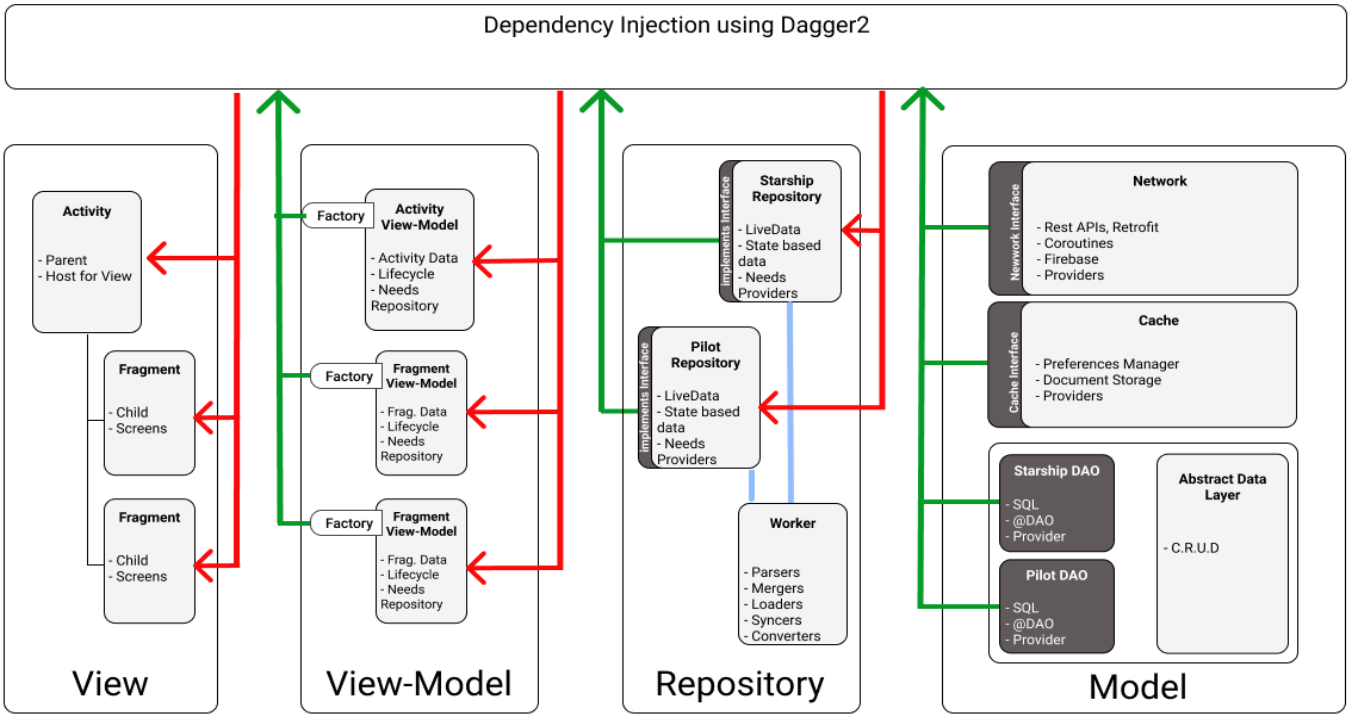
GITHUB REPO APP

Android application created using Kotlin

ARCHITECTURE

GitHub Repo app works on **M**odel-**V**iew-**V**iew**M**odel architectural pattern. It helps to cleanly separate the business and presentation logic of the application from its user interface (UI). Maintaining a clean separation between application logic and the UI helps to address numerous development issues and can make an application easier to test, maintain, and evolve.

Various other design patterns, e.g., Dependency Injection (Dagger2), Factory Pattern (ViewModeFactory), Builders (Retrofit Builder) have been used. Since GitHub Repo app accesses data from multiple sources, Repository pattern is being used. Repositories are classes or components that encapsulate the logic required to access multiple data sources.



ARCHITECTURE COMPONENTS & MORE

Android Architecture Components are a part of Android Jetpack.

As the Android Jetpack components are a collection of libraries that are individually adoptable and built to work together while taking advantage of Kotlin language features that make developers more productive.

These software components have been arranged in 4 categories in which one of the categories is Architecture Components. Other categories are Foundation Components, Behavior Components and UI Components. Some of the Components used in GitHub Repo app are:

- ViewModel: It manages UI-related data in a lifecycle-conscious way. It stores UI-related data that isn't destroyed on app rotations.
- Lifecycles: It manages activity and fragment lifecycles of the app, survives configuration changes, avoids memory leaks and easily loads data into the UI.
- LiveData: It notifies views of any database changes. Use LiveData to build data objects that notify views when the underlying database changes.
- Navigation: It handles everything needed for in-app navigation in Android application. Supports deep-linking.

Some of the other libraries used in the application are

- Dagger2: Dependency Injection framework
- Retrofit: Type-safe HTTP client for Android
- Espresso: The Espresso testing framework, provided by AndroidX Test, provides APIs for writing UI tests to simulate user interactions.
- Mockito: a popular mock framework which can be used in conjunction with JUnit.
- Kotlin Coroutines: Used for writing asynchronous code.

BUILD & RELEASE

- *run/debug Configuration*

  Android Studio uses run/debug configurations, which specifies details such as app installation, before launch, test options and can be saved for future use.

- *Build Variants/Flavors*

  Build variants can be used to configuration different target environments (*"dev", "stage", "prod"*). There are various methods available for increasing the build time when targeting to dev platforms.

- *Profiling*

  Using profiling to improve the build speed typically involves running the build with profiling enabled, making some tweaks to the build configuration, and profiling some more to observe the results of the changes.

- *Shrink, obfuscate, and optimize your app*

  Use ProGuard to perform compile-time code optimization and/or use or R8 compiler to handle compile-time tasks.

- *APK Analyzer*

  Use APK Analyzer to get immediate insight into the composition of the APK after the build process completes.

Releasing an app is multi-step process that involves multiple tasks

- Configure app for *release*
- Build and sign the release version
- Testing final release version on some common devices

Build, Test & Release Frameworks (CICD)

Continuous integration systems automatically build and test the app every time a developer check in updates to the source control system. A possible pipeline could have these:

- Jenkins Server – Builds and runs Junit tests.
- Firebase TestLab – Instrumentation Test on virtual and physical devices
- Travis CI
- Circle CI
- Team City – Automation Server by JetBrains
- AWS Device Farm

TESTING FRAMEWORK

Android provides scope for two categories of testing, namely, Unit and Integration testing.

- ### Unit Testing

  These tests should run on the local machine and should not require and physical or virtual device. Tests *ViewModel*.

  - JUnit Tests
  - Mocks, Doubles

- ### Integration Testing

  Android Instrumentation Test use Android framework libraries to communicate with the physical/virtual devices which in turn validate the business logic. Tests ViewModel by instrumenting Android UI and mocking data sources.

  - Espresso
  - UIAutomator
  - Firebase TestLab
  - AWS Device Farm
  - Mockito