**Readme**

Author: Phong Ho

Since: 08/01/2021

1. **Diagrams and models**

**Architecture overview:** Apply MVVM architect

Graphical user interface, diagram

Description automatically generated

**Appflow overview**

Diagram

Description automatically generated

**Models**

Diagram

Description automatically generated

At the first time, I want to make an weather application like Applition on Android Smartphone, but currently I do not have enough time for it, so only make UI like requirement.

A picture containing text, black, close

Description automatically generated

1. **Principle applied**

**Seperation of Concern principles**: apply when using MVVM architecture, include 3 layers: View, Viewmodel, Model. Each layer is responsible for one task.

**SOLID principles**:

*S - Single responsibility*: Each class or method func only have one job to to.

Example:

Class ApiManager: only init and provides Api

Class DataPaser: only parse response data from server

*O – Open-closed*: open for extension, but close for modifycation

Example: Apply when design repository layer to get remote data.

Currently, we only using WeatherApi to get data, the next time, we need extend some feature and need get data from another api. We only do some steps:

1. Define method for new API
2. Declare new API instance in ApiManager class
3. Using ApiManager to get new Api instance and call its method.

🡺 We don’t modify any class except ApiManager – this class manage all Api so that we need modify to add new Instance in this class

1. **Design Patterns & Practices applied**

Singleton design pattern: WeatherRepository class

Builder desgin Pattern: Service Builder class

Strategy Pattern: DataResponse class -> WeatherRepository class

1. **Folder structure**

|  |  |
| --- | --- |
| Folder | Responsibility |
| Api | Manage (API Manager),  create (ServiceBuilder),  define Api method (WeatherApi) |
| Database | Init, migrate database (WeatherDB)  Defind method access DB (WeatherDao) |
| Model | Define Model data |
| Repository | Manage, handle logic get data (local, remote) |
| Utils | Include class can reuse anywhere in project: Errorcode, StringUtils, etc |
| View | Include activty, adapter, v.v |
| Viewmodel | Include viewmodel class handle logic app |

1. **Libraries & android componet**:
   1. Gson, Retrofit2, OkhttpClient, Mockito
   2. Viewmodel, live data, room database
2. **Run app on local computer**:

**Run debug mode:**

1. Check out source code by android studio
2. Select build variant on debug mode
3. Run build

**Run release mode:**

1. Build -> Generate Signed Bundle/APK
2. Choose opt APK -> Next
3. On Key store path: choose existing -> select file release.keystore (../WeatherApp/app) -> Next
4. Choose destination foler generate apk -> release -> V2 -> Finish
5. Get apk and run adb: adb install app-release.apk

**Configure the build process to automatically sign your app: sorry for not config success ^^**

1. **Check list item done:**

Done: 9/12

|  |  |  |  |
| --- | --- | --- | --- |
| STT | Description | Status | Note |
| 1 | Programming language: Kotlin is required, Java is optional | Badge Tick1 outline |  |
| 2 | Design app's architecture (suggest MVVM) | Badge Tick1 outline |  |
| 3 | Apply LiveData mechanism | Badge Tick1 outline |  |
| 4 | UI should be looks like in attachment. | Badge Tick1 outline |  |
| 5 | Write UnitTests | Badge Tick1 outline | Unit test for database, livedata (folder android test) |
| 6 | Acceptance Tests | **Badge Cross outline** |  |
| 7 | Exception handling | Badge Tick1 outline | Handle error when get data from api (only simple a toast on UI) |
| 8 | Caching handling | **Badge Cross outline** |  |
| 9 | Secure Android app from | Badge Tick1 outline | Using: proguard, minifyEnabled, shrinkResource for release build |
| 10 | Accessibility for Disability Supports | **Badge Cross outline** |  |
| 11 | Entity relationship diagram for the database and solution diagrams for the components, infrastructure design if any | Badge Tick1 outline |  |
| 12 | Readme file includes | Badge Tick1 outline |  |