

Jetpack Compose Analyzation

with user list management application

8/16/2021

Contents

[**1.** **Basic** 2](#_Toc80205541)

[What is Jetpack Compose? 2](#_Toc80205542)

[Basics Of Compose 2](#_Toc80205543)

[**Composable functions** 2](#_Toc80205544)

[Advantages 2](#_Toc80205545)

[Disadvantages 2](#_Toc80205546)

[Integration 2](#_Toc80205547)

[**Create a new app with support for Jetpack Compose** 2](#_Toc80205548)

[**Create a new app with support for Jetpack Compose** 2](#_Toc80205549)

[Material Design 2](#_Toc80205550)

[**Use Material Design** 2](#_Toc80205551)

[**CREATE LIST** 3](#_Toc80205552)

[**Animation** 3](#_Toc80205553)

[**2.** **Sample application using Jetpack Compose** 3](#_Toc80205554)

[User List Management Application 3](#_Toc80205555)

[**Building UI Layout** 3](#_Toc80205556)

[**link view model** 3](#_Toc80205557)

[UI Library Integration 3](#_Toc80205558)

[**create library** 4](#_Toc80205559)

[**library integration** 8](#_Toc80205560)

# **Basic**

## What is Jetpack Compose?

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.



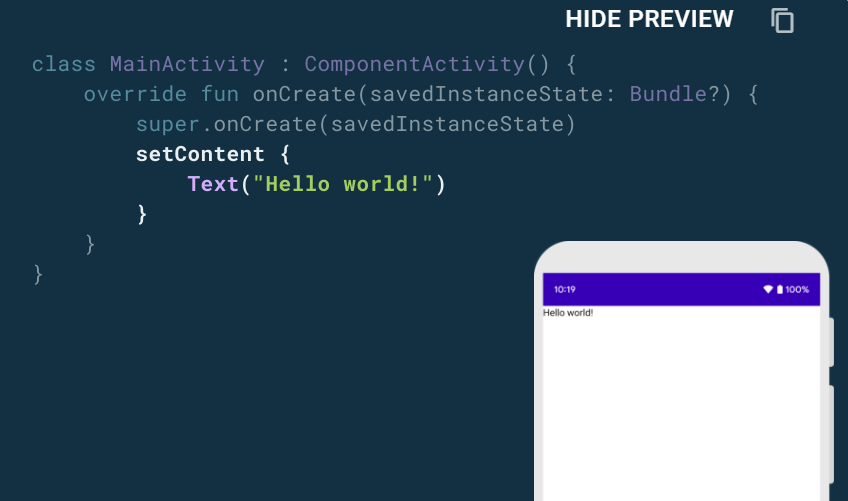
## Basics Of Compose

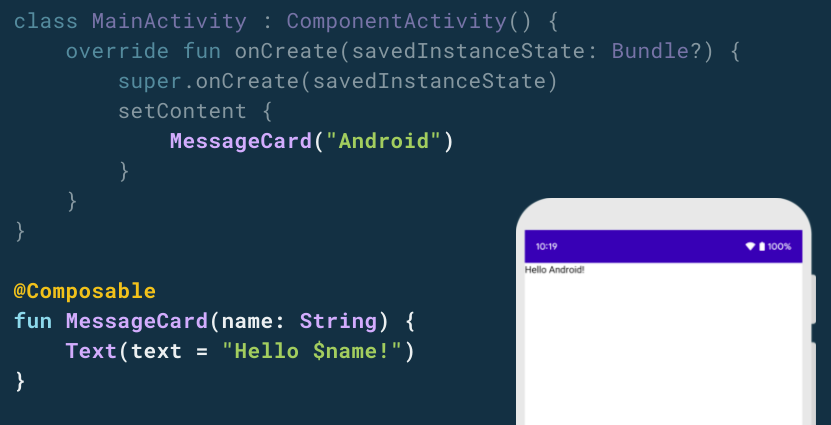
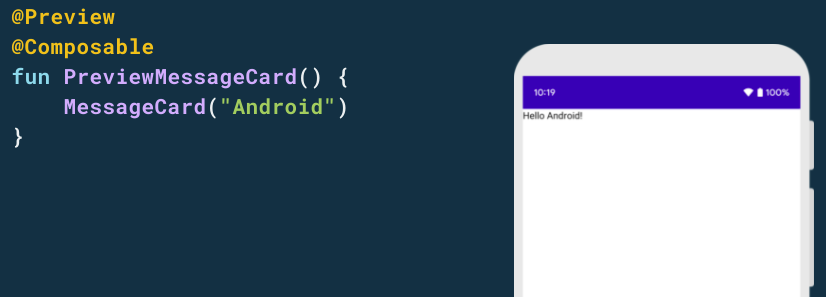
You won't be editing any XML layouts or using the Layout Editor. Instead, you will call Jetpack Compose functions to say what elements you want, and the Compose compiler will do the rest.

### **Composable functions**

Jetpack Compose is built around composable functions. These functions let you define your app's UI programmatically by describing how it should look and providing data dependencies, rather than focusing on the process of the UI's construction (initializing an element, attaching it to a parent, etc.). To create a composable function, just add the @Composable annotation to the function name.

* Add a text element



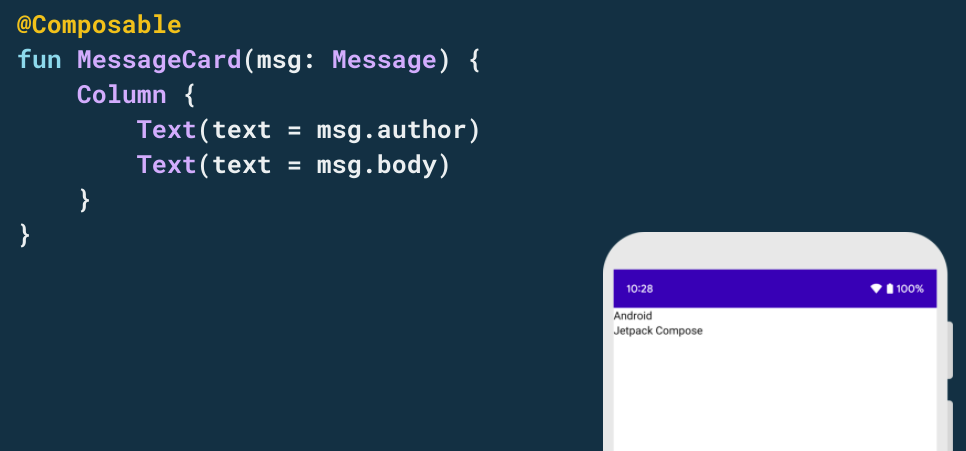
* Define a composable function 
* Preview your function in Android Studio 

**Layouts**

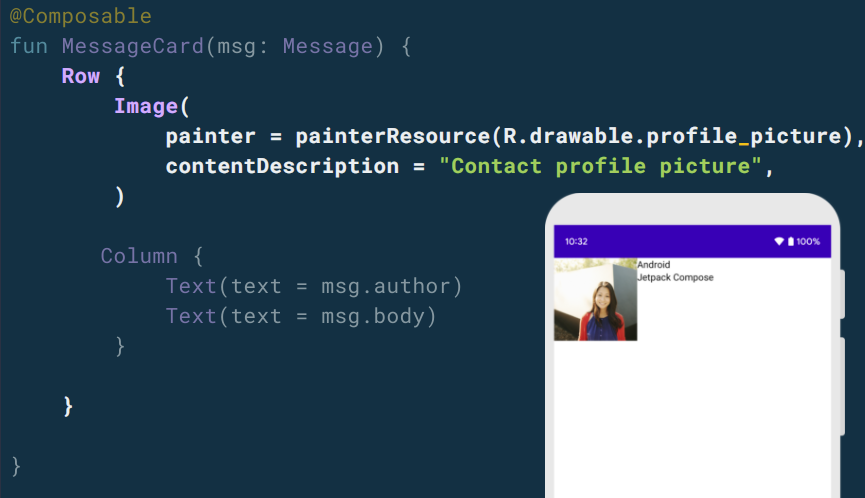
UI elements are hierarchical, with elements contained in other elements. In Compose, you build a UI hierarchy by calling composable functions from other composable functions.

* Using a Column

The Column function lets you arrange elements vertically. Add Column to the MessageCard() function.

You can use Row to arrange items horizontally and Box to stack elements. 

* Configure your layout

To decorate or configure a composable, Compose uses modifiers. They allow you to change the composable's size, layout, appearance or add high-level interactions, such as making an element clickable.

## Advantages

* Less code
  + Do more with less code and avoid entire classes of bugs, so code is simple and easy to maintain.
* Intuitive
  + Just describe your UI, and Compose takes care of the rest. As app state changes, your UI automatically updates.
* Accelerate Development
  + Compatible with all your existing code so y ou can adopt when and where you want. Iterate fast with live previews and full Android Studio support.
* Powerful
  + Create beautiful apps with direct access to the Android platform APIs and built-in support for Material Design, Dar theme, animations, and more.

## Disadvantages

* Jetpack Compose is still relatively new. Compose holds the promise for increased productivity, but new technologies can have unknown limitations and internal bugs that may need temporary workarounds.
* The declarative UI model, immutable UI and opinionated push toward MVVM may represent a significant learning curve for an existing development team.

## Integration

For the best experience developing with Jetpack Compose, you should download Android Studio Arctic Fox. That’s because when you use Android Studio to develop your app with Jetpack Compose, you can benefit from smart editor features, such as New Project templa

### **Create a new app with support for Jetpack Compose**

1. If you’re in the Welcome to Android Studio window, click Start a new Android Studio project. If you already have an Android Studio project open, select File > New > New Project from the menu bar.
2. In the Select a Project Template window, select Empty Compose Activity and click Next.
3. In the Configure your project window, do the following:
   1. Set the Name, Package name, and Save location as you normally would.
   2. Note that, in the Language dropdown menu, Kotlin is the only available option because Jetpack Compose works only with classes written in Kotlin.
   3. In the Minimum API level dropdown menu, select API level 21 or higher.
4. Click Finish.

### **Create a new app with support for Jetpack Compose**

<https://developer.android.com/jetpack/compose/interop/adding>

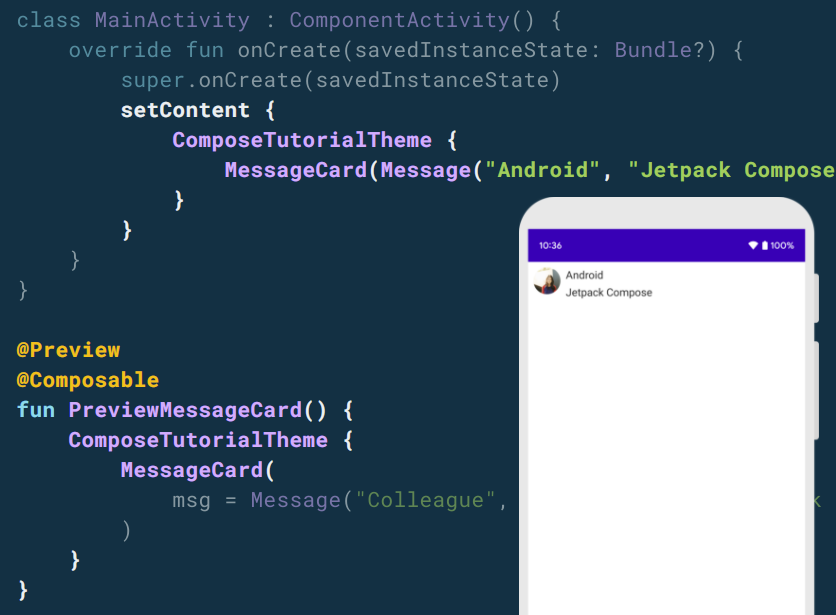
## Material Design

Compose is built to support material design principles. Many of its UI elements implement material design out of the box. In this lesson, you'll style your app with material widgets.

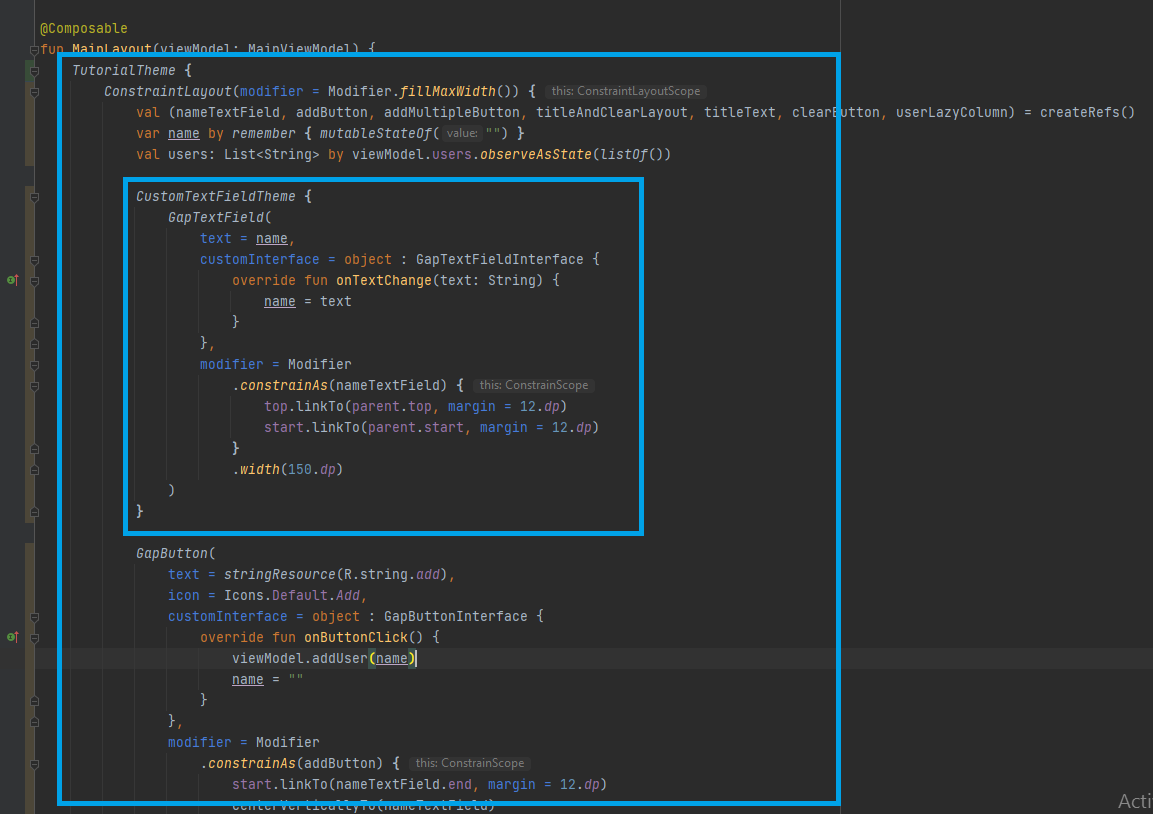
### **Use Material Design**

Jetpack Compose provides an implementation of Material Design and its UI elements out of the box. We'll improve the appearance of our MessageCard composable using Material Design styling.

To start, we wrap our MessageCard function with the Material theme created in your project, ComposeTutorialTheme in this case. Do it both in the @Preview and in the setContent function.

Material Design is built around three pillars: Color, Typography, Shape.

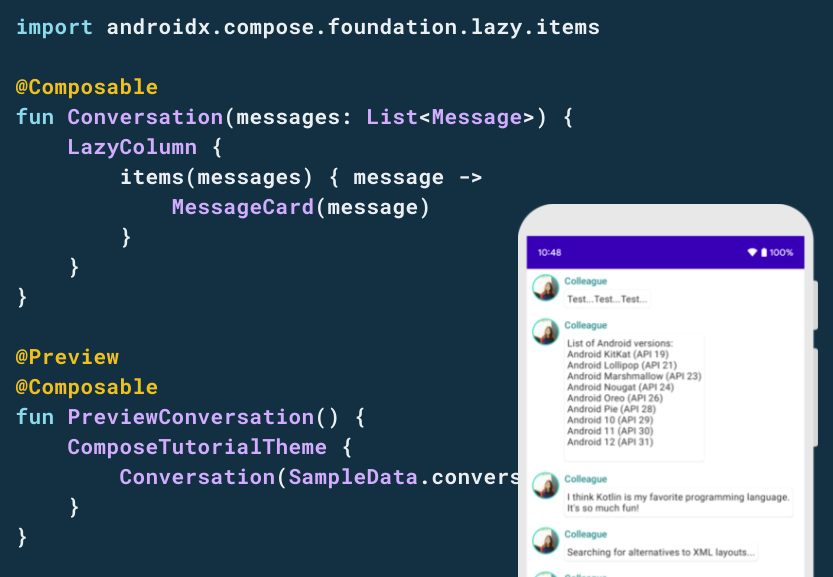
* **We can use certain theme for specific composable function inside main theme.**

TutorialTheme is applied through all components inside, but for GapTextField only CustomTextFieldTheme is working.

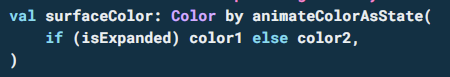
Lists and animations

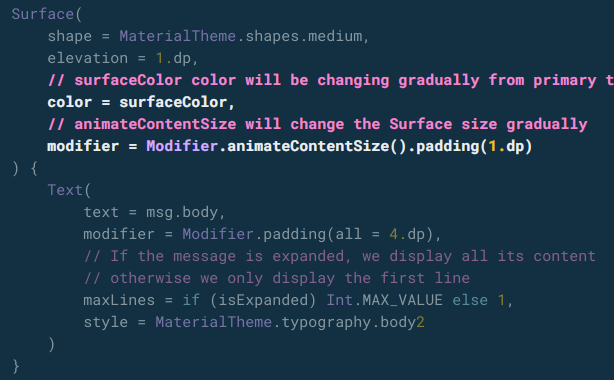
Lists and animations are everywhere in apps. In this lesson, you will learn how Compose makes it easy to create lists and fun to add animations.

### **CREATE LIST**

For creating list we can use Compose’s **LazyColumn** and **LazyRow**. These composables render only the elements that are visible on screen, so they are designed to be very efficient for long lists. At the same time, they avoid the complexity of RecyclerView with XML layouts.

### **Animation**

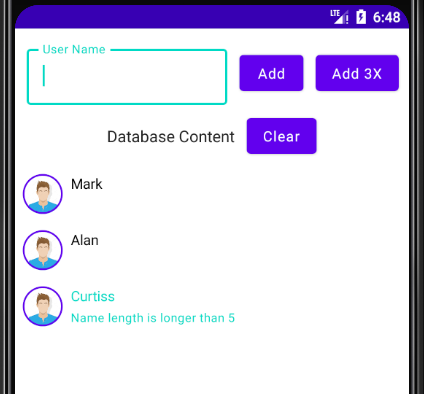
We will animate the background color by gradually modifying its value from color1 to clor2. To do so, we will use the **animateColorAsState** function. And, we will use the **animateContentSize** modifier to animate the message container size smoothly.



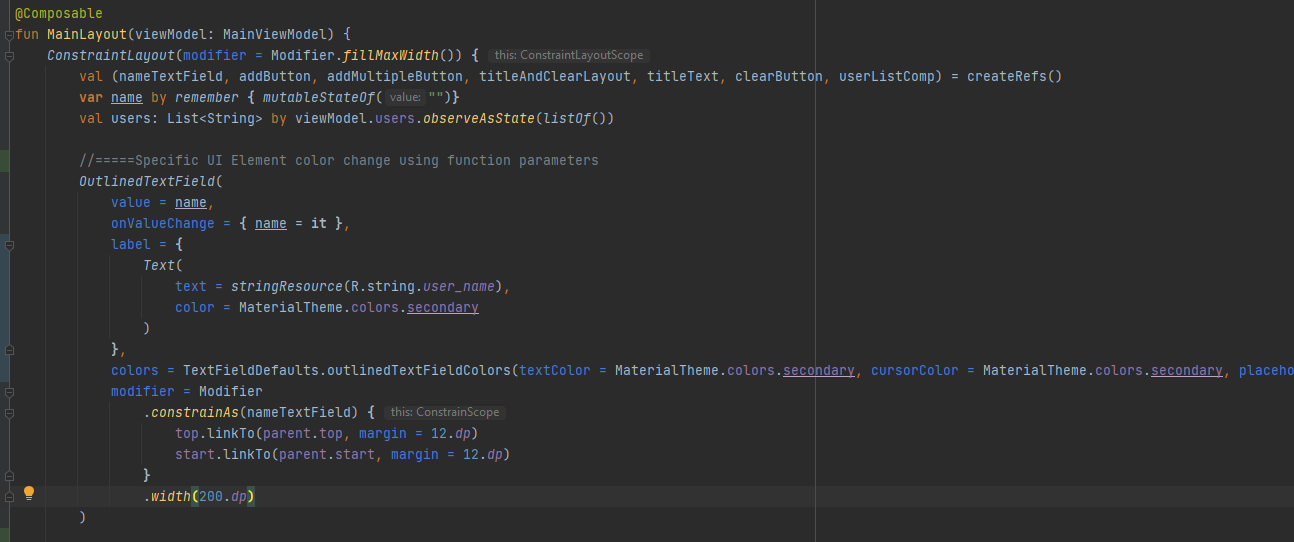
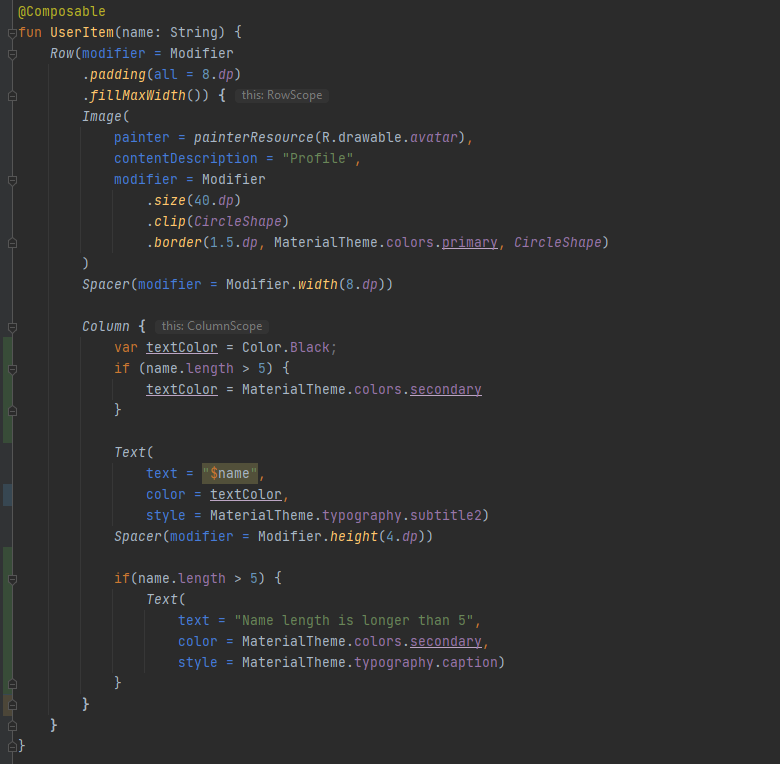
# **Sample application using Jetpack Compose**

## User List Management Application

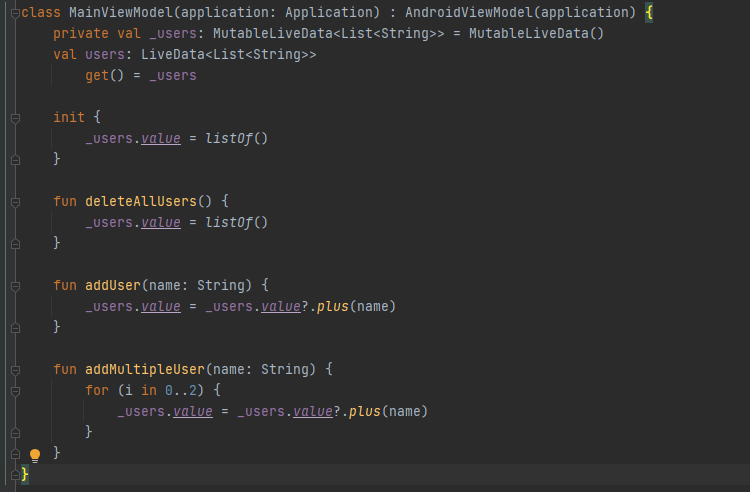
We will jump into Jetpack Compose more while developing user list management application step by step.

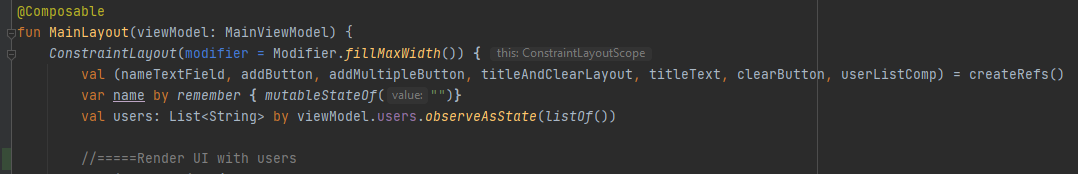


### **Building UI Layout**

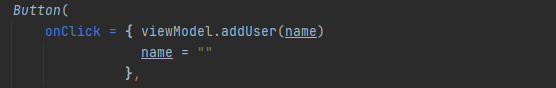
* Composable Functions used in App
  + MainLayout: Main page
    - ConstraintLayout for layout management
    - LazyColumn for user item list
  + UserItem: Each user item in list
    - Avatar image and column with name & description

### **link view model**

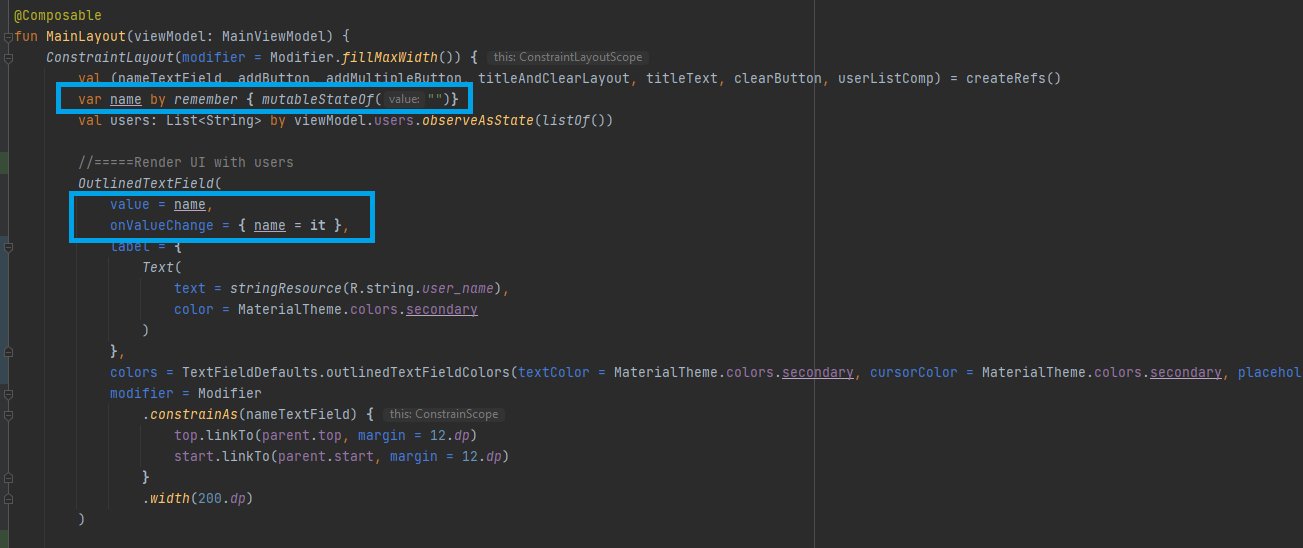
* MainViewModel
* Observe ViewModel data & Update UI

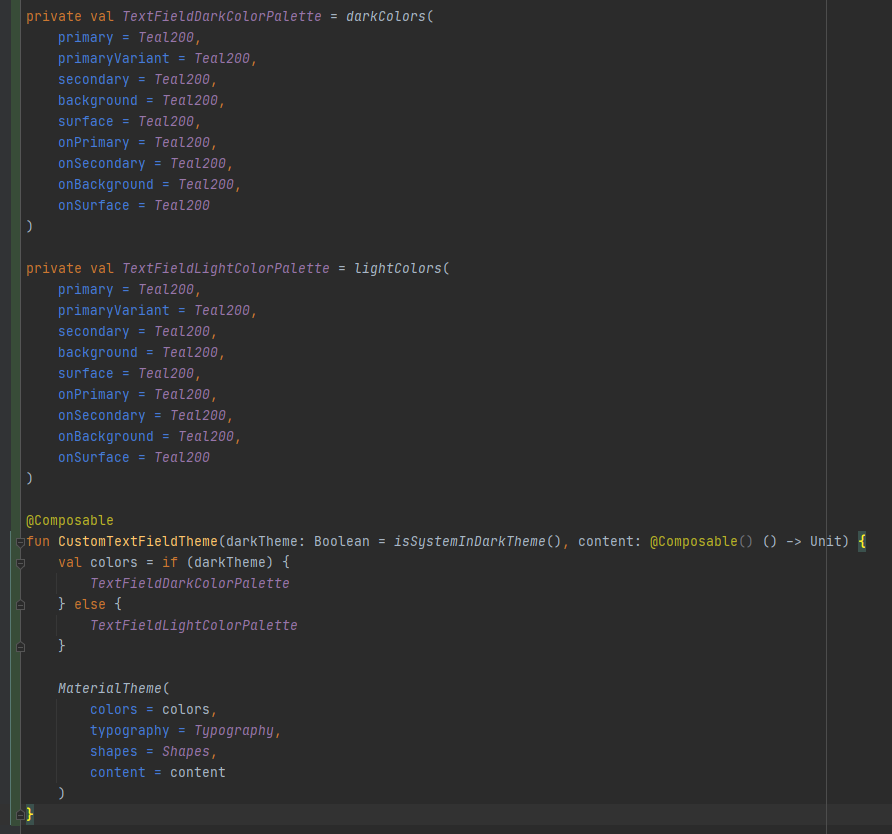
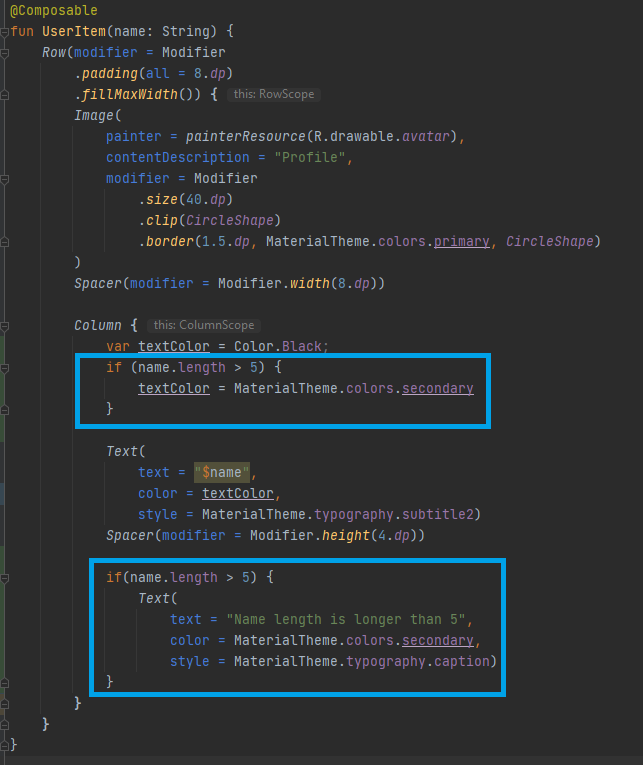
observeAsState observes a LiveData<T> and returns a State<T> object that is updated whenever the LiveData changes. State<T> is an observable type that Jetpack Compose can use directly. observeAsState will observe the LiveData only while it is in the composition.

* Receiving Button Click & Text Change Events

When button is clicked, it’s calling viewModel’s function for updating viewModel data.

* Modifying TextField value when user input

Compose is declarative and as such the only way to update it is by calling the same composable with new arguments. These arguments are representations of the UI state. Any time a state is updated a recomposition takes place. As a result, things like TextField don’t automatically update like they do in imperative XML based views. A composable has to explicitly be told the new state in order for it to update accordingly.

* Changing TextField border color by theme creation & updating composable function parameters
  + Theme Creation
    - Define custom theme for TextField (CustomTextFieldTheme)
    - Use custom theme in composable function
  + Composable function parameters
* Rendering different items in LazyColumn
* LazyColumn animation when item is removed (Like RecyclerView)

It is not officially supported yet but they are working on it. You can probably achieve it but in a hacky way.

**Workaround**

When your list updates, your composable is recreated and it doesn't support animations for items yet, so you have to add a boolean variable on your item and change the value when it's "deleted" instead of removing it from the list. Once the updated list is shown, you can animate the item being removed with a delay and then update the list without it once the animation is over.

## UI Library Integration

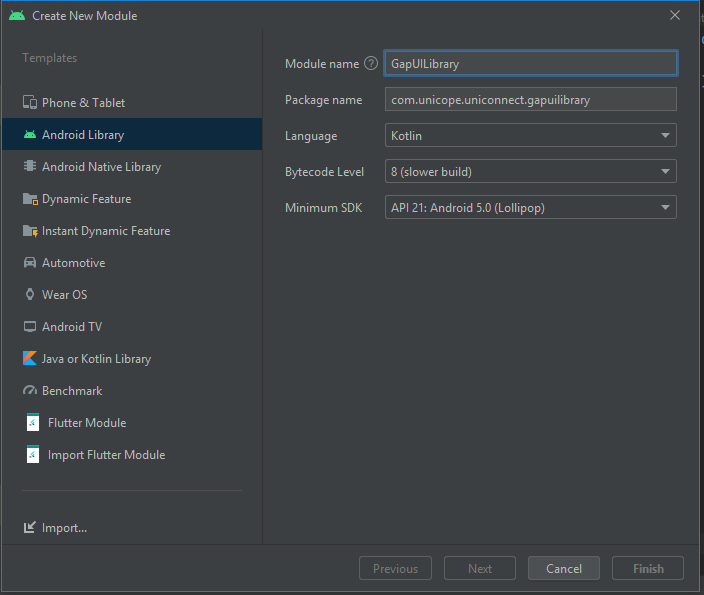
We can create custom UI library using Jetpack compose.

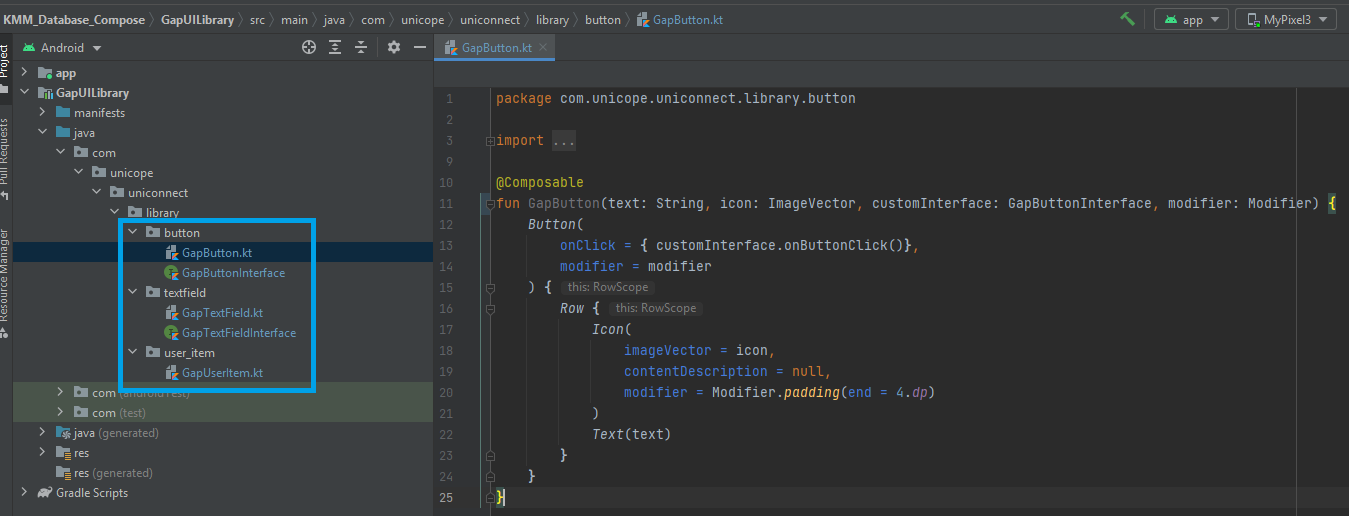
Below example is showing on how to create custom TextField, Button and ListItem as library (.aar) and UI library integration step by step.

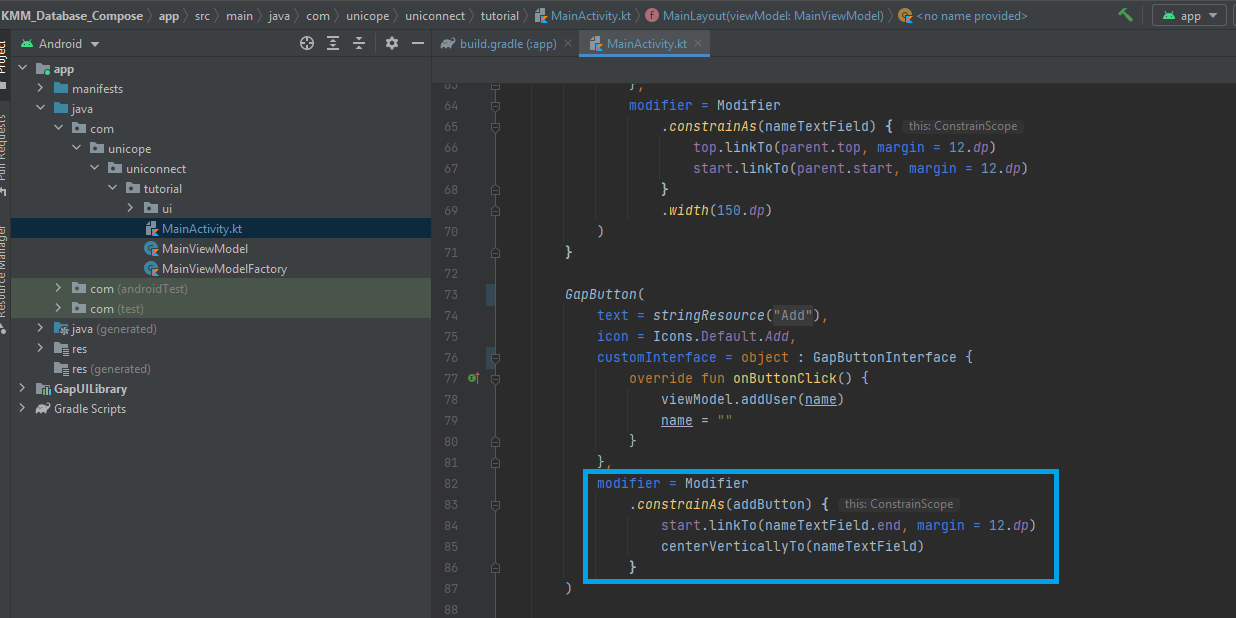
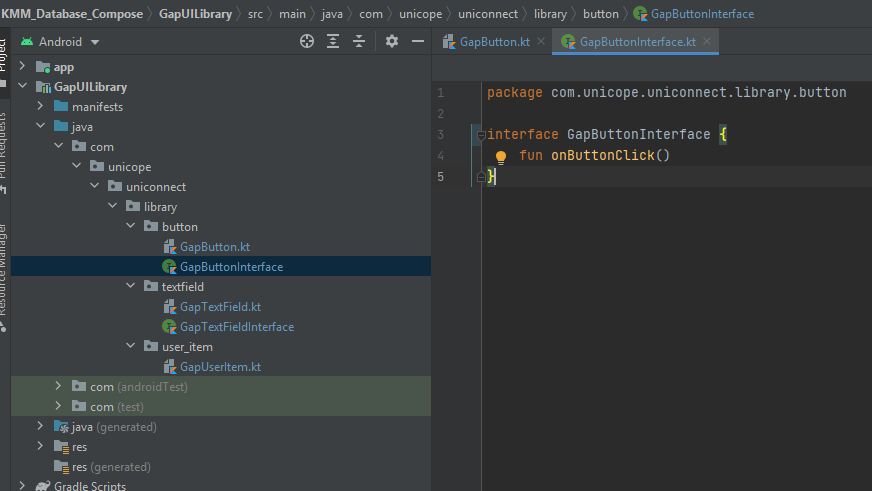
We will create custom RED color TextField.

### **create library**

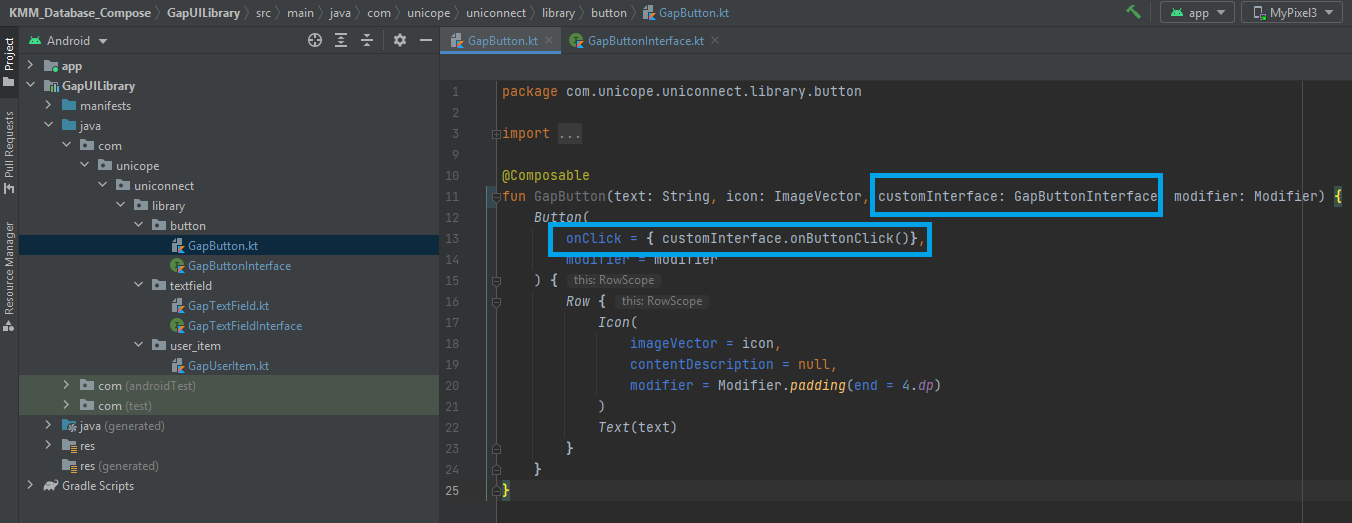
1. Create new module and input library name.

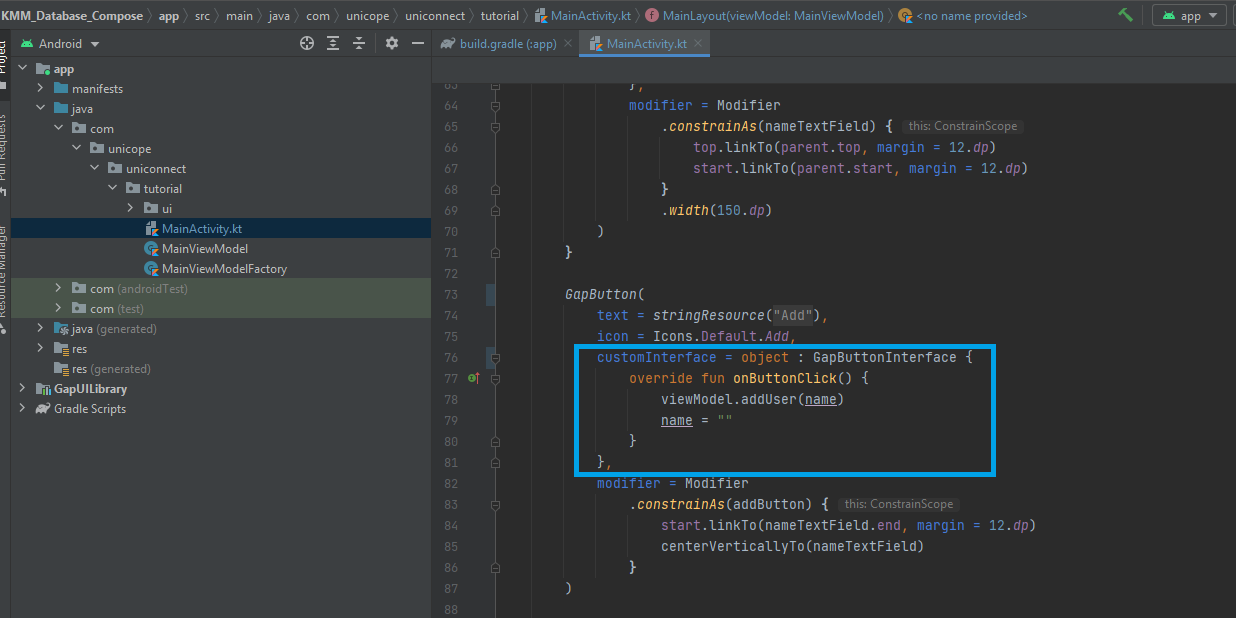
File/New/New Module

1. Create new kotlin files under new module and implement library composable functions.

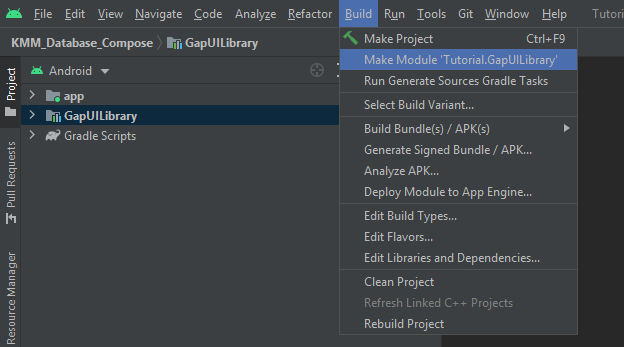
* Use modifier as parameter to decorate or configure a composable
* Listener & Callbacks
  + We can use interface that defines composable UI actions in library. 
  + This interface can be one of the parameter of your composable function.

Interface callback will be called when button is clicked and we willl listen it on main project.

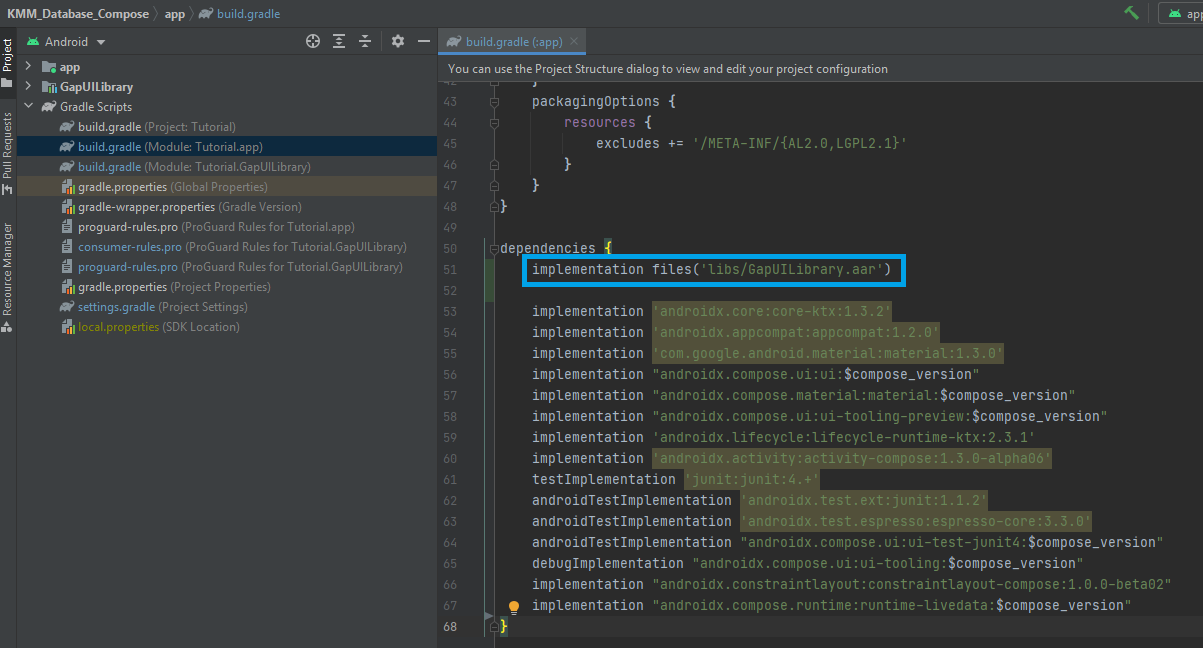


* + Define listeners on main project to receive library composable UI actions. (This is after .aar is imported in main project)

1. Export new module as library. Then .aar file is generated under new module.

…/build/outputs/LIBRARY\_NAME.aar

### **library integration**

1. Copy .aar file to app/libs and import library using app build gradle.
2. Use library composable function like standard composable functions.