MVVM arhitektura

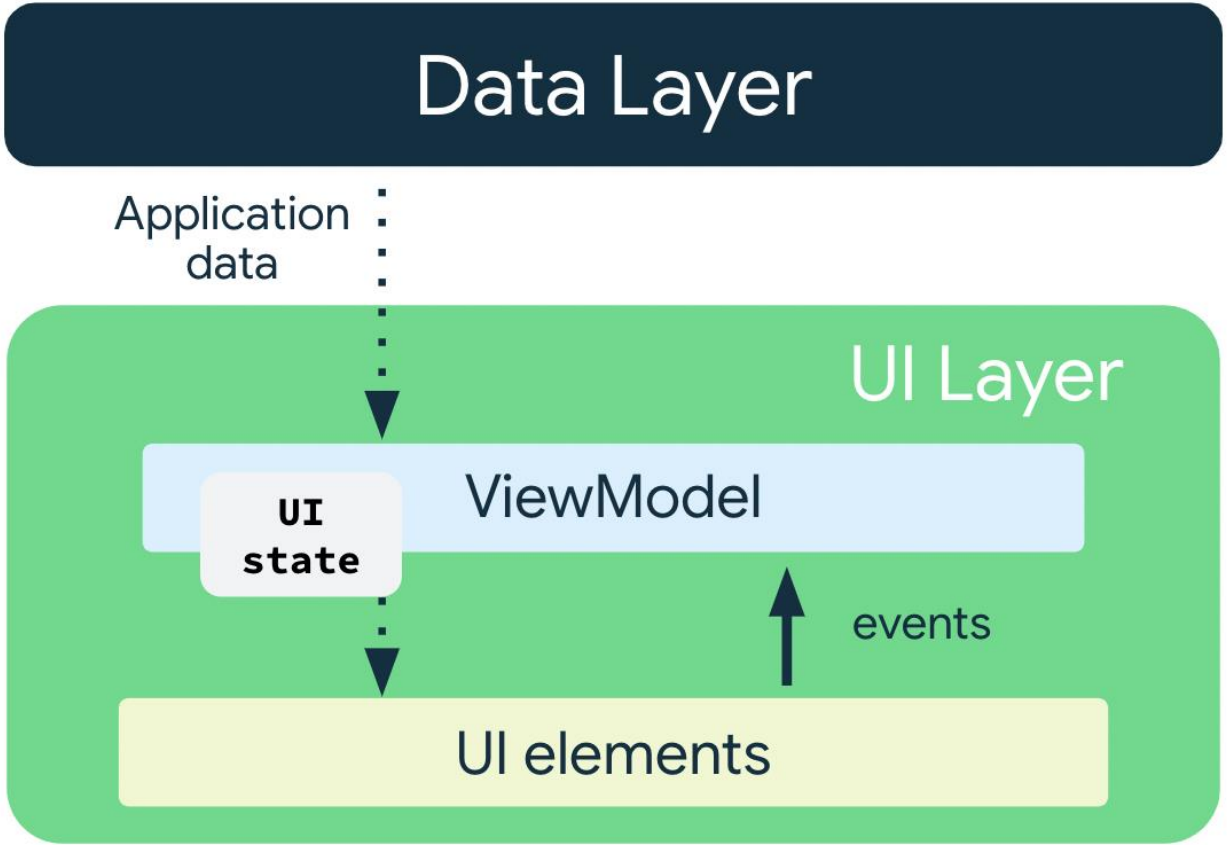
ViewModel i state u Jetpack Compose-u

prof. dr Bratislav Predić
dipl. inž. Nevena Tufegdžić

Razvoj mobilnih aplikacija i servisa
Računarstvo i informatika

2024.

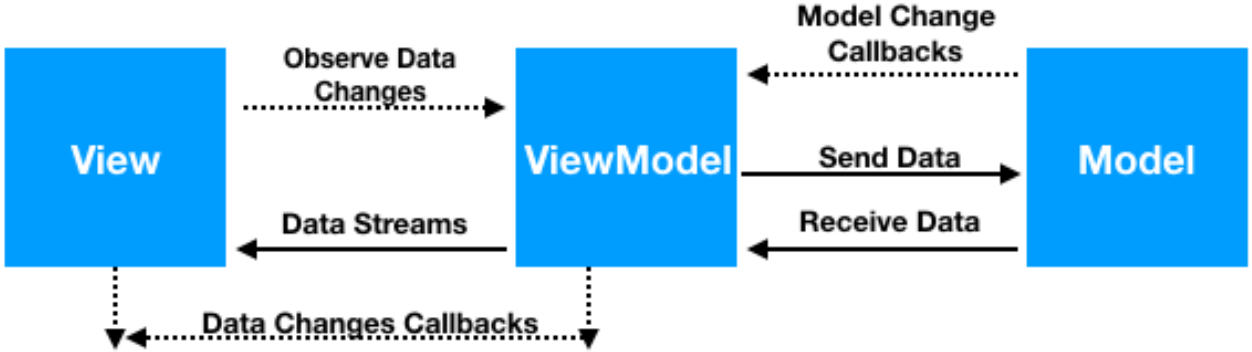
UI layer



prof. dr Bratislav Predić
dipl. inž. Nevena Tufegdžić

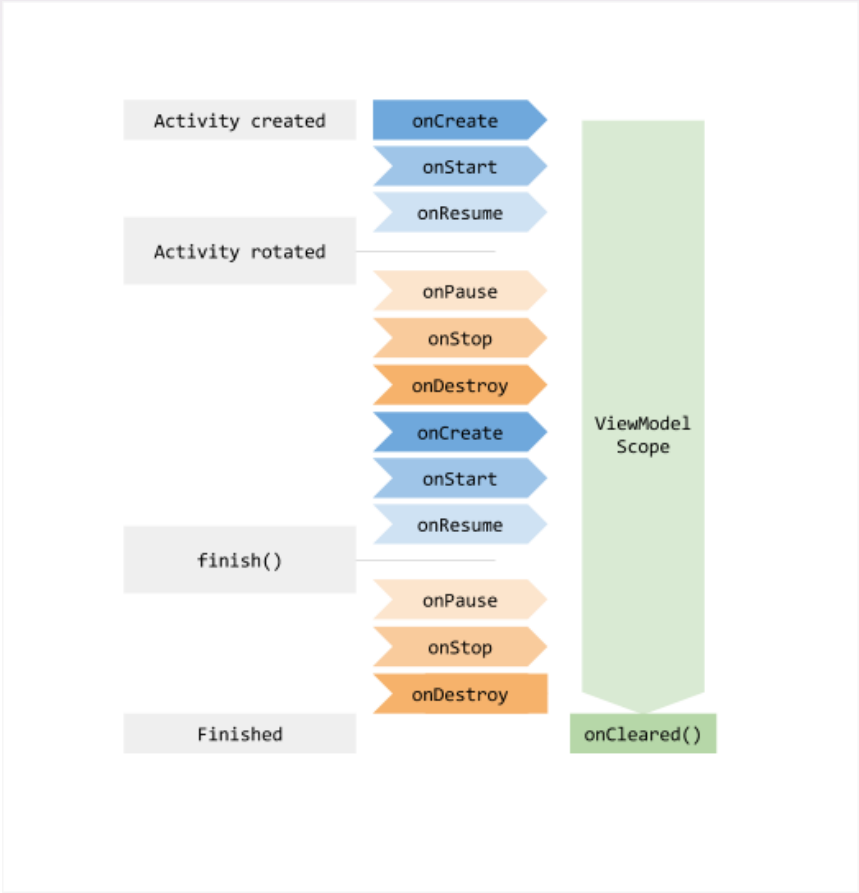
MVVM obrazac

- Model-View-ViewModel



ViewModel

- Posrednik između modela (podataka) i pogleda (UI aplikacije)
- U Androidu, ViewModel je komponenta koja čuva stanje (state) korisničkog interfejsa i zadužena je za njegovo ažuriranje
- Ne uništava se kada se Activity uništi, nego tek pri terminiranju procesa aplikacije



Immutability principle

- Princip koji state treba da podržava je *immutability principle*
  - Svaki deo state-a treba da bude nepromenljiv, u smislu da je nemoguće menjati ga ali je moguće dodeliti novu vrednost
- Posledica: nemoguće je doći u situaciju da dva dela aplikacije istovremeno menjaju podatke
  - Samo ViewModel sme da menja state, a UI poziva metode za to
- Ovo se posebno odnosi na korišćenje Data klasa

Immutability principle

✓ Immutable i grupisano po funkcionalnosti:

```
data class NewsItemUiState(
    val title: String,
    val body: String,
    val bookmarked: Boolean = false,
    ...
)
```

× Mutable, nije grupisano:

```
var title: String
var body: String
var bookmarked: Boolean = false
...
```

ViewModel - pravila

- Koristiti ViewModel kao state-holder ekrana, ne malih reusable komponenti (npr. custom checkbox)
- ViewModel i UI su spregnuti, ali odvojeni entiteti
- Ne držati reference na nečemu što ima svoj lifecycle u ViewModel-u (npr. referencu ka Activity-ju)
- Ne prosleđivati referencu ka ViewModel-u klasama, funkcijama ili manjim UI komponentama – ViewModel živi u okviru aktivnosti ili screen-level Composable funkcije

Stateful objekti

- StateFlow i MutableStateFlow
- Predstavlja tok koji se može posmatrati (slično Observer patternu)
- Za razliku od Observer patterna, nije potrebno vršiti subscribe na StateFlow
  - Android ovo vrši implicitno, tako da je pogodan za korišćenje u okviru Composable funkcija za praćenje promenljivih vrednosti
- StateFlow enkapsulira trenutnu vrednost promenljive u okviru value parametra

StateFlow

- *A SharedFlow that represents a **read-only state** with a single updatable data value that **emits updates** to the value to its collectors. A state flow is a **hot flow** because its active instance exists independently of the presence of collectors. Its current value can be retrieved via the **value** property.*
  - (Android dokumentacija)

1. Immutability principle
2. Consumer-i dobijaju update na novo stanje
3. Ne uništava se pri rekompoziciji
4. Pristup aktuelnom stanju enkapsuliran u *value* atributu

MutableStateFlow

- Pruža isto što i StateFlow, osim što je mutable – pruža setter funkciju
  - Moguće je promeniti mu vrednost
  - Ovakav deo ViewModel-a ne treba otkrivati ka spoljašljnosti
- MutableStateFlow se koristi za ažuriranje unutrašnjeg stanja u ViewModel-u, dok StateFlow preuzima vrednost od MutableStateFlow promenljive i to stanje otkriva spoljašnjosti
  - StateFlow nema setter, tako da nema brige od eksternog mešanja

ViewModel implementacija

- Kako bi ViewModel bio raspoloživ za korišćenje u projektu, potrebno je ući u build.gradle (app) i dodati sledeću zavisnost:

```
dependencies {
    // other dependencies

        implementation("androidx.lifecycle:lifecycle-viewmodel-compose:2.7.0")
    //...
}
```

ViewModel implementacija

- Definicija ViewModel-a:

```
class MyViewModel : ViewModel() {
    private val _stateVariable : MutableStateFlow<String>
        = MutableStateFlow("initial value")
    val stateVariable: StateFlow<String> = _stateVariable.asStateFlow()

    fun updateState() {
        // updating state here
    }
}
```

ViewModel - korišćenje

- Neka postoji Composable koji treba da svaki put prikazuje ažurno stanje aplikacije
- Stanje može da se menja, npr. klikom na dugme
- Pravila: svaki Composable dobija samo onaj deo ViewModel-a koji mu je neophodan, i ništa više
  - Composable za prikaz će dobiti samo referencu na state slice koji prikazuje
  - Dugme će dobiti referencu na funkciju ViewModel-a koja može da menja state
  - Jedino glavni (screen-level) Composable može da dobije ceo ViewModel

ViewModel - korišćenje

```
@Composable
fun MainScreen(viewModel: MyViewModel = viewModel()) {
    val stateSlice by viewModel.stateVariable.collectAsState()
            StateDisplay(stateToDisplay = stateSlice)
            Button( onClick = { viewModel.updateState() }) {
                Text(text = "Update state")
            }
        }
    }
}
```

- Napomena: primetiti da se kao default parametar ne instancira MyViewModel klasa, već koristi bibliotečka funkcija **viewModel()**

ViewModel - korišćenje

```
@Composable
fun StateDisplay( stateToDisplay: String) {
    Surface(modifier = Modifier.fillMaxWidth()) {
        Text(text = stateToDisplay)
    }
}
```

ViewModel - napomene

- Ako se kao parametar Composable funkcije prosleđuje instanca konkretnog ViewModel-a, onda će pri svakoj rekompoziciji biti kreiran nov ViewModel
  - Posledica: promene u konfiguraciji izazivaju gubitak podataka
- Funkcija viewModel() kreira ViewModel ako nikada do sada nije postojao, a u suprotnom preuzima referencu na postojeći
  - Posledica: pri promeni konfiguracije ne instancira se nov ViewModel, već se vraća stari => nema gubitka podataka

Alternativa StateFlow notaciji

- Umesto deklaracije StateFlow i MutableStateFlow parova, može se iskoristiti sledeća notacija:

```
var stateVariable = mutableStateOf("initial value")
        private set
```

- Ekvivalent ovome:

```
private val _stateVariable : MutableStateFlow<String>
         = MutableStateFlow("initial value")
    val stateVariable: StateFlow<String> =
_stateVariable.asStateFlow()
```

Literatura

- Android App Architecture (MVVM)
- ViewModel Overview (Android Developer)
- ViewModel Codelab
- StateFlow i SharedFlow
- Live Data

Hvala na pažnji!