# Jetpack Compose Modular Clean Architecture with Rorty App

Mesut G. · Follow

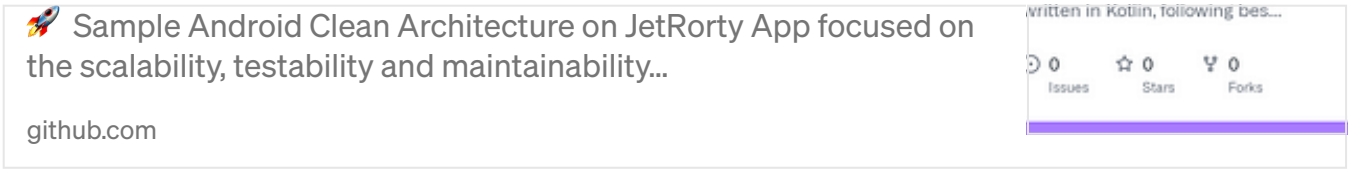
8 min read · Feb 24, 2022

Listen Share More

If you want to check directly the project before continuing reading the introduction, you can do by accessing the following link:

**GitHub - developersancho/JetRorty.Android: 🚀 Sample Android Clean Architecture on JetRorty App...**

ersancho/
.Android
roid Clean Architecture on JetRorty
the scalability, testability and





## Overview

Jetpack Compose is Android's modern toolkit for building native UI. It simplifies and accelerates UI development on Android. Quickly bring your app to life with less code, powerful tools, and intuitive Kotlin APIs. It makes building Android UI faster and easier.

Android isn't the only supported platform. For start, you can use it in **Web** and **Desktop** — **Multiplatform** apps.

## Advantages of Jetpack Compose(Pros)

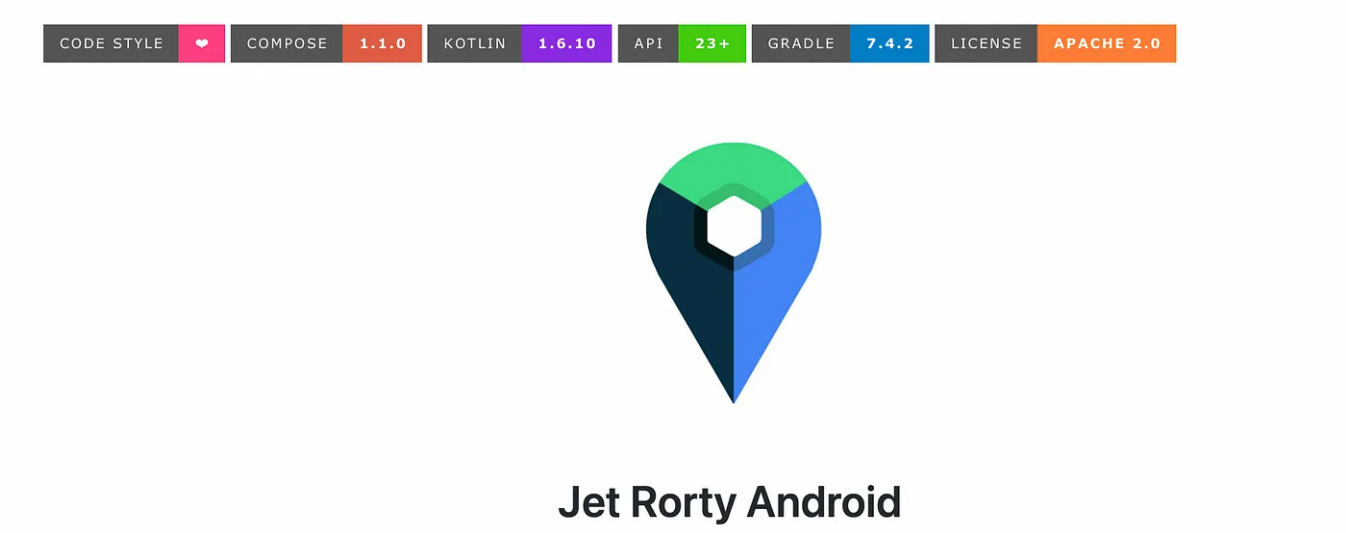
- Writing less code affects all stages of development.
- Compose uses a declarative API, which means that all you need to do is describe your UI.
- Compose is compatible with all your existing code.
- It is easy to update and easy to test.
- It is easily compatible with the existing views present in Android.
- It increase the development speed.
- It removes the boilerplate of findViewById or ViewBinding references.

## Disadvantages of Jetpack Compose(Cons)

- The downside of this solution is re-rendering preview each time code will change and builds are not that fast.
- It's little slower than xml to render a change.(*Try to build release build with turn off debug logs and should be works fine* 😊 )
- Some components are not supported, and some features are is coming. You can take a look **roadmap**

learned so far and to improve myself.

**JetRorty** focused on the scalability, testability and maintainability written in Kotlin, following best practices using Jetpack Compose.




## What Did I Develop

The project presents a modern, approach to Android application development using Kotlin and latest tech-stack with Jetpack Compose.

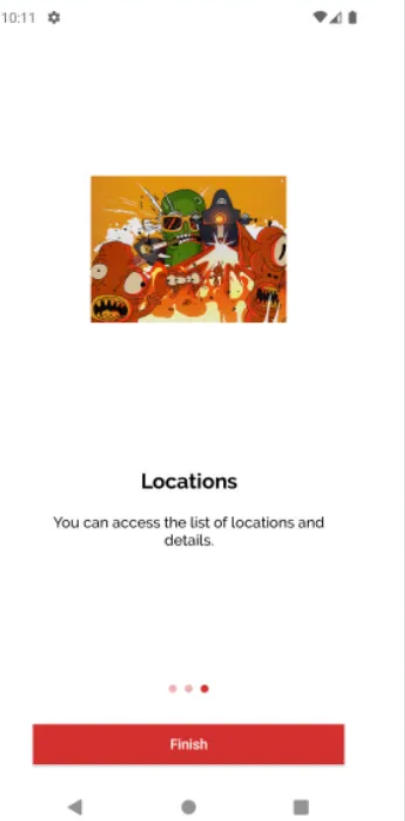
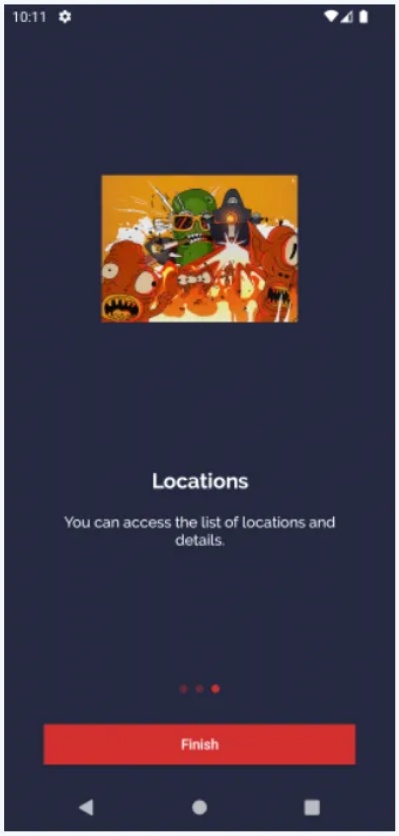
The goal of the project is to demonstrate best practices, provide a set of guidelines, modular application, scalable, maintainable and testable. This application may look simple, but it has all of these small details that will set the rock-solid foundation of the larger app suitable for bigger teams and long application lifecycle management.

For this project I used the Clean Modular Architecture. The MVP Features:

- Splash
- OnBoarding-Intro
- Home
- Characters List
- Characters Favorite List
- Character Detail

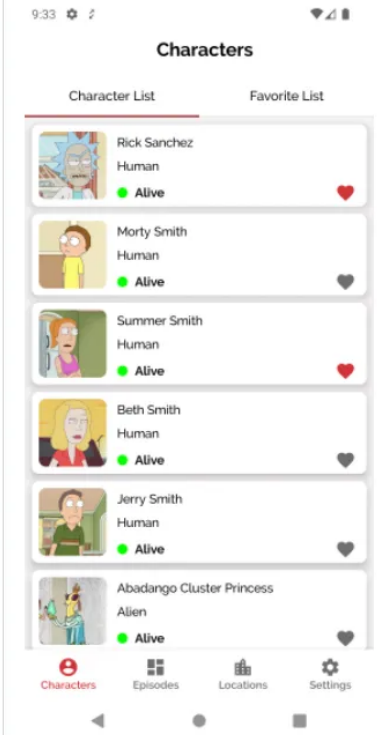
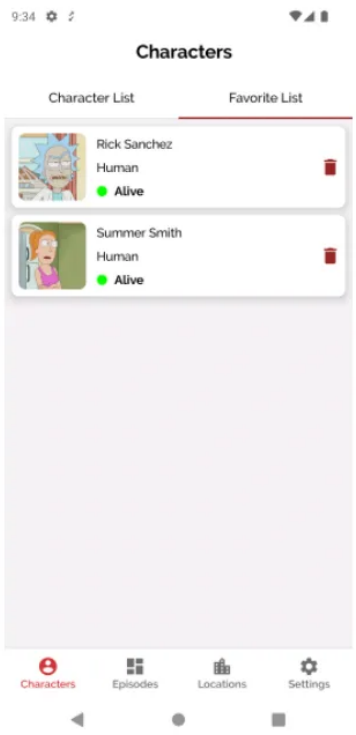
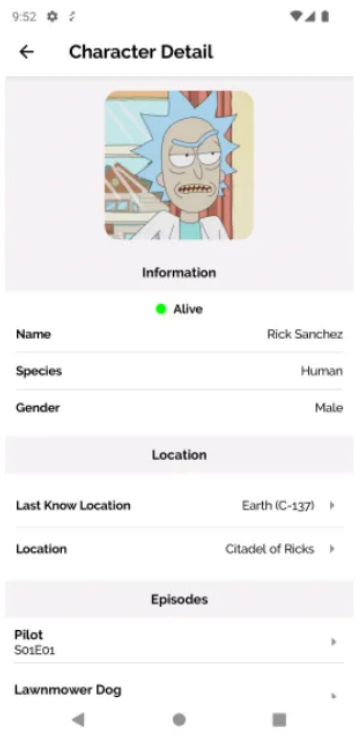
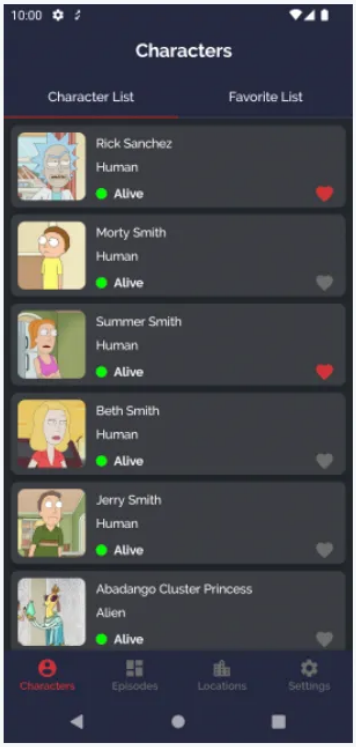
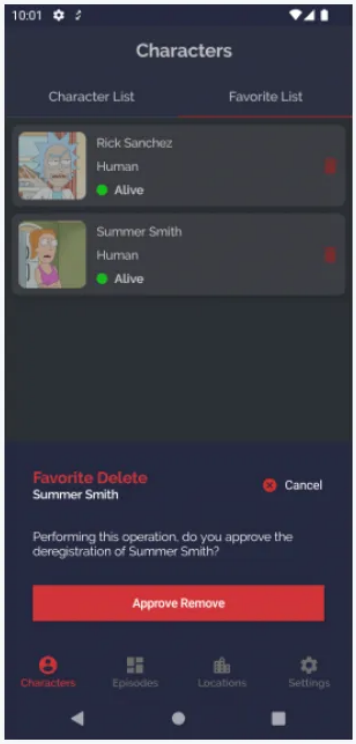
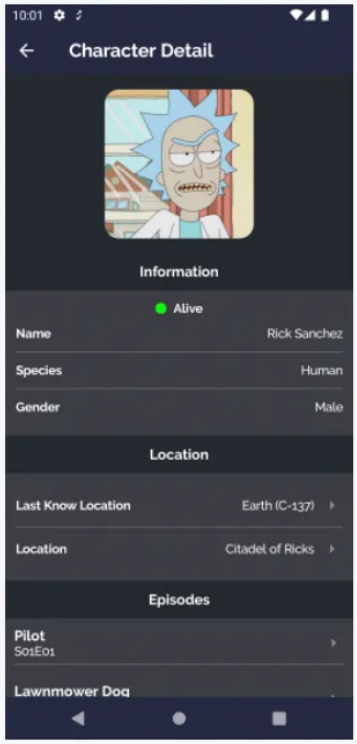
- Episodes List
- Episodes Favorite List
- Episodes Detail
- Locations List
- Locations Favorite List
- Locations Detail
- Settings
- App Language
- About

## Screenshots

| Mode | Splash | OnBoarding - Intro |
| --- | --- | --- |
| Light |  |  |
| Dark |  |  |

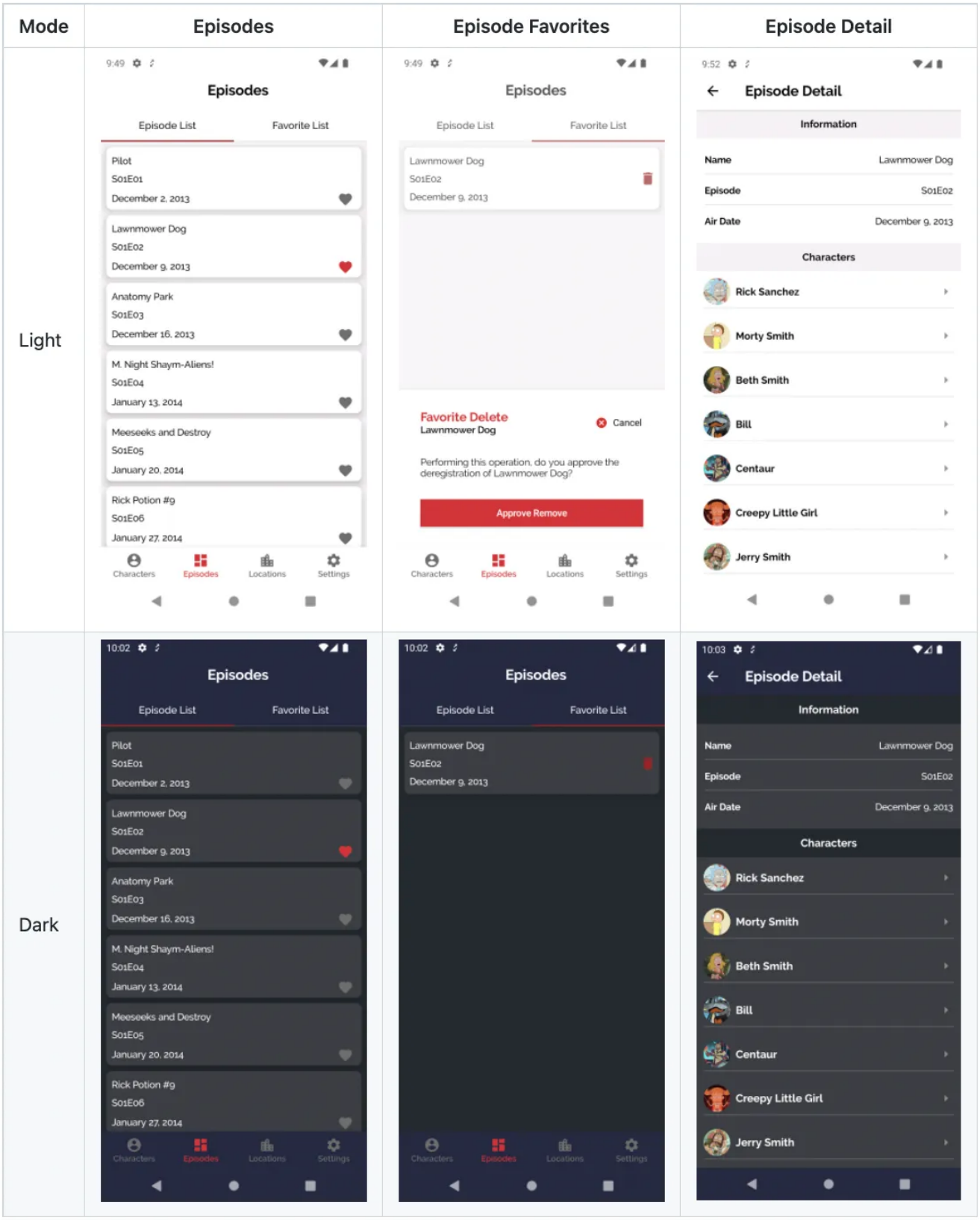


Splash and Intro

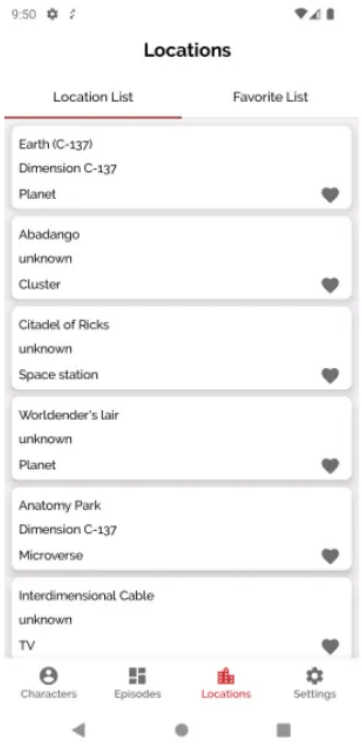
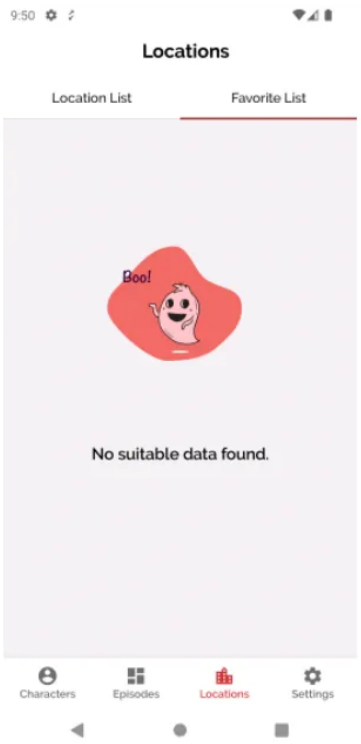
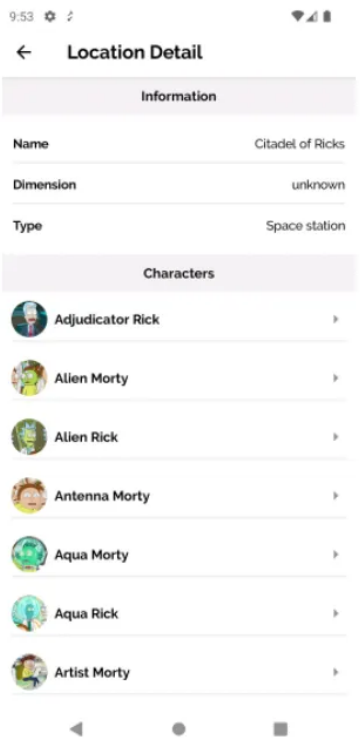
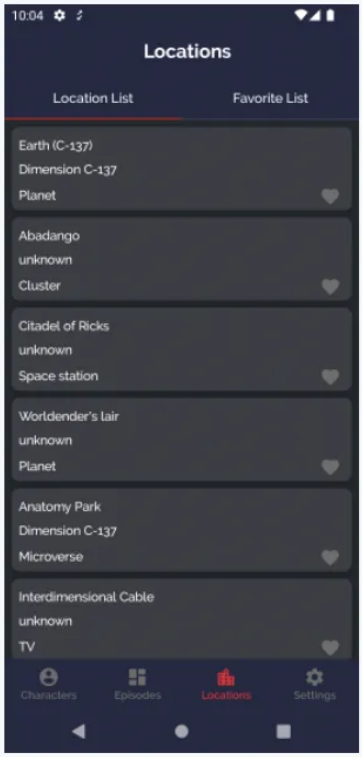
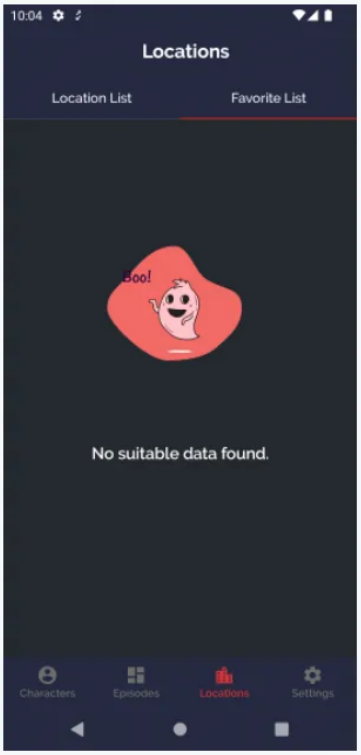
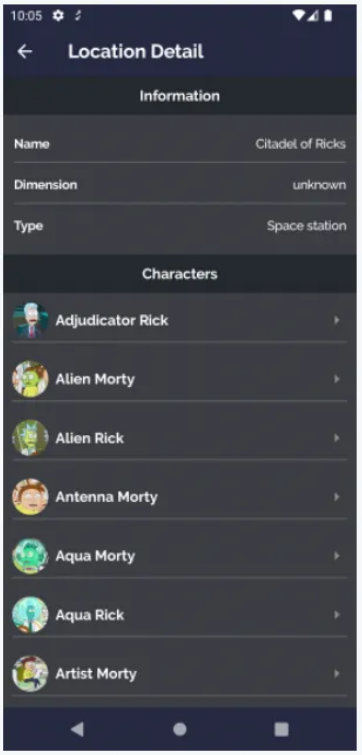
| Mode | Characters | Character Favorites | Character Detail |
| --- | --- | --- | --- |
| Light |  |  |  |
| Dark |  |  |  |



Characters

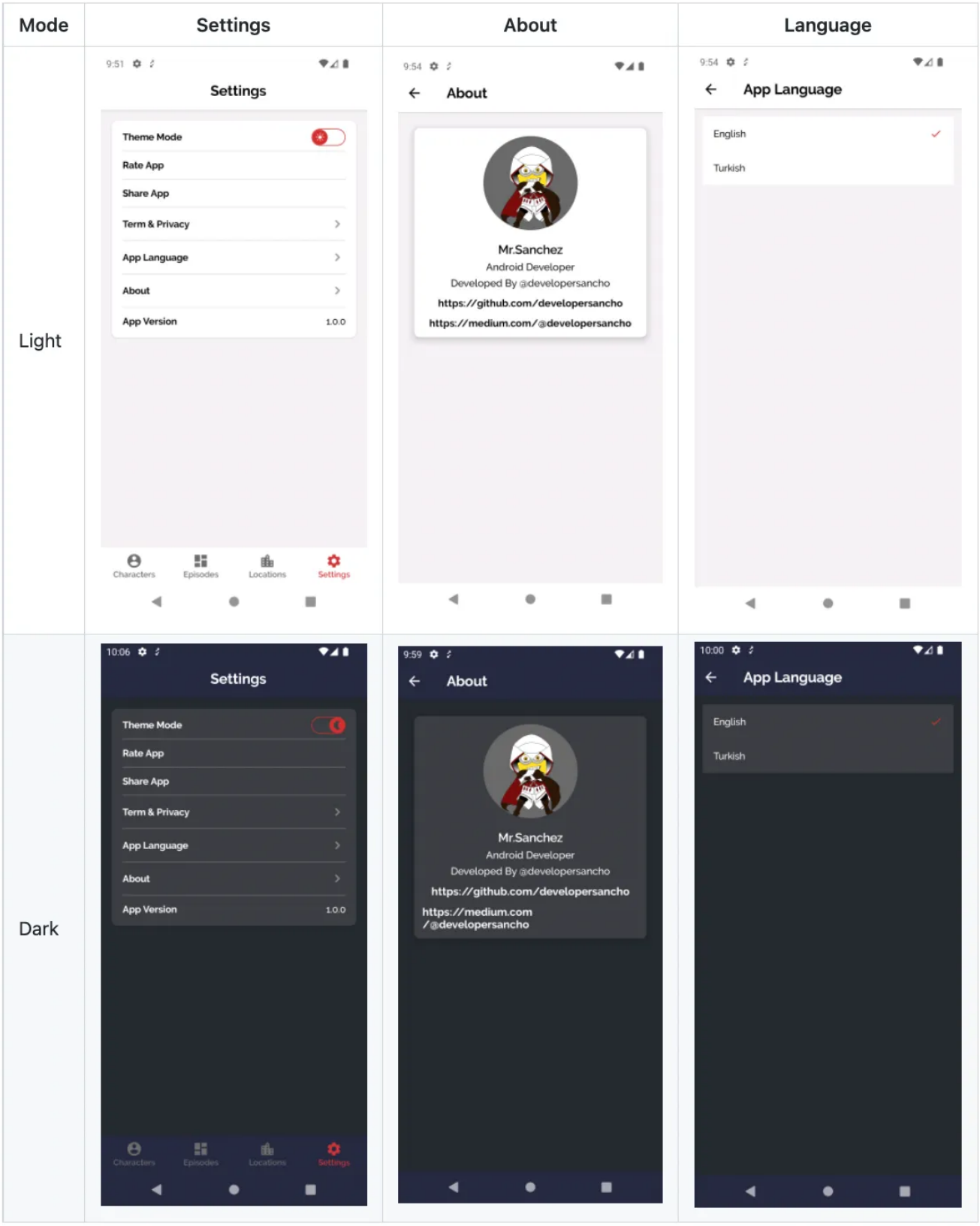

Mode
Episodes
Episode Favorites
Episode Detail
Light
Dark
Episode List
Favorite List
Pilot
S01E01
December 2, 2013
Lawnmower Dog
S01E02
December 9, 2013
Anatomy Park
S01E03
December 16, 2013
M. Night Shaym-Aliens!
S01E04
January 13, 2014
Meeseeks and Destroy
S01E05
January 20, 2014
Rick Potion #9
S01E06
January 27, 2014
Characters
Episodes
Locations
Settings
Favorite Delete
Lawnmower Dog
Cancel
Performing this operation, do you approve the deregistration of Lawnmower Dog?
Approve Remove
Episode Detail
Information
Name
Episode
Air Date
Characters
Rick Sanchez
Morty Smith
Beth Smith
Bill
Centaur
Creepy Little Girl
Jerry Smith

Episodes

| Mode | Locations | Location Favorites | Location Detail |
| --- | --- | --- | --- |
| Light |  |  |  |
| Dark |  |  |  |



Locations



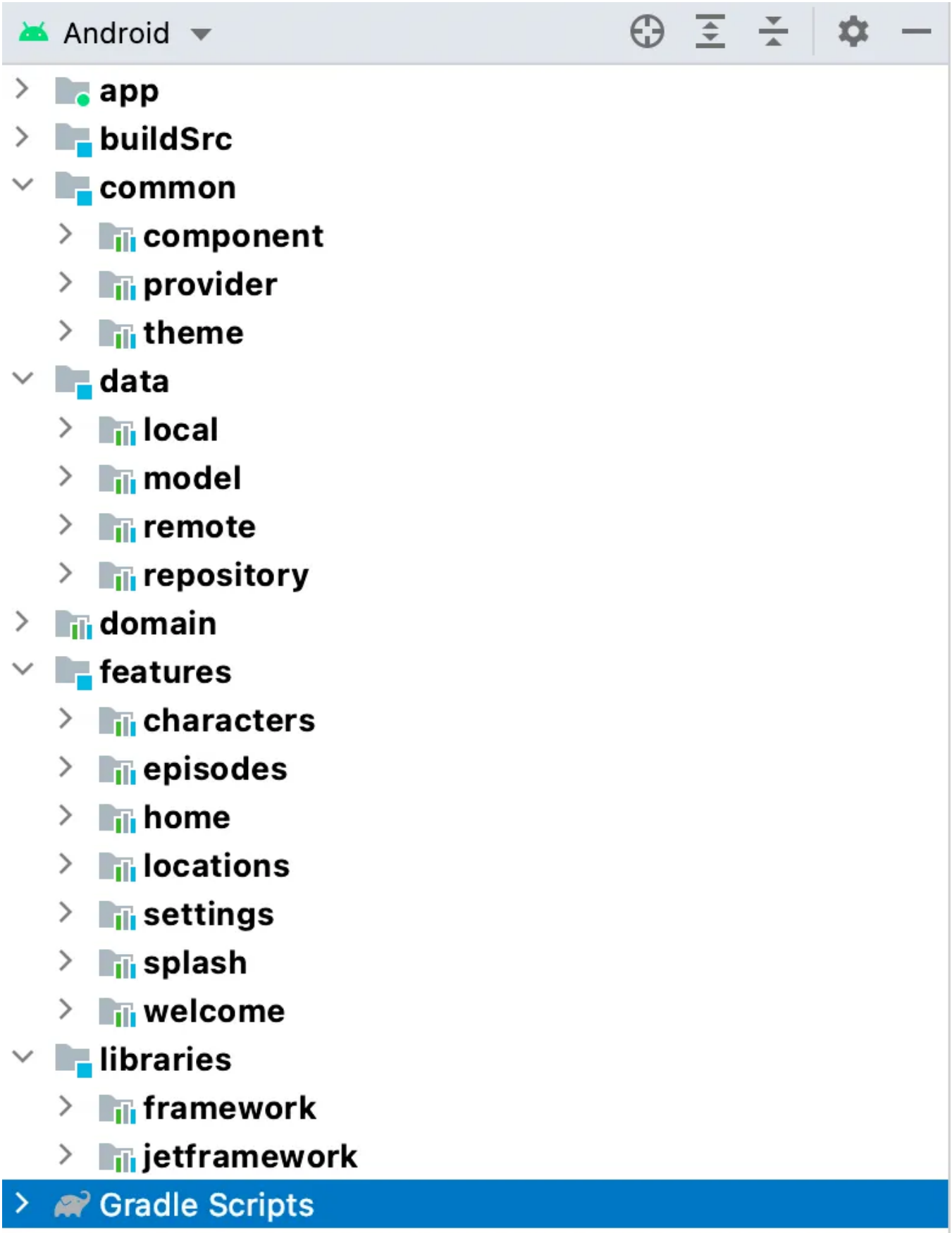

Settings

## Environment Setup

In order to be able to build the application you'll need Android Studio Minimum version `Bumblebee (2021.1.1)`

## Application structure

One of the key benefits of modularization architecture is supposed to be clear navigation throughout the app and source code. Looking at the root folder of the project, the following structure becomes clear.




**- App module**

The `:app` module is a com.android.application, which is needed to create the App Bundle.

### - Features module

The `:features` module is a com.android.library, which is needed to create the features module. It's android specific and contains compose pages, view models, activities, and so on. It also contains a service locator to manage dependencies, but you can use Dagger Hilt if you prefer. I used it.

### - Common module

The `:common` module contains the implementation of the common structures.

- The `:common:component` module contains the common view components.
- The `:common:provider` module contains the definitions of theme, resource and navigation provider interfaces.
- The `:common:theme` module contains the theme, color, font type and resource files.

### - BuildSrc module

The `:buildSrc` module is responsible for dependency management. It control and manage all dependencies in one place with Kotlin.

### - Data module

The `:data` module contains the implementation of the abstract definitions of the domain layer. Can be reused by any application without modifications. It contains repositories and data sources implementations, the database definition and its DAOs, the network APIs definitions, the definition of repositories, some mappers to convert network API models to database models, and vice versa.

- The `:data:model` module contains the definitions of ui model, network response and entity classes.
- The `:data:local` module contains the definitions dao, database, caching operations using Room.
- The `:data:remote` module contains the network operations defining API endpoints using Retrofit.
- The `:data:repository` module contains the exposing data to the domain layer

### - Domain module

The `:domain` module contains the definitions of the business logic of the app, some mappers for use in ui, and the definition of the use cases. This is the core layer of the application. The `domain` layer is independent of any other layers domain business logic can be independent from other layers.This means that changes in other layers will have no effect on domain layer eg. screen UI (presentation layer) or changing database (data layer) will not result in any code change within domain layer.

Components of domain layer include:

- `usecase`: They enclose a single action, like getting data from a database or posting to a service. They use the repositories to resolve the action they are supposed to do. They usually override the operator `invoke` , so they can be called as a function.

### - libraries module

- The `:libraries:framework` module contains different utilities that can be used by the different modules and base structures. Also the definitions of some extension methods.
- The `:libraries:jetframework` module contains different utilities that can be used by the different modules and base structures only for Jetpack Compose. Also the definitions of some extension methods.
- The `:libraries:testing` module contains the definitions of the test utilities.

### Configuration Plugins

With App Modularization we want to gain fine-grained dependency control but we also need to make sure we don't end up maintaining multiple configuration plugins.

For that we have the following common configuration custom plugin classes:

- **AndroidCoreLibraryPlugin**
- **android-feature.gradle.kts**
- **android-library.gradle.kts**
- **android-compose.gradle.kts**

## UnitTest

Unit tests are the fastest and easiest tests to write and the quickest to run. We write unit tests if we want to ensure that a class or method is working as intended.

I tried to apply unit test methods in `data`, `domain` and `presentation` layers of JetRorty application.

## Tech Stacks

This project uses many of the popular libraries, plugins and tools of the android ecosystem.

## - Dependencies

### Compose

- **Material** — Build Jetpack Compose UIs with ready to use Material Design Components.
- **Foundation** — Write Jetpack Compose applications with ready to use building blocks and extend foundation to build your own design system pieces.
- **UI** — Fundamental components of compose UI needed to interact with the device, including layout, drawing, and input.
- **ConstraintLayout** — ConstraintLayout-compose 1.0 provides ConstraintLayout functionalities in Jetpack Compose.
- **Lifecycle-ViewModel** — Perform actions in response to a change in the lifecycle status of another component, such as activities and fragments.
- **Paging** — The Paging Library makes it easier for you to load data gradually and gracefully within your app's RecyclerView.
- **Lottie** — Lottie is a mobile library for Android and iOS that parses Adobe After Effects animations exported as json with Bodymovin and renders them natively on mobile!
- **Coil** — An image loading library for Android backed by Kotlin Coroutines.
- **Navigation** — Annotation processing library for type-safe Jetpack Compose navigation with no boilerplate.

### Accompanist

- **SwipeRefresh** — A library which provides a layout which provides the swipe-to-refresh UX pattern, similar to Android's SwipeRefreshLayout.
- **Systemuicontroller** — System UI Controller provides easy-to-use utilities for updating the System UI bar colors within Jetpack Compose.
- **Insets** — Insets for Jetpack Compose takes a lot of the ideas which drove Insetter for views, and applies them for use in composables.
- **Placeholder** — A library which provides a modifier for display 'placeholder' UI while content is loading.
- **Navigation** — A library which provides Compose Material support for Jetpack Navigation Compose. This features composable bottom sheet destinations.

**Jetpack**

- **Android KTX** — Provide concise, idiomatic Kotlin to Jetpack and Android platform APIs.
- **AndroidX** — Major improvement to the original Android Support Library, which is no longer maintained.
- **Lifecycle** — Perform actions in response to a change in the lifecycle status of another component, such as activities and fragments.
- **ViewModel** — Designed to store and manage UI-related data in a lifecycle conscious way. The ViewModel class allows data to survive configuration changes such as screen rotations.
- **Room** — Provides an abstraction layer over SQLite used for offline data caching.
- **Paging3** — The Paging Library makes it easier for you to load data gradually.
- **Dagger Hilt** — Dependency Injection library.
- **Google-KSP** — Kotlin Symbol Processing API
- **Retrofit** — Type-safe http client and supports coroutines out of the box.
- **OkHttp-Logging-Interceptor** — Logs HTTP request and response data.
- **Coroutines** — Library Support for coroutines.

- **Flow** — Flows are built on top of coroutines and can provide multiple values. A flow is conceptually a stream of data that can be computed asynchronously.
- **Material Design** — Build awesome beautiful UIs.
- **Coroutines** — Library Support for coroutines,provides runBlocking coroutine builder used in tests.
- **Timber** — A logger with a small, extensible API which provides utility on top of Android's normal Log class.
- **Moshi** — A modern JSON library for Kotlin and Java.
- **Chucker** — An HTTP inspector for Android & OkHTTP (like Charles but on device).
- **Gradle Kotlin DSL** — makes it easy to manage dependency all module that we have.
- **SplashScreen** — Android 12 adds the SplashScreen API, which enables a new app launch animation for all apps when running on a device with Android 12 or higher.

### - Test Dependencies

- **JUnit** — a simple framework to write repeatable tests. It is an instance of the xUnit architecture for unit testing frameworks.
- **Mockk** — provides DSL to mock behavior. Built from zero to fit Kotlin language.
- **AndroidX** — the androidx test library provides an extensive framework for testing Android apps.
- **Robolectric** — industry-standard unit testing framework for Android.
- **Turbine** — a small testing library for kotlinx.coroutines `Flow`.
- **MockWebServer** — a scriptable web server for testing HTTP clients.
- **Coroutines** — provides testing utilities for effectively testing coroutines.

### - Code Analyze Plugins

- **Ktlint** — an anti-bikeshedding Kotlin linter with built-in formatter.

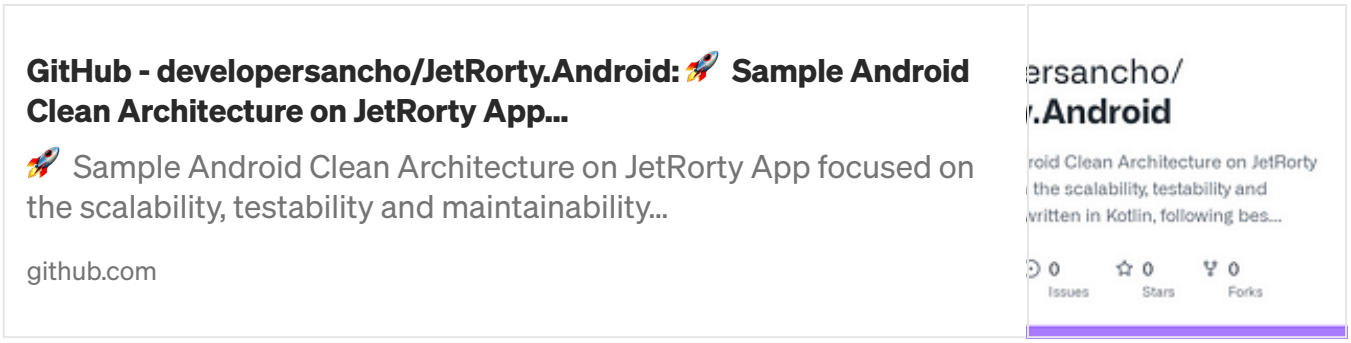
- **Detekt** — a static code analysis tool for the Kotlin programming language.
- **Spotless** — a code formatter can do more than just find formatting errors.
- **Versions** — make easy to determine which dependencies have updates.

## How Can You Contribute?

- **Open issues** with suggestion of better approaches or ideas for the app.
- Connect with me on **Linkedin**.
- Star the **Github** repository.
- Follow me on **Github**.

## Full Project

You can access the source code of the project from the link below.




## Thanks 🚀

I hope it helps you. If it is useful to you, you can clap this article and follow me for such these articles.

Happy and healthy coding!

**Que tengas un buen día, continuará 😊 🏄 🚀**

Jetpack Compose | Modular Architecture | Clean Architecture | Android | Kotlin

Follow

## Written by Mesut G.

243 Followers

Hola everybody, Sharing Software Development Experience, focus on Mobile.
https://developersancho.medium.com/subscribe Continue as long as

### More from Mesut G.

Mesut G.

## Jetpack Compose: Splash Screen API

Migrate your existing splash screen implementation to Android 12

5 min read · Mar 8, 2022

103 3

Mesut G. in Dev Genius

# Jetpack Compose Clean Architecture with Rorty App

If you want to check directly the project before continuing reading the introduction, you can do by accessing the following link:

8 min read · Apr 15, 2022



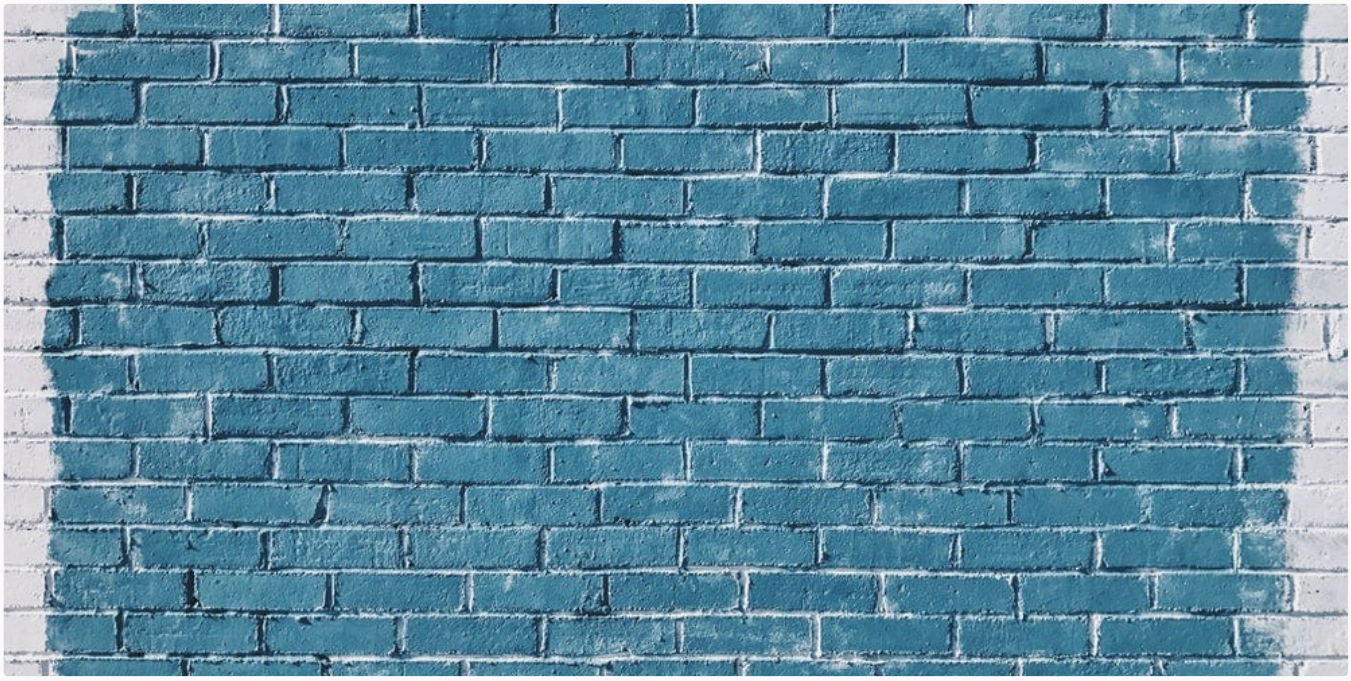
Mesut G.

## CI-CD: Firebase App Distribution with Fastlane on Android

Distribute your apps on Android with Firebase App Distribution.

5 min read · Apr 28, 2022

110 1



Mesut G. in Innovance Blog

## Jetpack Compose: Migration to Gradle Kotlin DSL

Let's explain using gradle kotlin dsl with buildSrc.

5 min read · Mar 21, 2022

35 1

See all from Mesut G.

## Recommended from Medium

Tobias Wissmueller in ITNEXT

## Jetpack Compose: Navigation

A Practical Introduction

· 4 min read · Feb 24

72


Julie Perilla Garcia in Level Up Coding

## To Be A Great Software Developer— You Need a System

Here's how to build one.

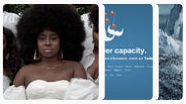
✦ · 7 min read · Jun 23

1.6K 23

## Lists

**Now in AI: Handpicked by Better Programming**

248 stories · 20 saves



**Staff Picks**

382 stories · 140 saves

The Useful Tech in Mac O'Clock

## 9 New Must-Have macOS Productivity Apps For Daily Usage

How did you manage all these days without these apps?

✦ · 11 min read · 6 days ago

1.7K 18

Mohammad Azam

## Building Large-Scale Apps with SwiftUI: A Guide to Modular Architecture

Update (03/02/2023): Added View Specific Logic section.

· 35 min read · Mar 2

181 1

Ankit Sangwan

## Docker for Android Development

6 min read · Jun 13

5

Prasoon Agrawal

## Navigation in Jetpack Compose Android.

Developing Android apps with Jetpack Compose has been one of the biggest game-changers in the world of mobile development. Its intuitive...

4 min read · Feb 13

175 3

See more recommendations