TalentCube Test Challenge

25th January 2019

REQUIREMENTS

Build an app that allows the user to answer exactly one question that is recorded, stored on the device and can be replayed.

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.

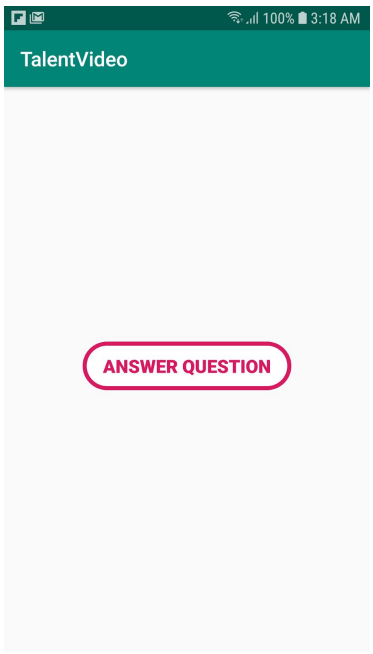
   ```
   git clone https://github.com/mfathy/TalentVideo.git
   ```

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

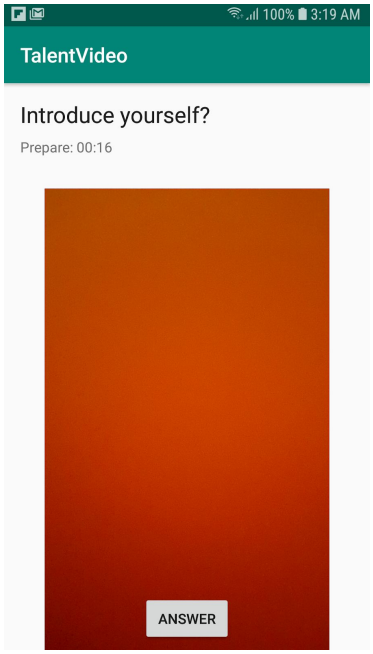
User interface



Start Screen

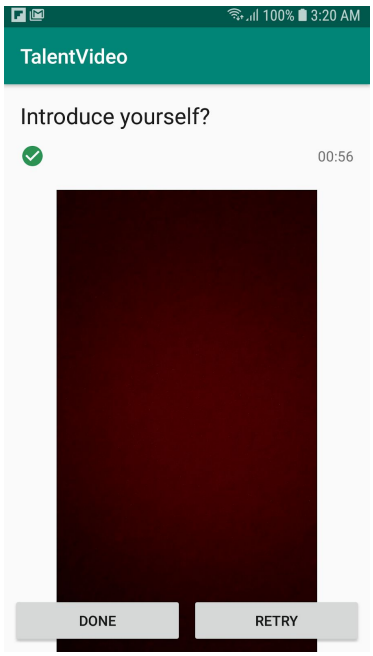
Shows the user an option to start the first step of a video interview. At this screen the app selects a random question to show to the user, so he can answer it on the next screen.

When the app randomly selects an answered question, it will show the review screen instead.



Question Screen

Shows a random question to the user. The user has only 45 seconds to prepare an answer for the question, then the recording will start automatically. Also he has another 45 seconds to answer his question. Then the app will re-direct him to the review screen.



Review Screen

Shows the video answer recorded by the user so he can review it. He has only 60 seconds to review his answer then the app will redirect him to the start screen.

Data Sources

There are two levels of data persistence:

- Fake Network “Hard-coded response”- Fast.
- 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 “Fake”.
- A Cached data store layer to access the local data from database.

The chosen fetch of data is simple:

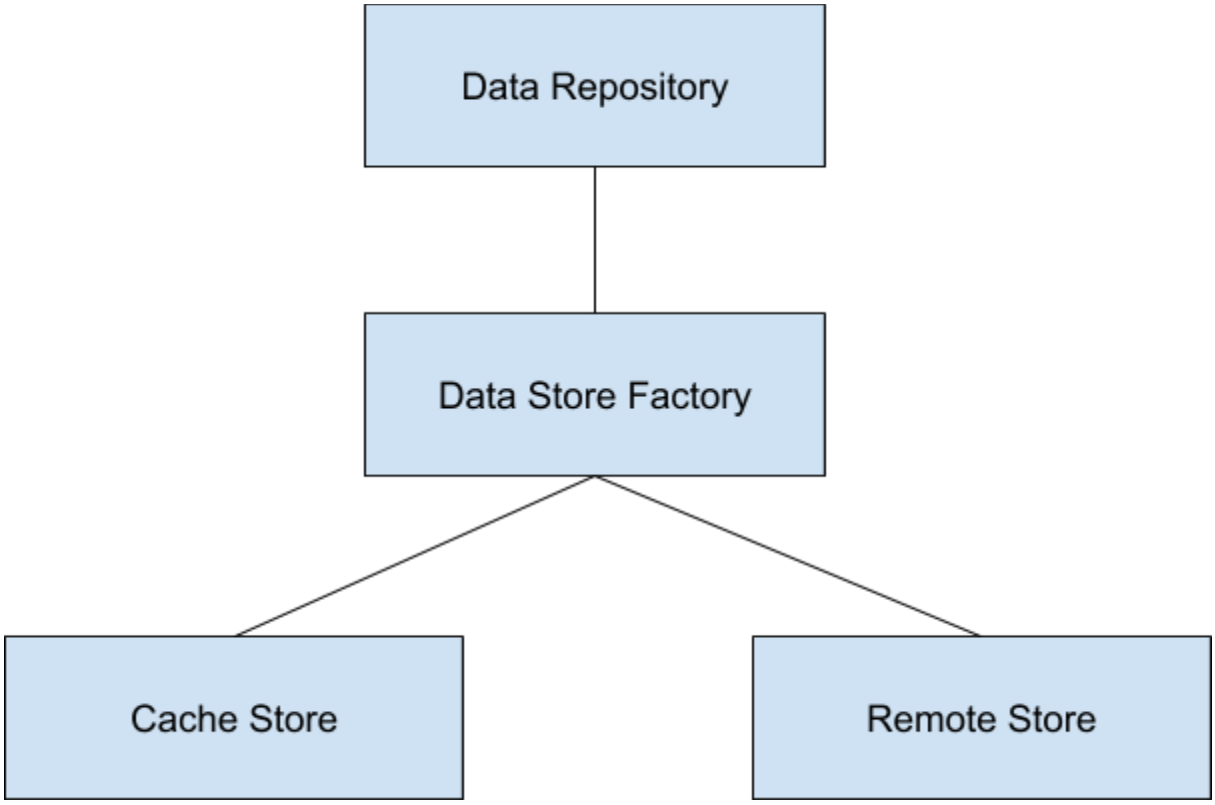
- In get question:
    - Return remote copy.
    - Return local/cached copy.

Remote data source””

The remote data source uses fake dummy helper class.

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.

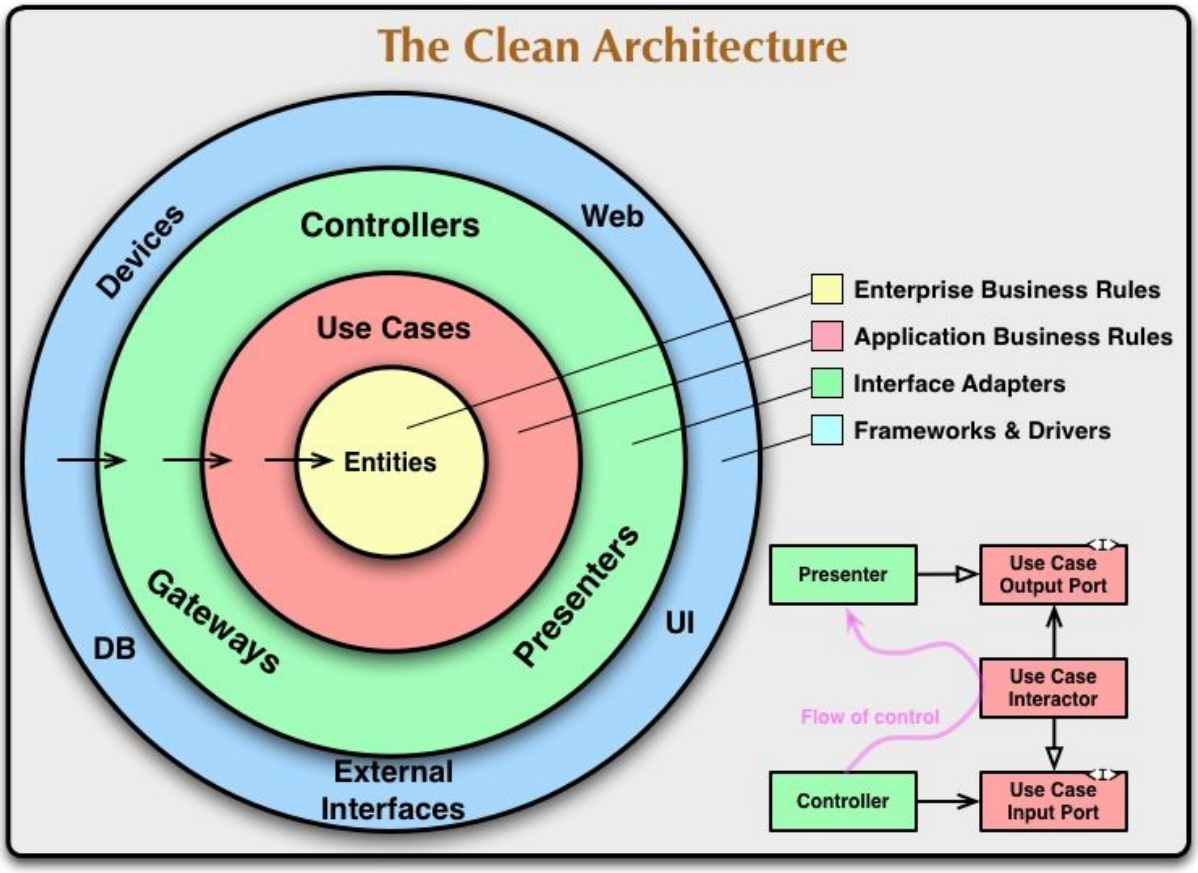
Testing

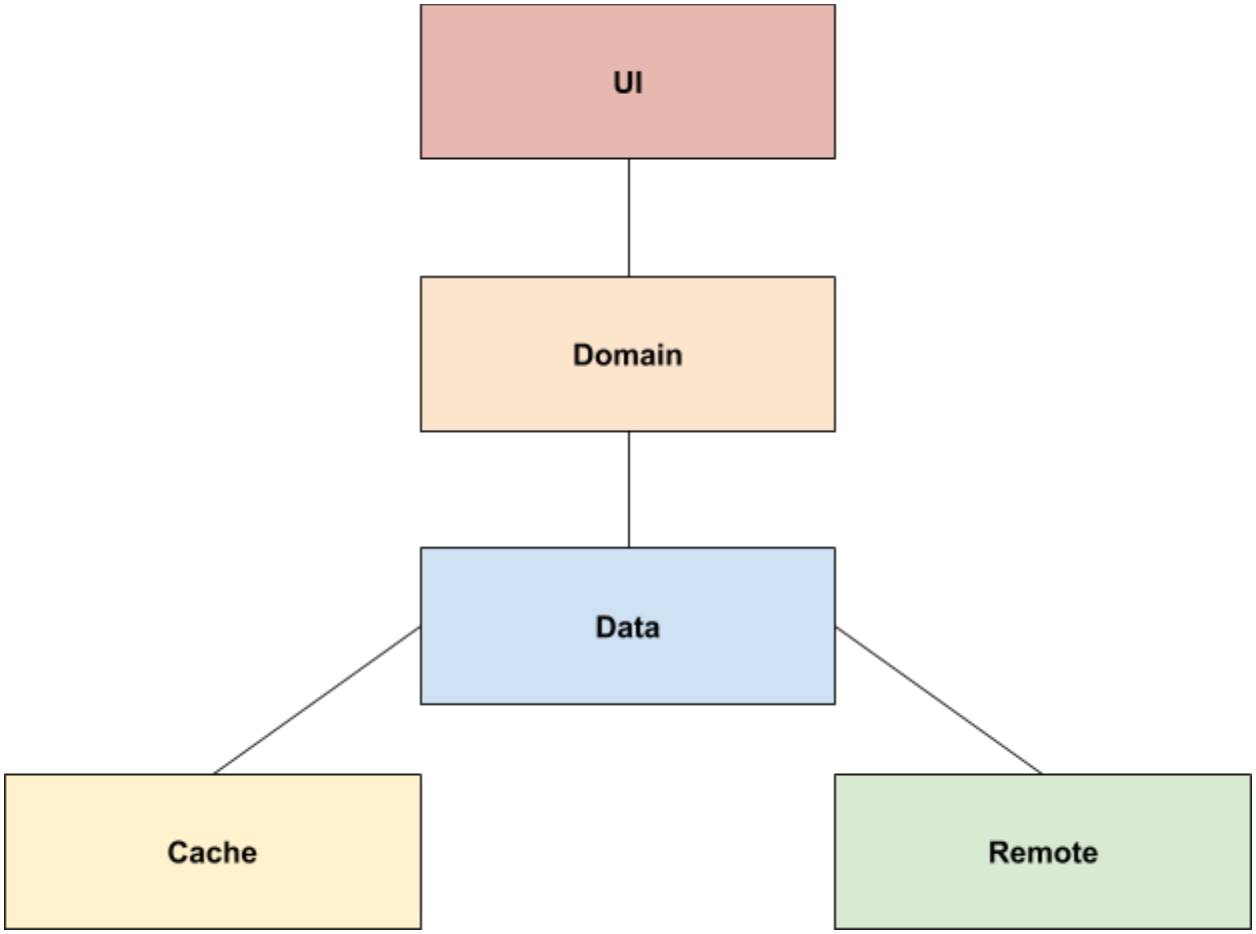
I have included the required Instrumentation, Unit and UI tests with the project:

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

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

- **Android Camera2 API**
- 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.
- 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.
- Hamcrest - Junit Matchers
- Android Architecture Components - Room, LiveData, ViewModel, Navigation.

Thank you.