Android MVI Architecture #2

Thuy

Growth Session #14 - May 17-18 2018

Intro

- Session #1
- In Session #2:
  - Deeply understand and implement MVI
  - Continue with demo application

MVI Architecture

- Embrace reactive and functional programming
- Has only one entry point to forward data between View and ViewModel
  - The View takes input from ViewModel and emit back intents
  - The ViewModel take input from the View and emit back by view states

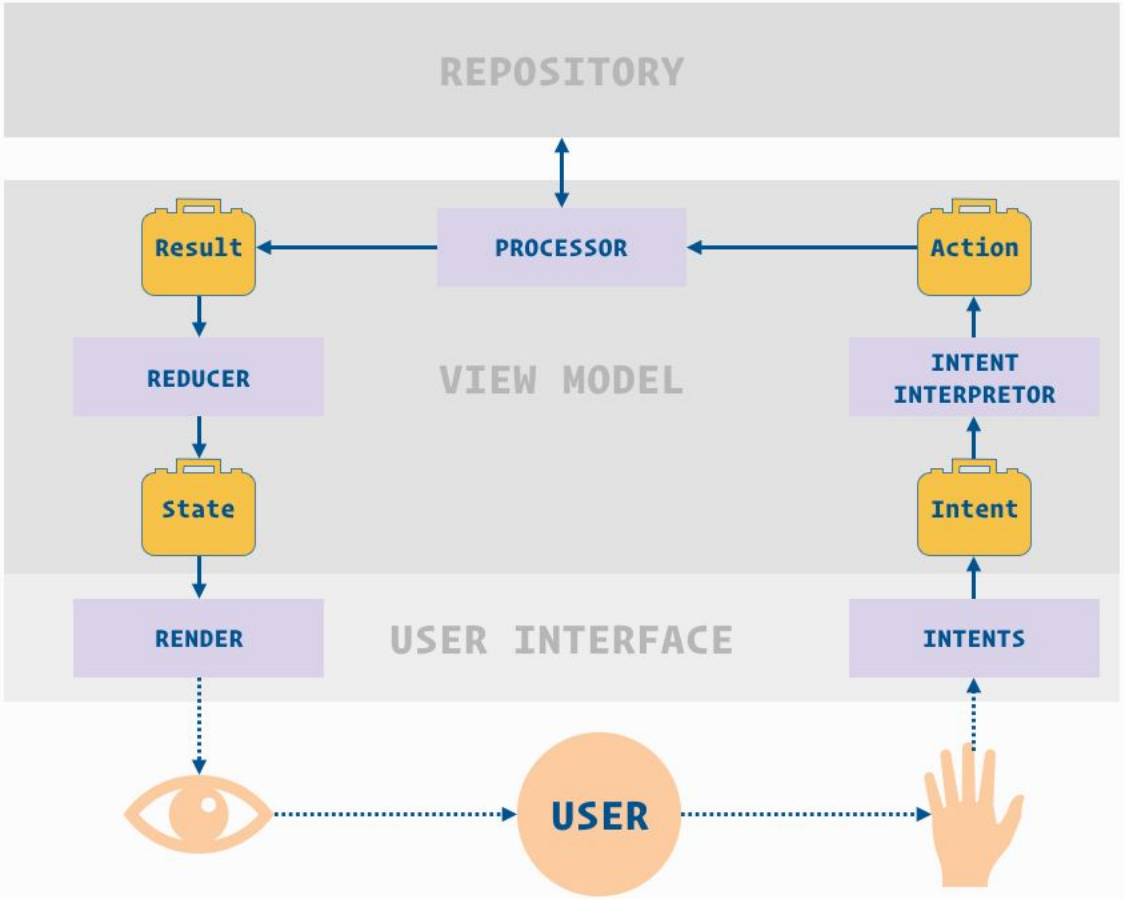
```
public interface MviView {
  Observable<MviIntent> intents();

  void render(MviViewState state);
}

public interface MviView {
  Observable<MviIntent> intents();

  void render(MviViewState state);
}
```

Implement MVI



Reducer and ViewState

- Reducer
  - Generate the ViewState
  - Take the latest ViewState available, apply the latest result and return a whole new ViewState

```
private val reducer = BiFunction { previousState: TasksViewState, result: TasksResult ->
  when (result) {
    is LoadTasksResult ->  {
    ^BiFunction previousState.copy(isLoading = true)
    }
    is CompleteTaskResult -> {
      ^BiFunction previousState.copy(taskComplete = false)
    }
```

- ViewState
  - Contains all the information the View needs to render itself

Conclusion: when to use MVI?

- If you are fan of Functional Programing, Reactive Programing and Pure function
- User-centric: MVI puts the user right at the heart of discussions

Thanks!

Contact Nimbl3

hello@nimbl3.com

399 Sukhumvit Road, Interchange 21
Klongtoey nua, Wattana
Bangkok 10110

28C Stanley St,
Singapore 068737

20th Floor, Central Tower
28 Queen's Road
Central, Hong Kong

nimbl3.com