Welcome!

This guide explains how the android smart phone app,

ComposeMVVMSample, was designed, how it is expected to be used, by android smart phone app developers, such as yourself and how it works under the covers.

This app was designed to help anyone use and understand how jetpack compose and mvvm apps are supposed to be created.

It is expected that, after renaming the sample, you will have the start of a well designed app.

ComposeMVVMSample has functionality and structure to help with common issues that all projects need, such as:

- communication between the user interface and the data being displayed, in such a way as to not do long running operations on the ui thread, which would lock up the app.
- Logging and other custom operations, via extension functions, **Extensions.kt**
- coroutine management
- Prefs.kt class to handle persisting data easily, when a database is not needed.
- Helpful, custom composeables **ui\composeables\core**
- repo repository to handle long running operations, such as fetching data from the web, other apps or just long running operations.

How to Get it

- Open up Android Studio
- Make sure all projects are closed.
- In upper right, it should say clone repository, press that.
- Enter **Repository directory** where you want new project on your hard drive.
- For **Repository URL** enter: ***https://github.com/pstorli/ComposeMVVMSample.git***
- Press the **blue clone button**.

How to Rename it

So you now have a project named **ComposeMVVMSample**.

But the whole point of this project is that instead of pressing New Project in Android Studio you now get to simply pull this one down, anytime you need to start a new project, and use it as your starting point.

But to truly make it your own, we need to **rename it** and also get it **checked into github** safely as your very own new project.

Go above directory you just cloned, **ComposeMVVMSample** and rename it to your new project name, I'll call it **MySample**

- Go into your new project, delete the **MySample\**.git directory.
- Open up new project in Android Studio, MySample.
- Rename **dirs com.pstorli.composemvvmsample** to **com.mycompany.mysample**, or whatever you want.

  a) Goto dir **MySample\\app\\src\\main\\java\\com**

  b) Rename **pstorli** dir to **mycompany**

  c) Rename **composemvvmsample** dir to **mysample**

- **Open up your new project in Android Studio.**

How it works

So you now have a project named **ComposeMVVMSample.**

But to truly make it your own, we need to rename it and also get it checked into github safely as your very own new project.

Why would you want to do that?

Well, this little, simple sample, demonstrates how to use mvvm and jetpack compose together.

It shows how your user interface, jetpack compose, and repository honor the ViewModel, as the place to update and store data necessary for various screens.

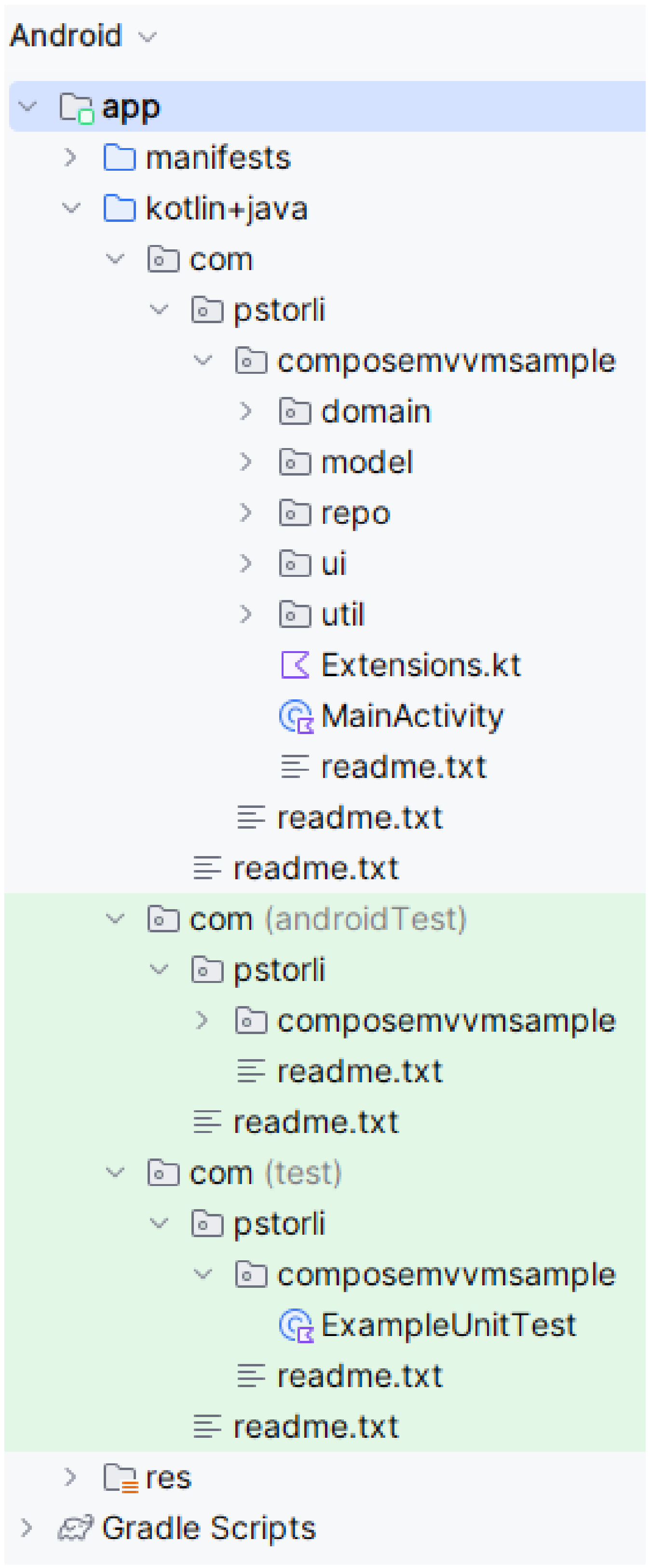
NOTE:

- App execution starts at

  MainActivity.onCreate, **MainActivity.kt**, also holds the viewModel, which holds project data.

  val viewModel: ViewModel by viewModels()

- *Project root:*
  ComposeMVVMSample\app\src\main\java
  app\kotlin+java\com\pstorli\composemvvmsample



```
Android
  app
    manifests
    kotlin+java
      com
        pstorli
          composemvvmsample
            domain
            model
            repo
            ui
            util
            Extensions.kt
            MainActivity
            readme.txt
          readme.txt
        readme.txt
      com (androidTest)
        pstorli
          composemvvmsample
          readme.txt
        readme.txt
      com (test)
        pstorli
          composemvvmsample
            ExampleUnitTest
          readme.txt
        readme.txt
    res
  Gradle Scripts
```

- **domain package**

  Used to hold custom project data

- **model package:**

  handles data src and ui / viewmodel communication

  ViewModel.kt has **viewmodel** which is data source, holds **repo** and **prefs**

  VMHelper.kt holds viewmodel logic.

  CoHelper.kt handles viewmodel /repo communication, via coroutines, used to perform long running operations, that would otherwise lock up the ui thread.

- **repo package**

  Repo.kt for long running functions, done **not on ui thread**!

- **ui package** holds custom, generic, composeable classes core and project screens\StartScreen.kt

  screens holds project's compose screens. sample start screen, **StartScreen.kt** and

  theme\Theme.kt

- **util package holds utility classes**

  Prefs.kt class used for persisting data, when a database is overkill.

  Consts.kt class used for project wide constants.