PegB Test Challenge

13th January 2019

REQUIREMENTS

A mobile application for getting weather forecast of the desirable location.

User should be able to:

a) choose location from drop down view (spinner or something else), after what he/she should be able to

b) select the time range for which the forecast is needed. After selecting required fields user should be able to

c) see forecast info in a new page (view for this page depends on developer).

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/WeatherForcast.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.

Hints:

Due to time constraints I didn't include the following:

- **No caching support [Local not memory].**
- **No Espresso Ui tests.**
- **Some unit tests is missing in ViewModels.**
- **No Comments.**

DISCUSSION

Data Sources

There is only one levels of data persistence:

- Network - Very slow.

The data layer consists of:

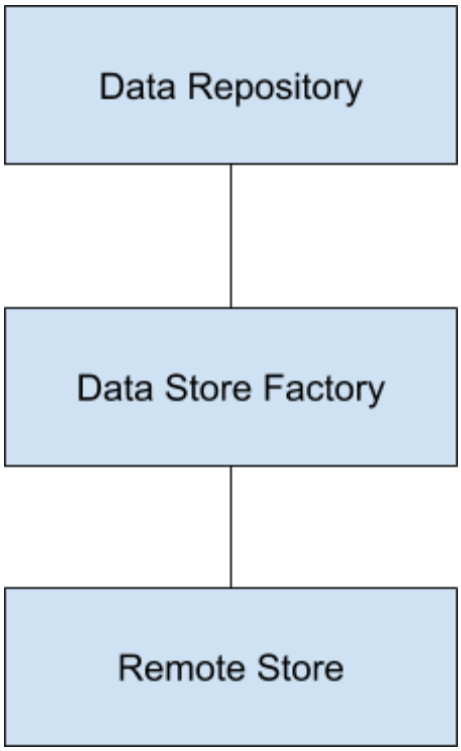
- A repository pattern to provide data outside the layer itself.
- A Remote data store layer to access remote server data.

The chosen fetch of data is simple:

- In every get operation
    - Return remote copy.

Remote data source""

The remote data source uses Okhttp or Retrofit API to call the Backend API.



Dependency Injection

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

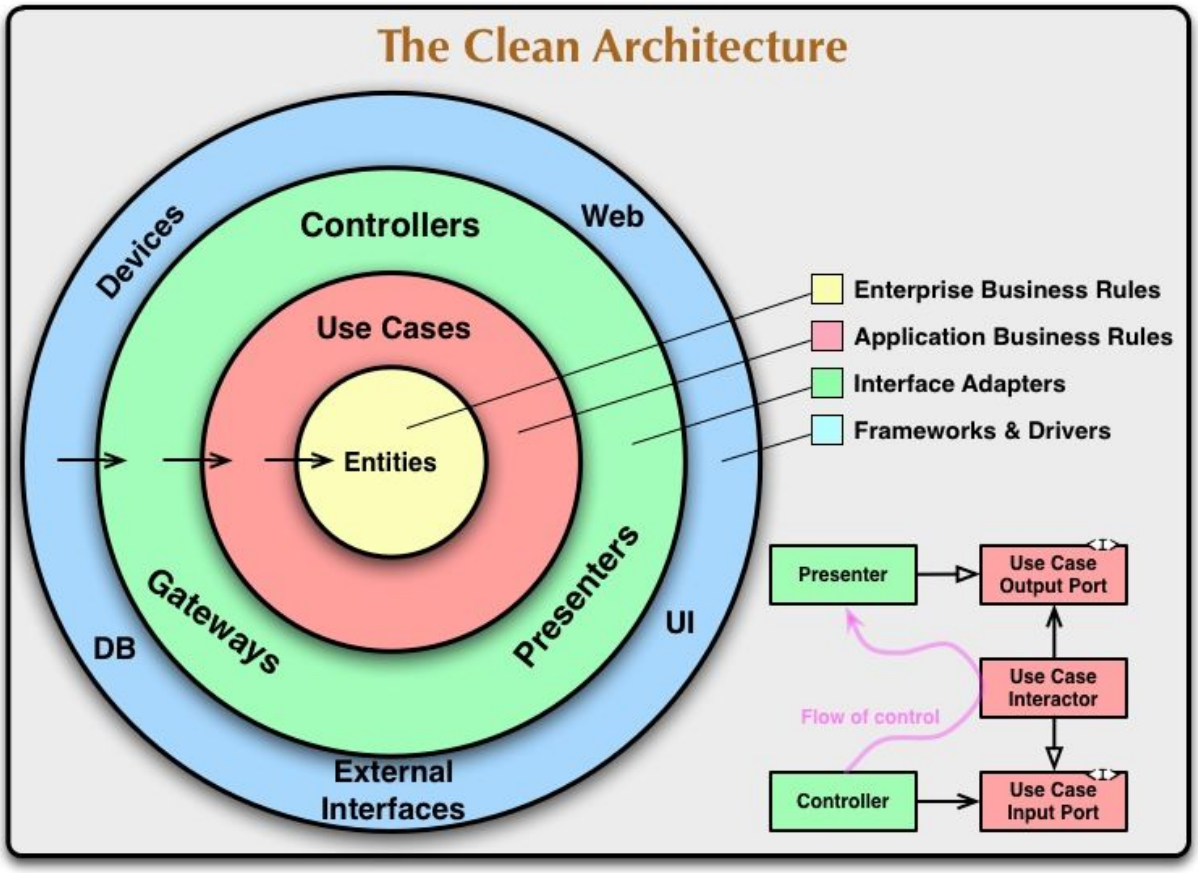
Testing

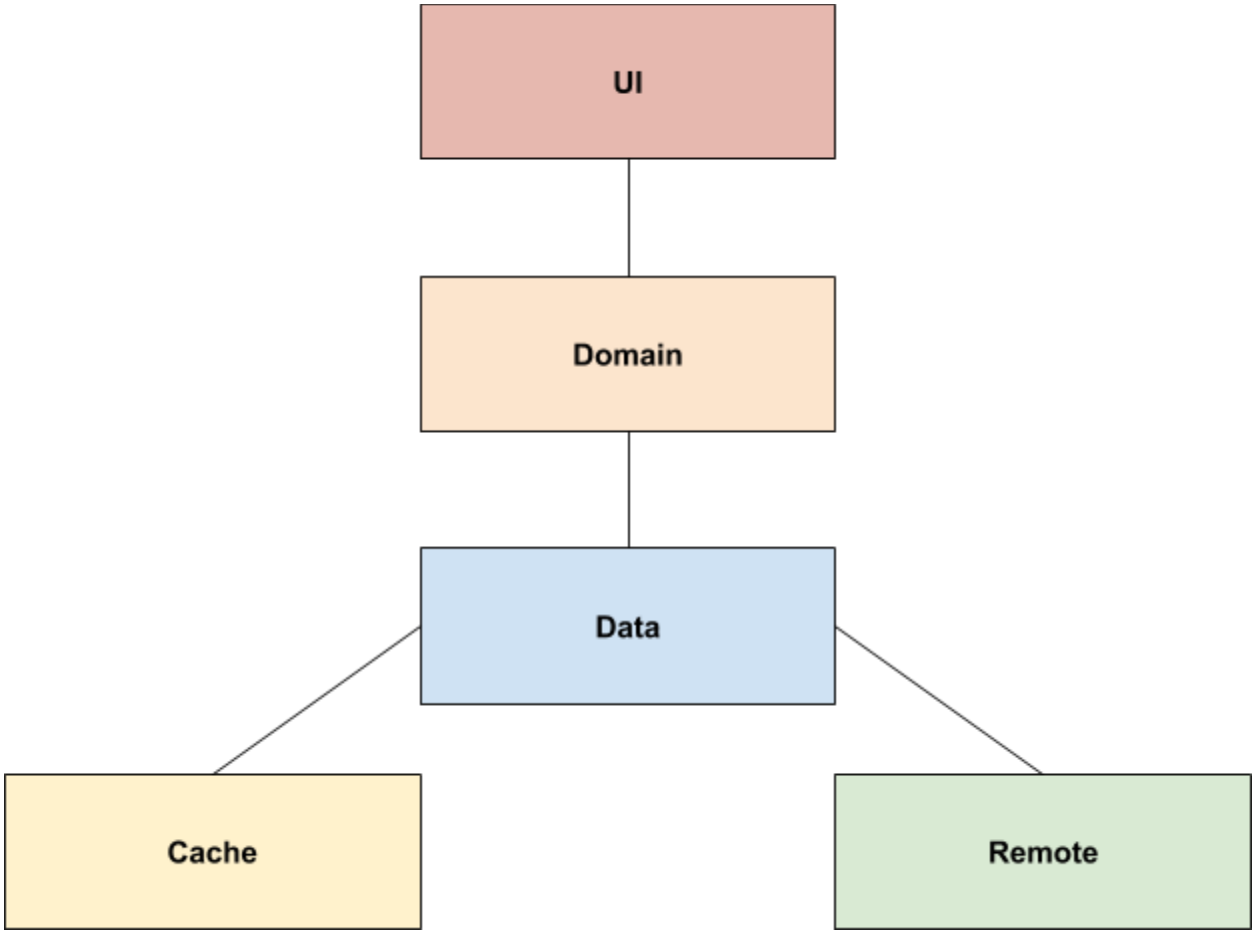
I have included the required Instrumentation, Unit and UI tests with the project:

- Unit tests for most of the app classes.
- Integration tests for testing integration between layers components and the layer itself.

Architecture

I have used a custom version of **clean architecture** with **MVVM,** which has some of clean architecture principles except layer independence, as I've used data layer models across the domain and ui layer.


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



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.

Why MVVM?

- **A good event-driven architecture**: ViewModel exposes streams of events to which the Views can bind to.

- A **one-to-many relation** between View and ViewModel, it uses data binding to ensure that the View and ViewModel remain in sync bi-directionally.

- **Testability**: since presenters are hard bound to Views, writing unit test becomes slightly difficult as there is a dependency of a View. ViewModels are even more Unit Test friendly.

Libraries

- Common Android support libraries - Packages in the com.android.support.* namespace provide backwards compatibility and other features.

- AndroidX Library - AndroidX is a major improvement to the original Android Support Library. Like the Support Library, AndroidX ships separately from the Android OS and provides backwards-compatibility across Android releases. AndroidX fully replaces the Support Library by providing feature parity and new libraries.
- Mockito - A mocking framework used to implement unit tests.
- Play-services - for google maps support.
- Dagger - for dependency Injection
- Gson - a json serialize and deserialize library.
- 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.
- Android Architecture Components - LiveData & ViewModel.

Thank you.