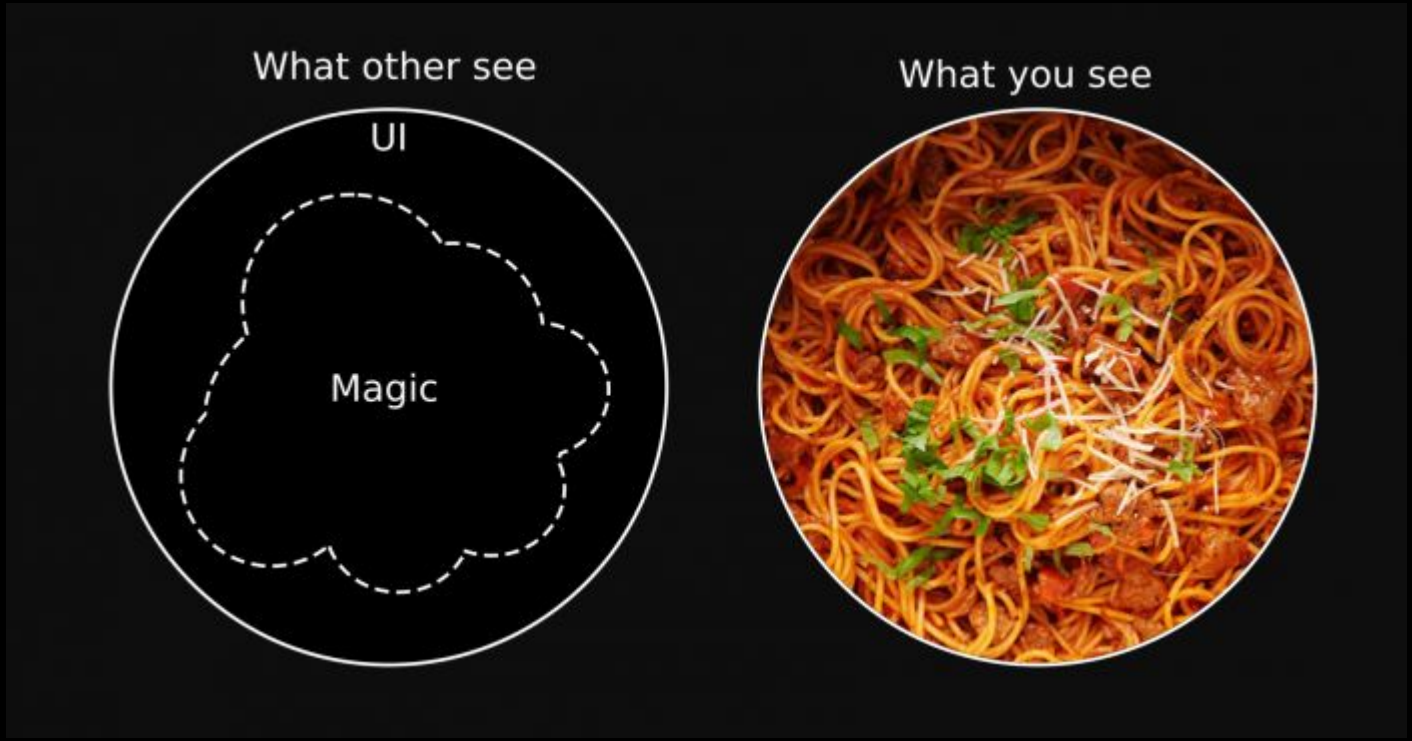


Primjer


12:00
Dessert Clicker
Today's Special
Desserts sold
18
Total Revenue
$155

Arhitektura aplikacije

Zašto arhitektura?



Aplikacije sa dobro definiranom arhitekturom

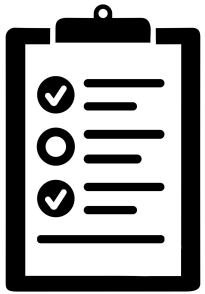
- Lakše je:
  - Testirati aplikaciju
  - Održavati je
  - Proširivati funkcionalnosti
  - Raditi u timu na njenom razvoju

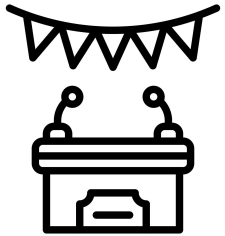
Separation of concerns

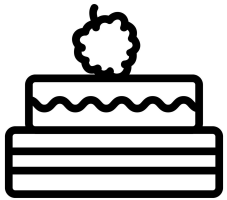
Razdvajanje odgovornosti

Separation of concerns

Planiranje vjenčanja



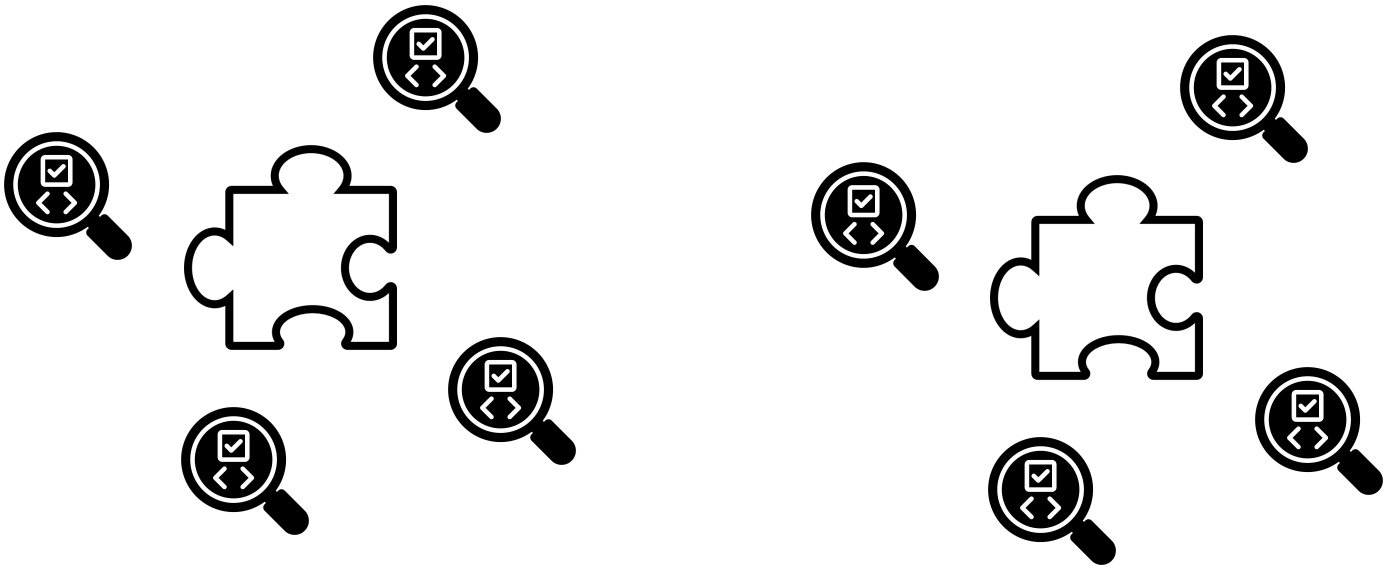



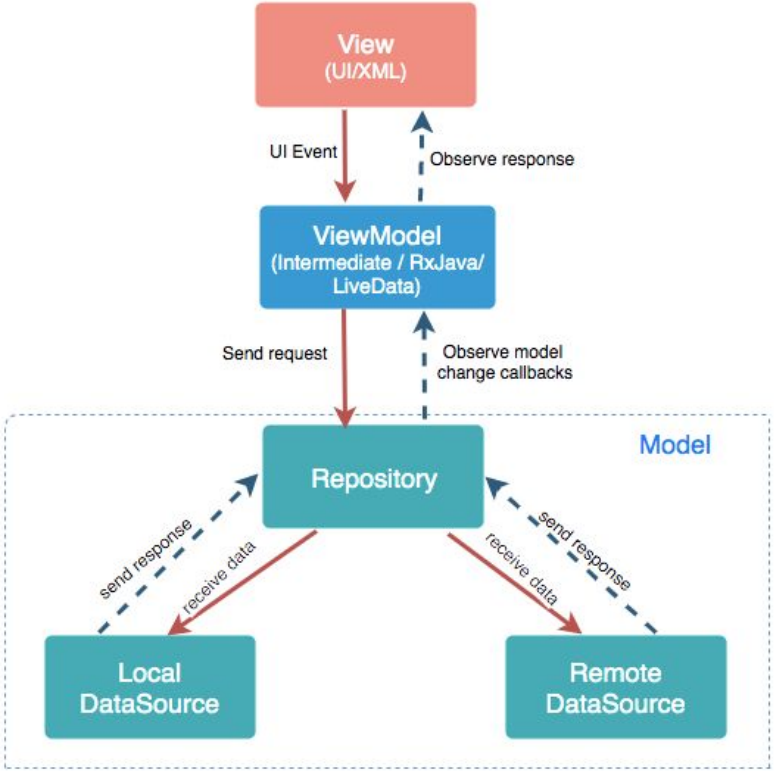

Separation of concerns

Razdvajanje odgovornosti - enkapsulacija

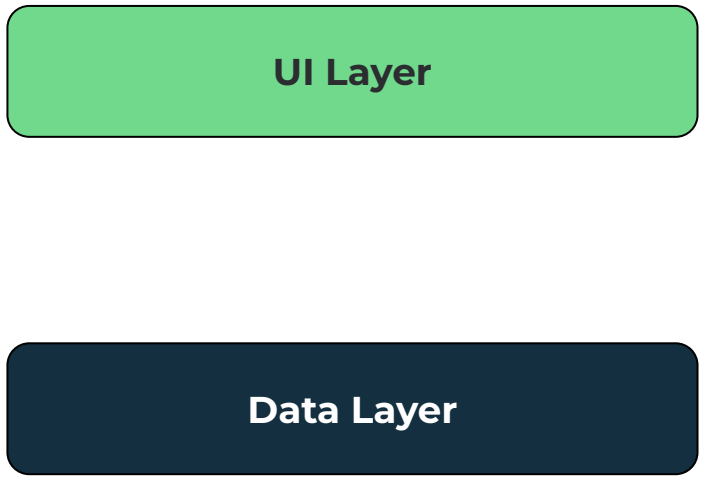
Testiranje



MVVM arhitektura



Slojevita arhitektura



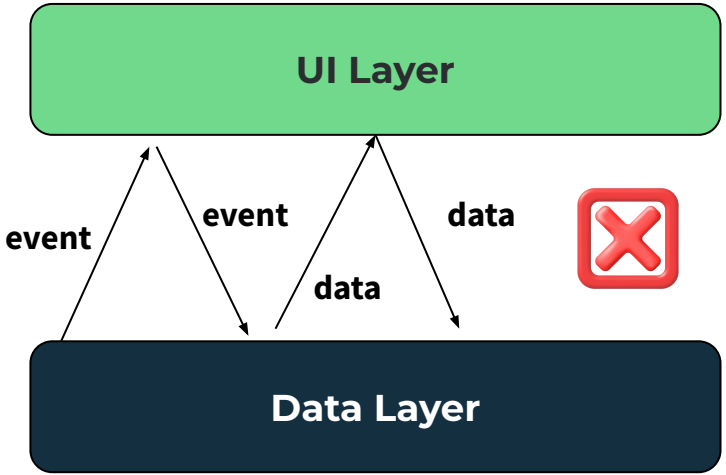
Slojevita arhitektura

Sloj	Funkcije	Primjeri
UI	• Interakcija s korisnikom • Interakcija s operativnim sistemom • Izgled aplikacije i ekrani	• Tekst • Slike • Dugmad / ponašanje na klik • Polja za unos teksta / korisnički input

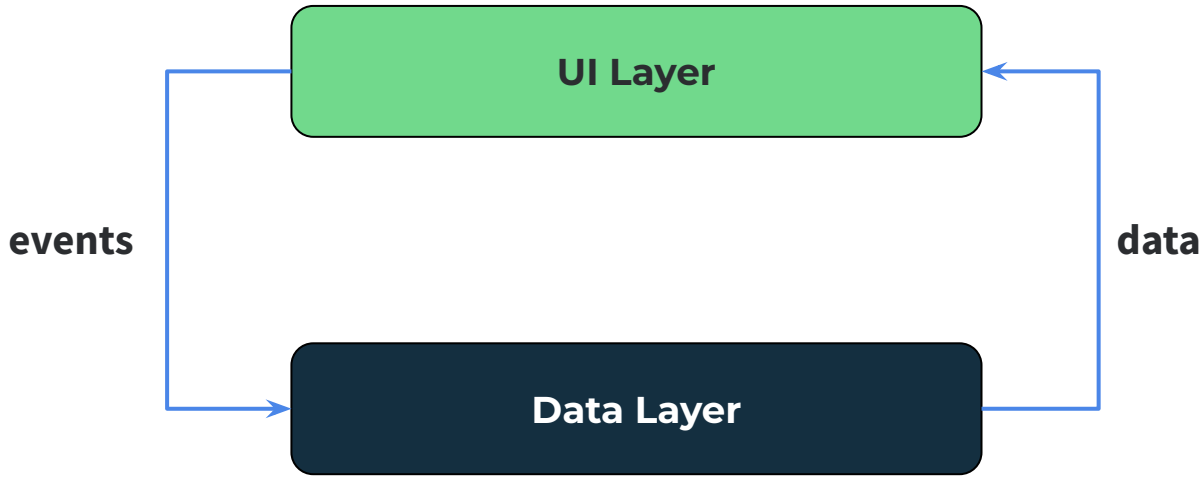
Slojevita arhitektura

Sloj	Funkcije	Primjeri
UI	• Interakcija s korisnikom • Interakcija s operativnim sistemom • Izgled aplikacije i ekrani	• Tekst • Slike • Dugmad / ponašanje na klik • Polja za unos teksta / korisnički input
Data	• Informacije koje aplikacija koristi	• Naslov e-maila, datum, predmet, sadržaj (e-mail aplikacija) • Naslov članka, sadržaj, slike (aplikacija za vijesti)

Slojevita arhitektura

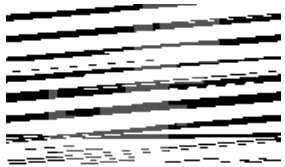


Unidirectional Data Flow

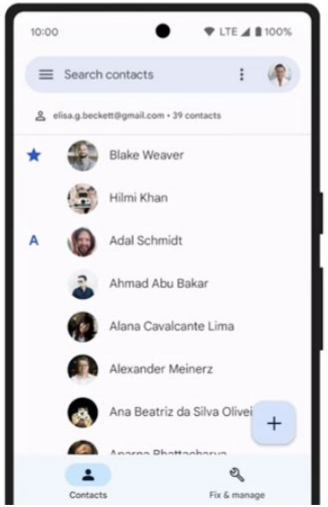


Events (događaji)

LIKE



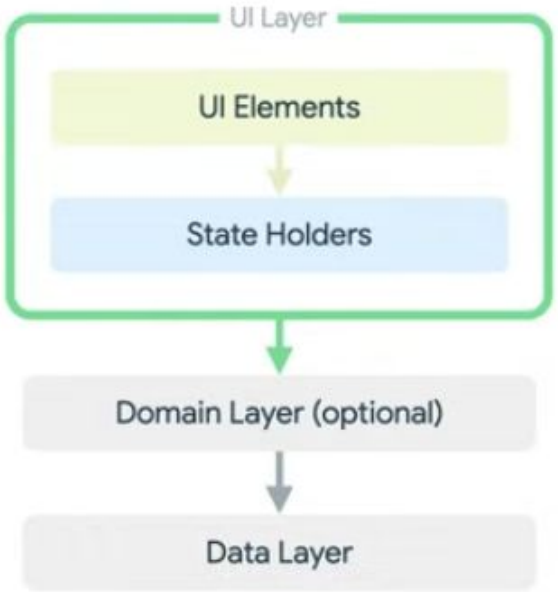
Primjer


10:00
LTE 100%
Search contacts
elisa.g.beckett@gmail.com • 39 contacts
Blake Weaver
Hilmi Khan
A
Adal Schmidt
Ahmad Abu Bakar
Alana Cavalcante Lima
Alexander Meinerz
Ana Beatriz da Silva Olivei
Contacts
Fix & manage


UI Layer (prikaz liste)
Dodavanje novog kontakta
Podaci o kontaktima za prikaz
Data Layer (podaci o kontaktima)

UI sloj

Arhitektura aplikacije



UI sloj - pipeline

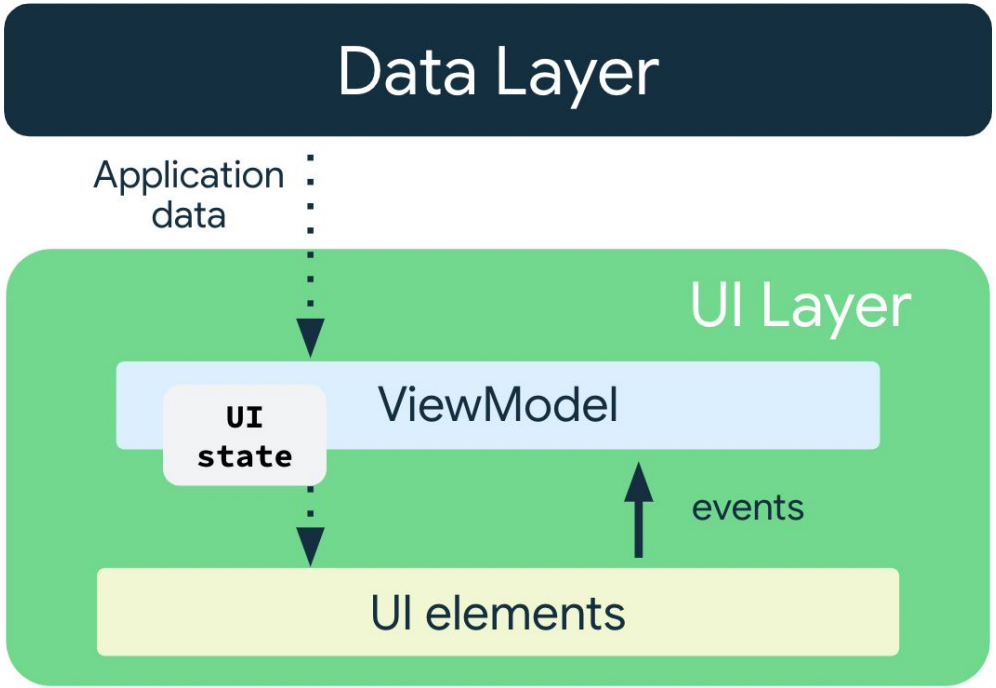
1. App data → UI data
2. UI data → UI elements
3. User events → UI changes
4. Repeat

Koncepti UI sloja

1. Define UI state
2. Production of UI state
3. Expose UI state
4. Consume UI state

Šta je UI?

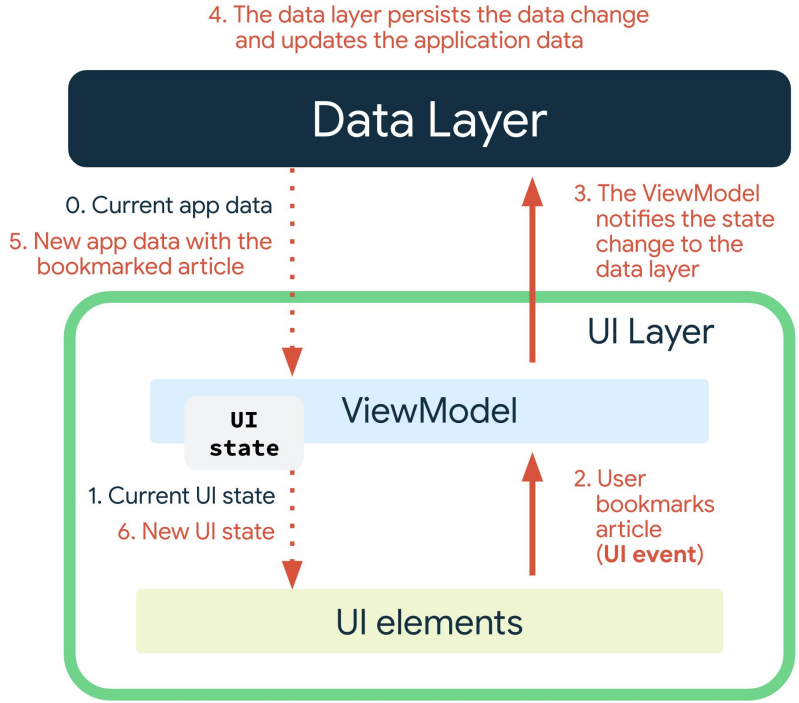
State holder



UI pipeline

Dagger in Kotlin: Gotchas and Optimizations

Manuel Vivo · 4 min read



ViewModel

- Šta je ViewModel
  - Komponenta koja upravlja sa UI state i poslovnom logikom.
  - Čuva podatke tokom promjene konfiguracije (npr. rotacija ekrana).
  - Odvaja UI od logike → razdvajanje odgovornosti.
- Kako koristiti
  - Ubacivanje u build.gradle (Module: app) skriptu

```
dependencies {
        ...
    implementation("androidx.lifecycle:lifecycle-viewmodel-compose:2.8.0")

}
```

UI Layer

1. Definiranje UI state
    - šta UI treba da prikaže
    - obično data class koja sadrži sve podatke za jedan ekran.

```
data class GameUiState(
    val score: Int,
    val currentWord: String,
    val isError: Boolean
)
```

UI Layer

2. Produkcija UI state

- ViewModel proizvodi i ažurira UI state.
- On odlučuje:
  - kada se stanje mijenja,
  - kako se obrađuju korisničke akcije,
  - koje podatke UI dobija.

```
_state.value = _state.value.copy(score = newScore)
```

UI Layer

3. Izlaganje UI state

- ViewModel izlaže UI state UI sloju.
- Najčešće kao:
  - StateFlow
  - LiveData
  - ili MutableState (Compose)
- UI može čitati, ali ne može mijenjati stanje direktno.

```
private val _state = MutableStateFlow(GameUiState())
val state: StateFlow<GameUiState> = _state
```

UI Layer

4. Konzumiranje UI state

- UI (Compose) čita to stanje i prikazuje ga na ekranu.

```
val uiState by viewModel.uiState.collectAsState()
Text(text = uiState.currentWord)
```

Demo


Prepoznaj riječ
0/10
tas
Prepoznaj riječ koristeći sva slova
Unesite svoju riječ
Pošalji
Preskoči
Rezultat: 0