Home24 Test Challenge

12th December 2018

REQUIREMENTS

The application will use Home24 public API to load several articles from remote server and present them to the user one by one. User should mark the article as “liked” or ”disliked” before another article will be presented. User will have a possibility to review his choices by pressing the relevant button.

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.

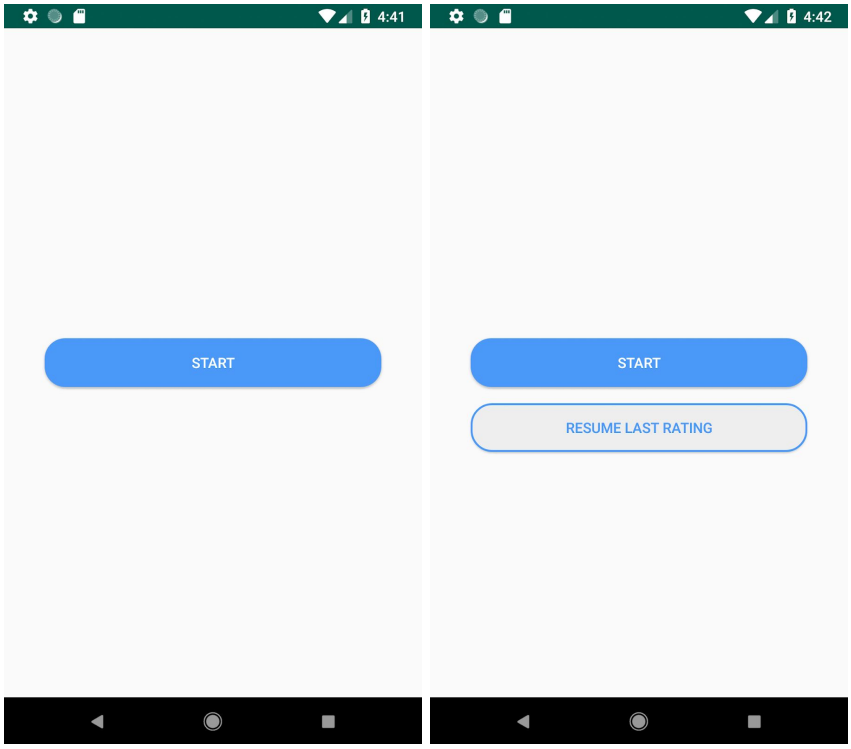
   ```
   git clone https://github.com/mfathy/Android-Article-Reviewer.git
   ```

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.

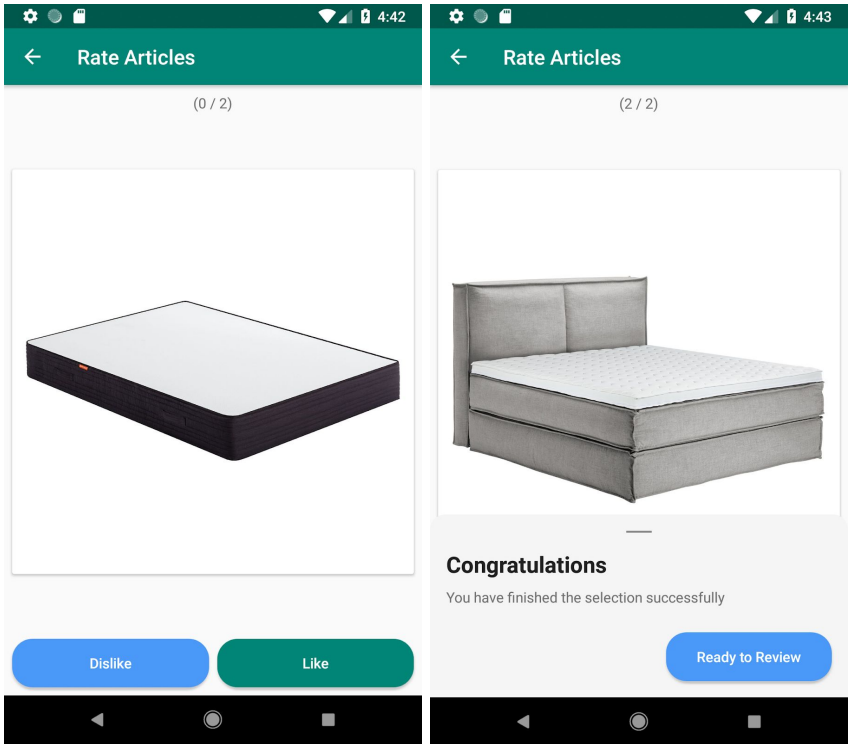
DISCUSSION

User interface



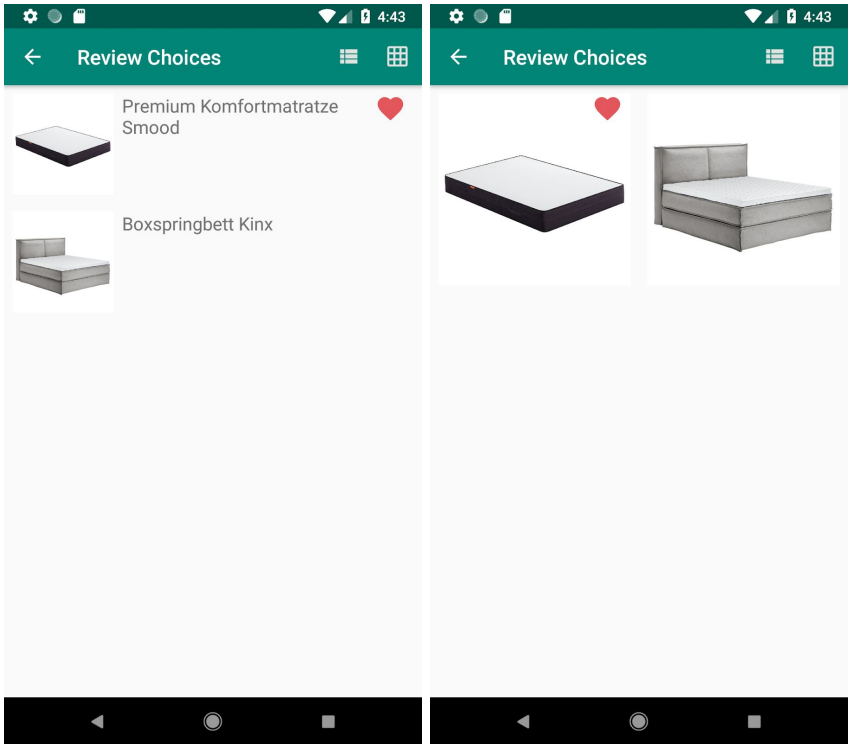
Start Screen

A full size screen which has 2 buttons to start the articles rating process, **Start button** starts a clean new process, and **Resume Last Rating** resumes an incomplete old cached process.



Select Screen

Shows a sequence of articles to the user so he/she can rate them, user can like or dislike articles in order, once he/she is done rating articles, then he/she can review his/her choices. The user can navigate back to Start Screen and continue this rating later on or start a new one.



Review Screen

Shows a list/grid of articles the user already rated including his rating.

Data Sources

There are two levels of data persistence:

- Network - Very slow.
- Disk(Room Database) - 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.
- A Cached data store layer to access the local data from database.

The chosen fetch of data is simple:

- In every get operation when user start the rating process:
    - Return local/cached copy if exists.
    - Return remote copy.

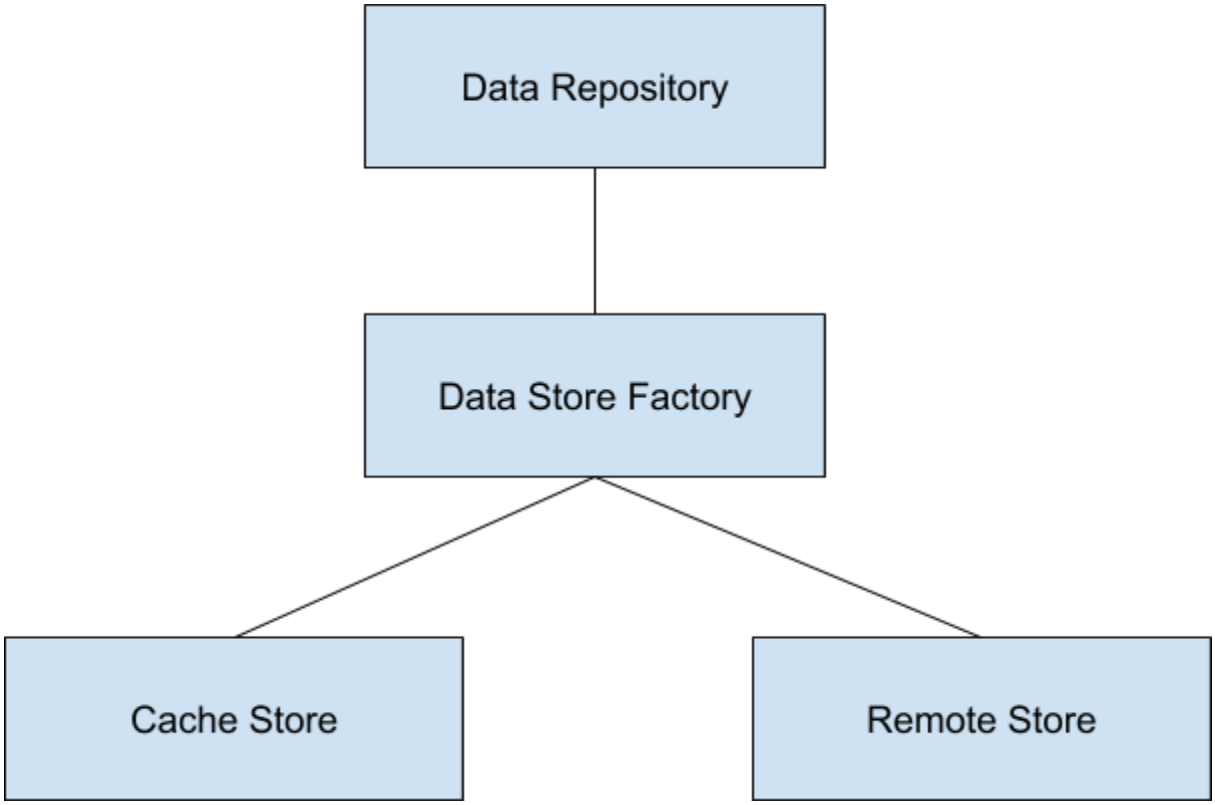
- In every like/dislike/clear operation:
    - Update local/cached copy only.

Remote data source””

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

Local data source””

The local data source uses room database to cache/add/update/delete data locally.



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.

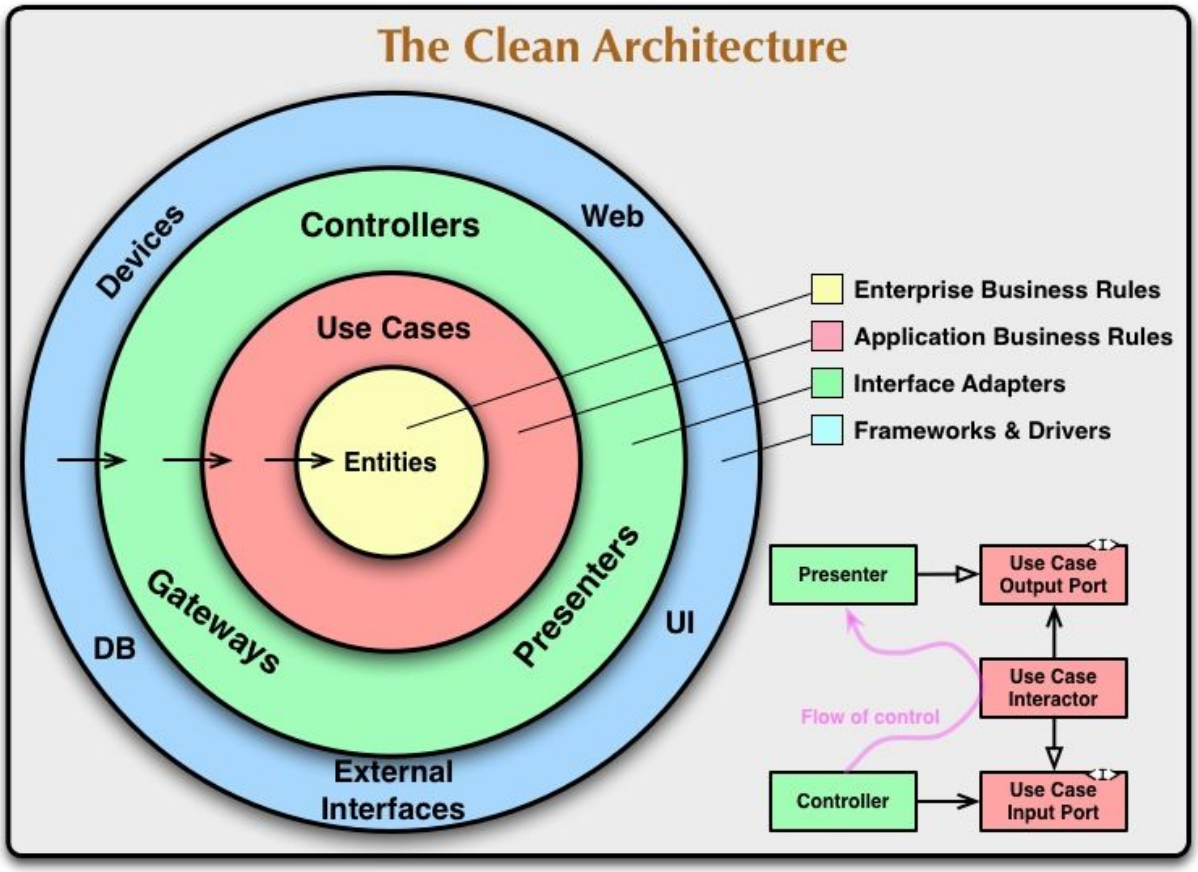
- Integration tests for testing integration between layers components and the layer itself.
- Ui tests using Espresso.

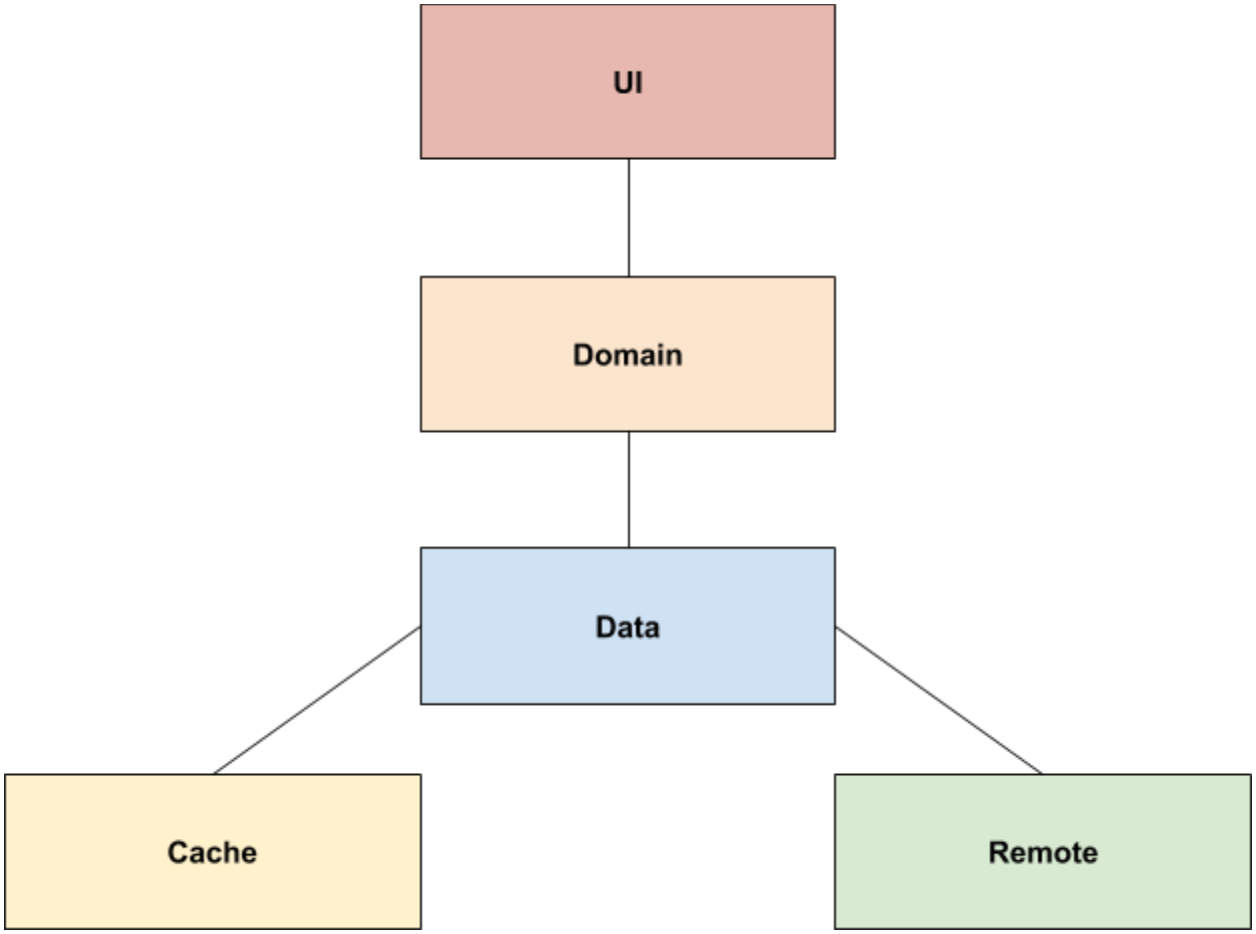
To run all tests at once, please locate the following files on Android Studio and run them:

- **AndroidTestSuite** >> to run all UI espresso tests.
- **UnitTestsSuite** >> to run all unit tests.

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.





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.