Mobiquity Test Challenge

24th Febreuary 2019

GETTING STARTED

These instructions will get you a copy of the project up and running on your local machine:

1. Clone the project repository from Github.

   ```
   git clone https://github.com/mfathy/MobiqTest.git
   ```

2. Open **Android studio**, Select File | Open... and point to the the project, wait until the project syncs and builds successfully.

3. Run the project using Android studio.

DISCUSSION

Data Sources

There is one level of data persistence:

- Network with http caching.

The chosen fetch of data is simple:

- In every get products request:
  - Return remote copy.
  - Return cached copy:
    - If there is Internet, get the cache that was stored 5 seconds ago.
    - If there is no Internet, get the cache that was stored 7 days ago.

Dependency Injection

I've used **dagger** for dependency injection, also I've added different component and modules for test layer.

Testing

I have included the required Instrumentation, Unit and UI tests with the project:

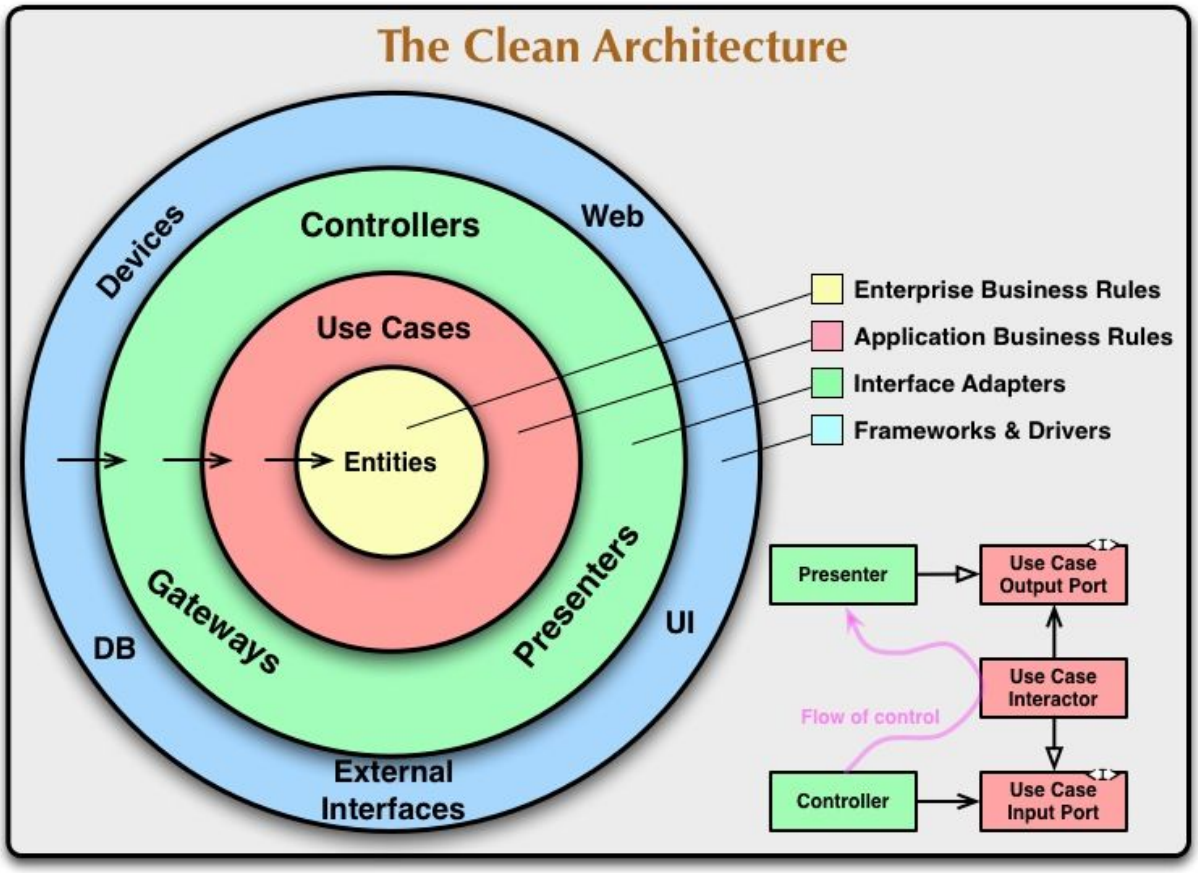
- Unit tests for most of the app classes.
- Integration tests for testing integration between layers components and the layer itself.
- Ui tests using Espresso:
    - *For espresso testing I have use normal way of mocking the response using dagger injection >> check espresso tests on master branch.*
    - *Another way of testing using Okhttp mock web server to easily mock valid, invalid responses and server errors to show how the UI will acts at those situations >> check espresso tests on espresso-tests-mockwebserver branch.*

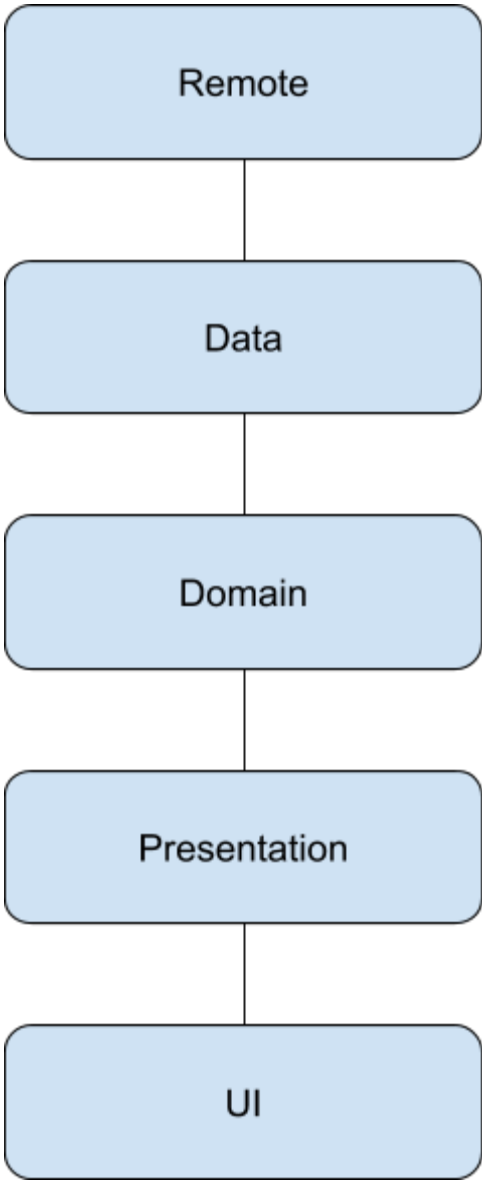
To run all test case at once look for the following test suites:

- **UnitTestSuite**
- **InstrumentedTestSuite**

Architecture

Following **clean architecture** with **MVVM,** which has some of clean architecture principles except layer independence.


The Clean Architecture
Devices
Controllers
Web
Use Cases
Entities
Gateways
Presenters
DB
UI
External
Interfaces
Enterprise Business Rules
Application Business Rules
Interface Adapters
Frameworks & Drivers
Presenter
Use Case
Output Port
Use Case
Interactor
Flow of control
Controller
Use Case
Input Port



Relation between layers:

- The UI layer contains views which observes to the presentation layer and listen for data changes.
- The Presentation layer receives UI layer calls and respond to domain layer interactors and run them.
- The Domain layer calls data layer.
- The Data layer manage which source should read from “in our case - only remote”, then serve the request back to the domain layer.

MVVM

The MVVM architecture.

- **Model**: refers either to a domain model, or to the data access layer.
- **View**: refers to the UI.
- **View model**: is an abstraction of the view exposing public properties and commands. It has a binder, which automates communication between the view and its bound properties in the view model.

A typical scenario:

1. A user opens the app and activity created and starts listening for ViewModel data changes.
2. A call to fetch products in ViewModel is started from the starting activity.
3. ViewModel calls GetProducts use case to start fetching data.
4. GetProducts use case calls data repository asking for data.
5. Data repository determines which data store should read from. “In our case we have remote data store only with http caching to decrease server requests.”
6. Remote data store request the data from the server and respond with **Success** otherwise **Error**.

Libraries

- Common Android X libraries - AndroidX is the open-source project that the Android team uses to develop, test, package, version and release libraries within Jetpack.
- Mockito - A mocking framework used to implement unit tests.
- Dagger - for dependency Injection
- RxJava - Reactive Extensions for the JVM – a library for composing asynchronous and event-based programs using observable sequences for the Java VM.
- Okhttp - An HTTP+HTTP/2 client for Android and Java applications.
- Hamcrest - Junit Matchers
- MockWebServer - A scriptable web server for testing HTTP clients.
- Retrofit - A type-safe HTTP client for Android and Java.
- Gson - a json serialize and deserialize library.
- Android Architecture Components - LiveData, ViewModel.

Thank you.