LESSON 02 COMPOSABLES

AARHUS
UNIVERSITY
DEPARTMENT OF ELECTRICAL AND COMPUTER
ENGINEERING

20 OCTOBER 2022

MICHEL VEDEL HOWARD
ASSISTANT PROFESSOR

CONTENTS

GRADLE FILES

Gradle is a build and package manager tool like NuGet or Npm

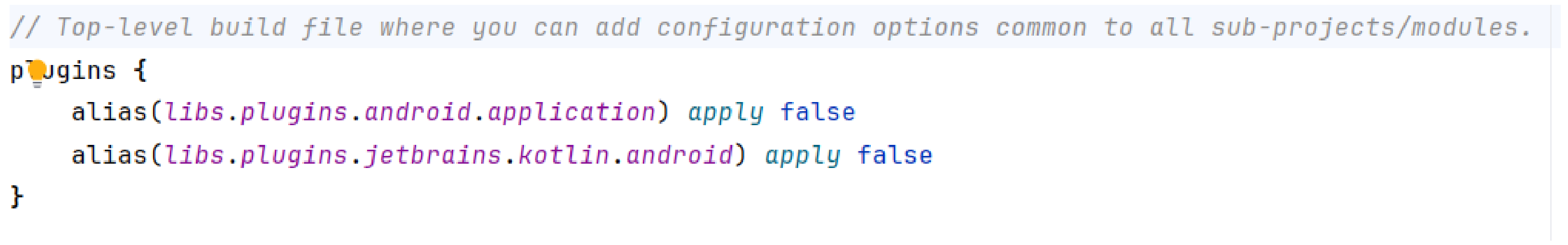
The Gradle files are written in Kotlin

There are at least two Gradle files one for the project and one for the app module

We will not implement more than one module in an app.

PROJECT GRADLE

The project Gradle file specifies global dependencies



```
// Top-level build file where you can add configuration options common to all sub-projects/modules.
plugins {
    alias(libs.plugins.android.application) apply false
    alias(libs.plugins.jetbrains.kotlin.android) apply false
}
```

This specifies two plugins used building android applications in Kotlin

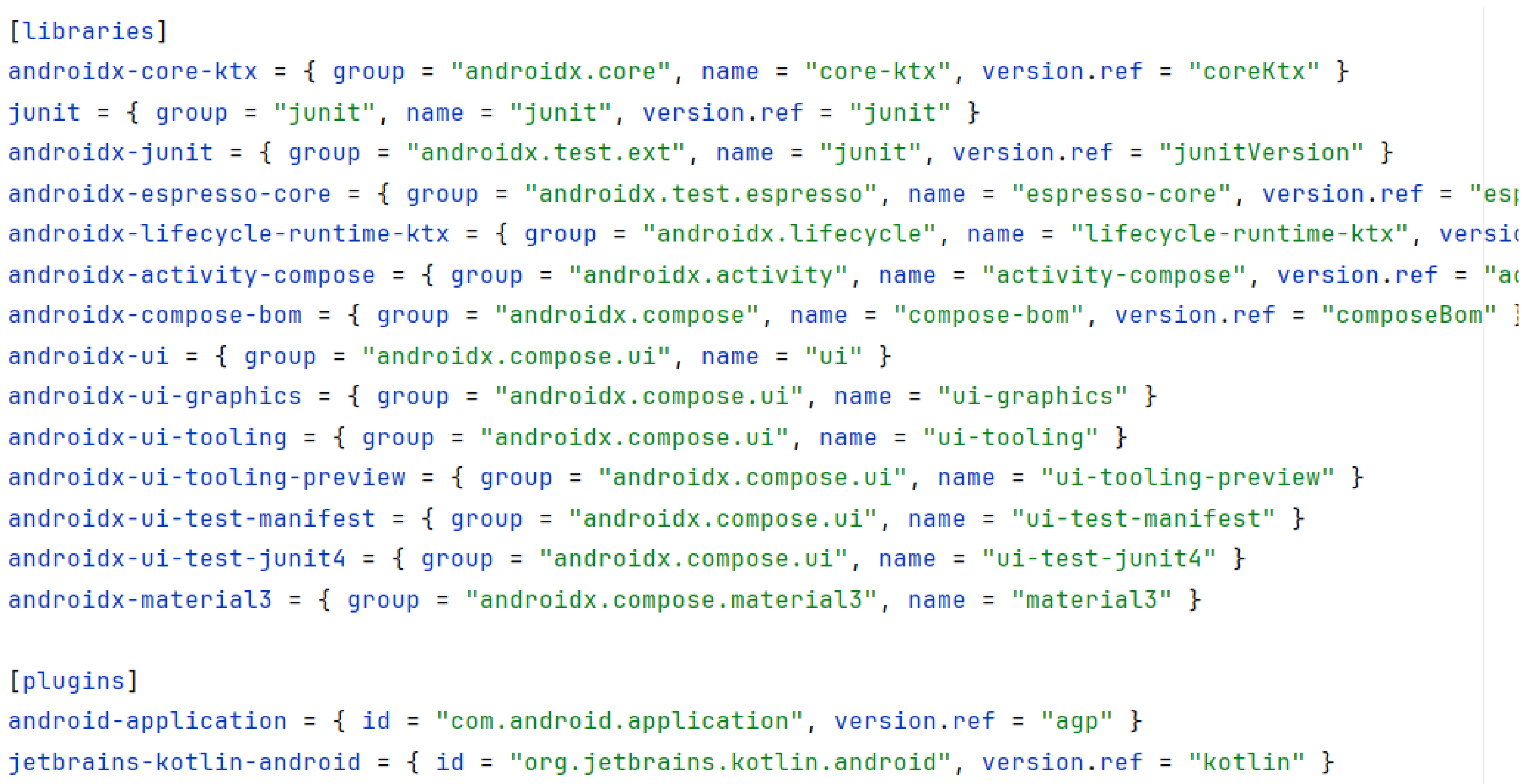
The are not applied in this file, but specified

The specification of the actual repository ids of the plugins are made in the

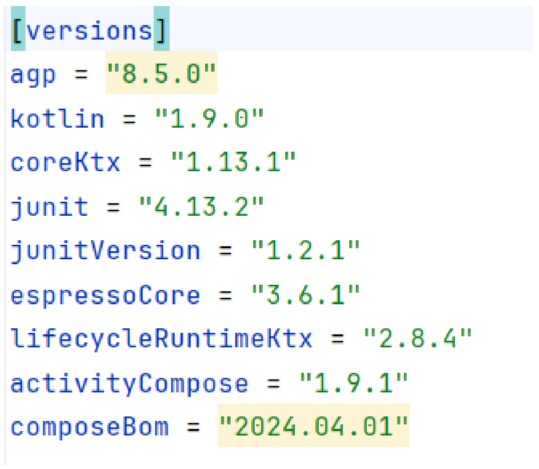
lib.versions.toml file

The gradle file refers to the aliases in this file

LIB.VERSIONS.TOML



```toml
[libraries]
androidx-core-ktx = { group = "androidx.core", name = "core-ktx", version.ref = "coreKtx" }
junit = { group = "junit", name = "junit", version.ref = "junit" }
androidx-junit = { group = "androidx.test.ext", name = "junit", version.ref = "junitVersion" }
androidx-espresso-core = { group = "androidx.test.espresso", name = "espresso-core", version.ref = "esp
androidx-lifecycle-runtime-ktx = { group = "androidx.lifecycle", name = "lifecycle-runtime-ktx", versi
androidx-activity-compose = { group = "androidx.activity", name = "activity-compose", version.ref = "ac
androidx-compose-bom = { group = "androidx.compose", name = "compose-bom", version.ref = "composeBom" }
androidx-ui = { group = "androidx.compose.ui", name = "ui" }
androidx-ui-graphics = { group = "androidx.compose.ui", name = "ui-graphics" }
androidx-ui-tooling = { group = "androidx.compose.ui", name = "ui-tooling" }
androidx-ui-tooling-preview = { group = "androidx.compose.ui", name = "ui-tooling-preview" }
androidx-ui-test-manifest = { group = "androidx.compose.ui", name = "ui-test-manifest" }
androidx-ui-test-junit4 = { group = "androidx.compose.ui", name = "ui-test-junit4" }
androidx-material3 = { group = "androidx.compose.material3", name = "material3" }

[plugins]
android-application = { id = "com.android.application", version.ref = "agp" }
jetbrains-kotlin-android = { id = "org.jetbrains.kotlin.android", version.ref = "kotlin" }
```



```toml
[versions]
agp = "8.5.0"
kotlin = "1.9.0"
coreKtx = "1.13.1"
junit = "4.13.2"
junitVersion = "1.2.1"
espressoCore = "3.6.1"
lifecycleRuntimeKtx = "2.8.4"
activityCompose = "1.9.1"
composeBom = "2024.04.01"
```

MODULE GRADLE

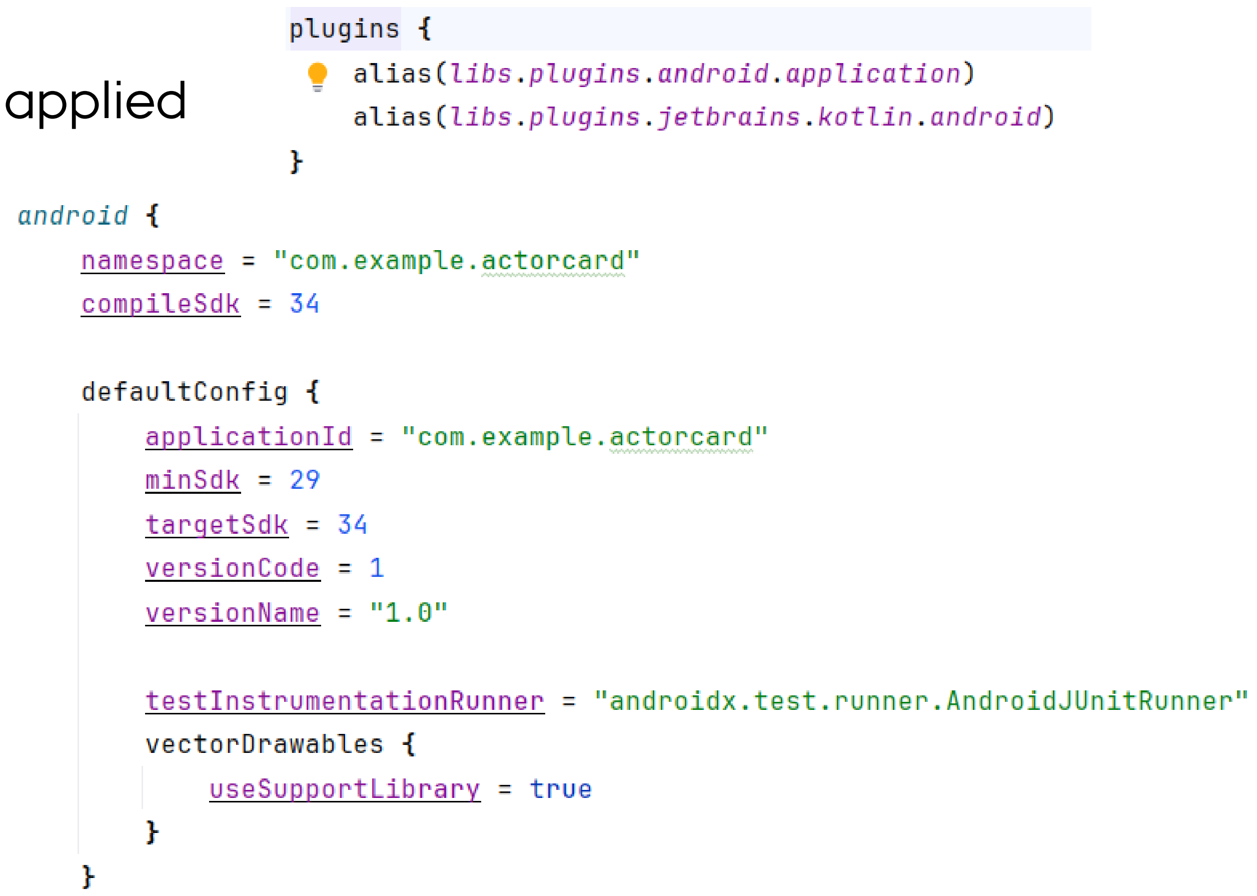
The previously defined plugins are applied

The SDK compiled to

The minimal SDK it can run on

Target SDK usually=Compile SDK

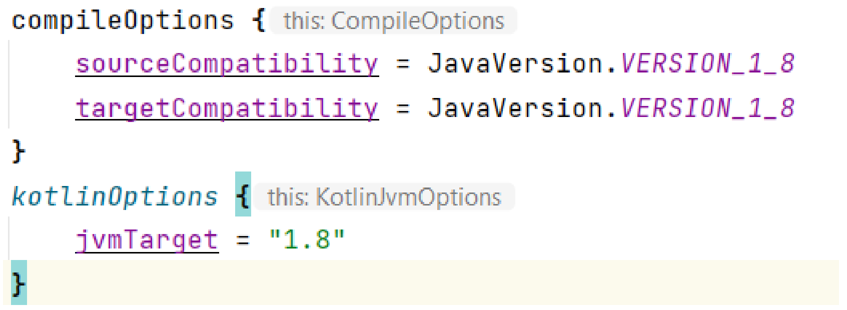
But is the SDK the app was tested and designed to



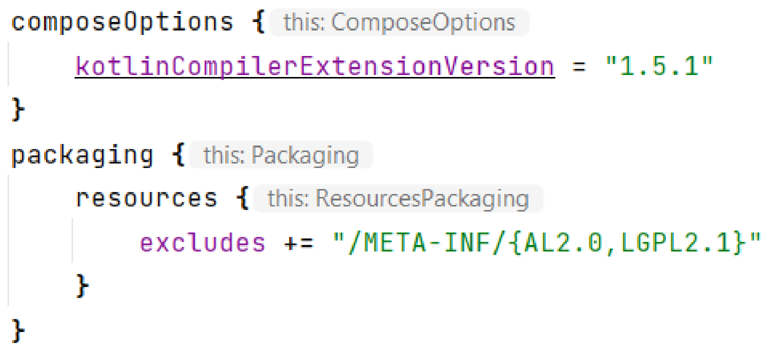
```
plugins {
    alias(libs.plugins.android.application)
    alias(libs.plugins.jetbrains.kotlin.android)
}
android {
    namespace = "com.example.actorcard"
    compileSdk = 34

    defaultConfig {
        applicationId = "com.example.actorcard"
        minSdk = 29
        targetSdk = 34
        versionCode = 1
        versionName = "1.0"

        testInstrumentationRunner = "androidx.test.runner.AndroidJUnitRunner"
        vectorDrawables {
            useSupportLibrary = true
        }
    }
```

MODULE GRADLE

Java compatibility

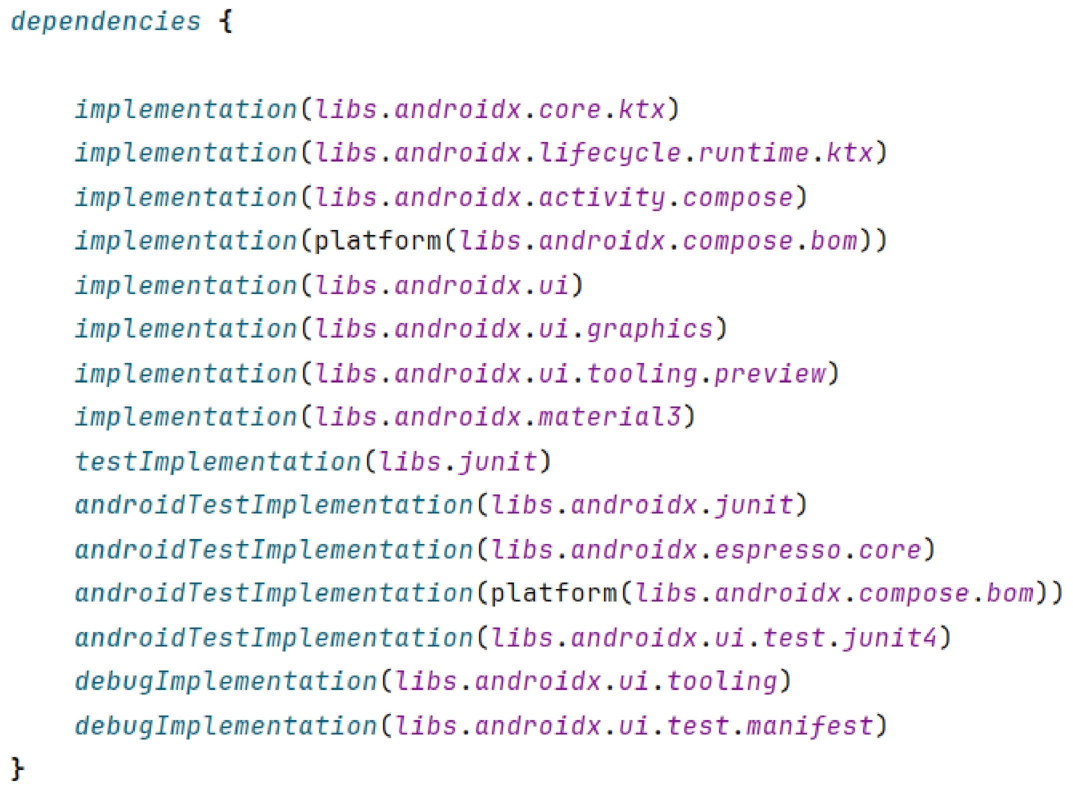

```
compileOptions { this: CompileOptions
    sourceCompatibility = JavaVersion.VERSION_1_8
    targetCompatibility = JavaVersion.VERSION_1_8
}
kotlinOptions { this: KotlinJvmOptions
    jvmTarget = "1.8"
}
```

Compose extension and packaging


```
composeOptions { this: ComposeOptions
    kotlinCompilerExtensionVersion = "1.5.1"
}
packaging { this: Packaging
    resources { this: ResourcesPackaging
        excludes += "/META-INF/{AL2.0,LGPL2.1}"
    }
}
```

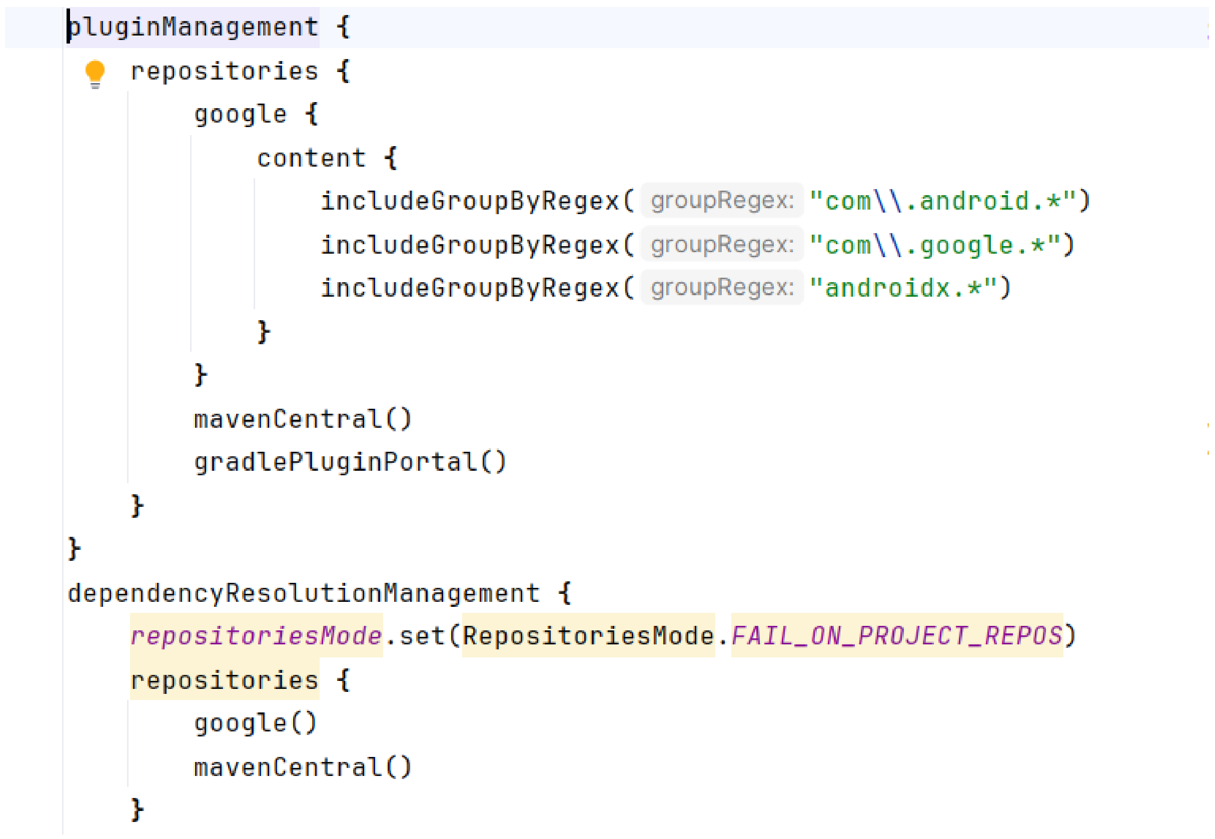
DEPENDENCIES

Dependencies which are important, try to use newest versions the IDE will help identify old versions



```
dependencies {

    implementation(libs.androidx.core.ktx)
    implementation(libs.androidx.lifecycle.runtime.ktx)
    implementation(libs.androidx.activity.compose)
    implementation(platform(libs.androidx.compose.bom))
    implementation(libs.androidx.ui)
    implementation(libs.androidx.ui.graphics)
    implementation(libs.androidx.ui.tooling.preview)
    implementation(libs.androidx.material3)
    testImplementation(libs.junit)
    androidTestImplementation(libs.androidx.junit)
    androidTestImplementation(libs.androidx.espresso.core)
    androidTestImplementation(platform(libs.androidx.compose.bom))
    androidTestImplementation(libs.androidx.ui.test.junit4)
    debugImplementation(libs.androidx.ui.tooling)
    debugImplementation(libs.androidx.ui.test.manifest)
}
```

REPOSITORIES SETTINGS.GRADLE.KTS



```
pluginManagement {
    repositories {
        google {
            content {
                includeGroupByRegex( groupRegex: "com\\.android.*")
                includeGroupByRegex( groupRegex: "com\\.google.*")
                includeGroupByRegex( groupRegex: "androidx.*")
            }
        }
        mavenCentral()
        gradlePluginPortal()
    }
}
dependencyResolutionManagement {
    repositoriesMode.set(RepositoriesMode.FAIL_ON_PROJECT_REPOS)
    repositories {
        google()
        mavenCentral()
    }
```

CUSTOM COMPOSABLES

Jet pack consists of a lot of predefined composables. But to make any thing useful you must construct your own composables.

CUSTOM COMPOSABLES

Composables are functions!

The are annotated with **@Composable** which turns a function into a composable

```
@Composable
fun Greeting(name: String, modifier: Modifier = Modifier) {
    Text(
        text = "Hello $name!",
        modifier = modifier
    )
}
```

This composable takes two parameters the text and a modifier that defaults to a static immutable Modifier this modifier is set in the text also.

THE MODIFIER

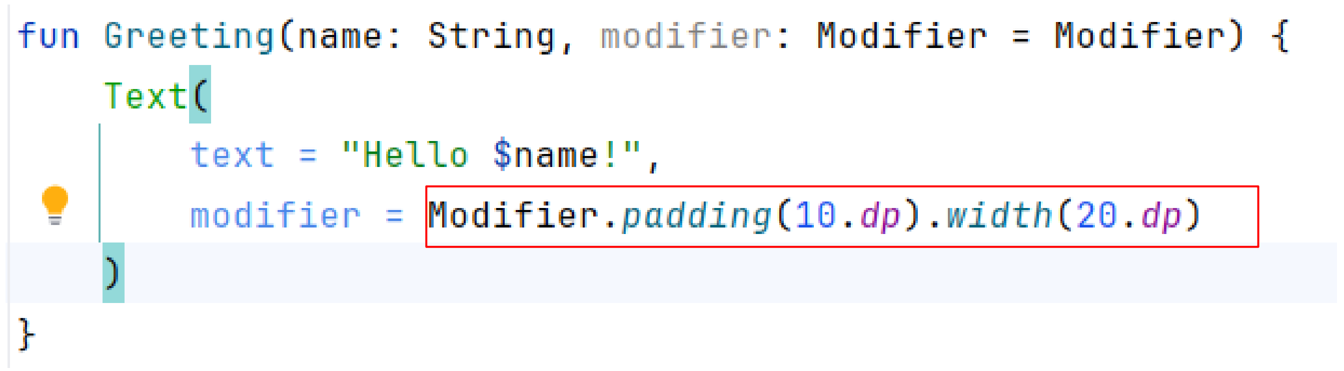
The modifier is an important object.

It is the styling object in compose and it is equipped with a host of properties that style individual predefined components

We will encounter this object again and again in the future

NEW MODIFIER

The invocation of functions on Modifier generates new objects of type Modifier



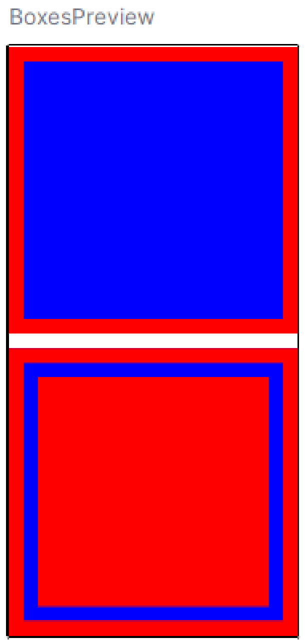
```kotlin
fun Greeting(name: String, modifier: Modifier = Modifier) {
    Text(
        text = "Hello $name!",
        modifier = Modifier.padding(10.dp).width(20.dp)
    )
}
```

CHAINING THE MODIFIER

```kotlin
@Composable
fun FirstBox(modifier: Modifier = Modifier) {
    Box(modifier = modifier) { }
}

@Composable
fun SecondBox(modifier: Modifier = Modifier) {
    Box(
        modifier = modifier.then(
            Modifier
                .height(60.dp)
                .width(60.dp)
                .background(color = Color.Red)
                .border(width = 20.dp, color = Color.Blue)
        )
    ) { }
}
```

```kotlin
@Preview
@Composable
fun BoxesPreview() {
    BasicExamplesTheme(dynamicColor = false) {
        val modifier = Modifier
            .height(200.dp)
            .width(200.dp)
            .background(color = Color.Blue)
            .border(width = 10.dp, color = Color.Red)
        Column(
            verticalArrangement = Arrangement.spacedBy(10.dp),
            horizontalAlignment = Alignment.CenterHorizontally
        ) {
            FirstBox(modifier)
            SecondBox(modifier)
        }
    }
}
```

CHAINING MODIFIERS



CHAINED MODIFIERS RULES

Modifiers are applied based on the sequence they are specified. This means that the effects of one modifier can be influenced by the modifiers that came before it and vice versa.

Size type modifiers are generally not overridden the first size has mandate

Color type modifier are overridden

Borders are applied in reverse order

Paddings are added

And many more rather specialized rules

CUSTOM COMPOSABLES

```kotlin
fun Menu(modifier: Modifier = Modifier) {
    Column {
        Row(
            modifier = modifier,
            verticalAlignment = Alignment.CenterVertically,
            horizontalArrangement = Arrangement.spacedBy(20.dp)
        ) {
            IconButton(onClick = { Log.v(TAG,  msg: "Home") })
            {
                Icon(imageVector = Icons.Default.Home, contentDescription = "Home")
            }
            Text(modifier = modifier, text = "Home")
        }
        Row(
            modifier = modifier,
            verticalAlignment = Alignment.CenterVertically,
            horizontalArrangement = Arrangement.spacedBy(20.dp)
        ) {
            IconButton(onClick = { Log.v(TAG,  msg: "Search") })
            {
                Icon(imageVector = Icons.Default.Search, contentDescri
            }
            Text(modifier = modifier, text = "Search")
        }
    }
}
```

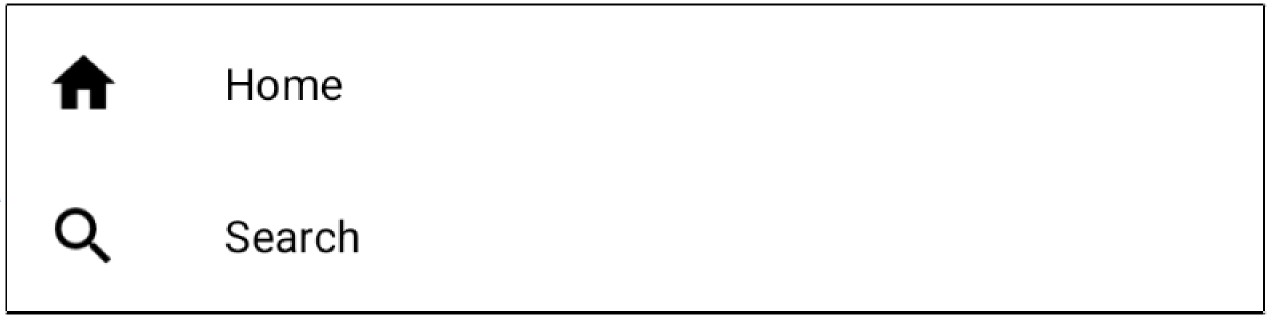
A composable calling a composable

A composable calling several composables

A composable calling a composable

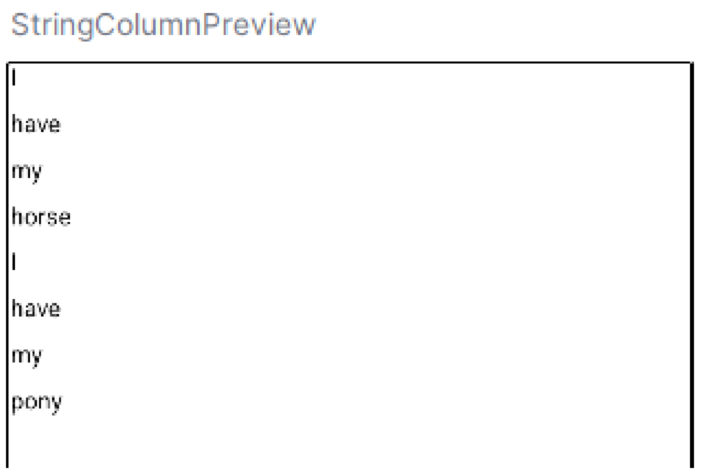
A composable calling several composables

MenuPreview



CALLING MANY COMPOSABLES

```
@Composable
fun StringColumn(text: String) {
    Column(
        modifier = Modifier.fillMaxSize(),
        verticalArrangement = Arrangement.spacedBy(10.dp)
    ) {
        text.split( ...delimiters: " ").forEach { Text(text = it) }
    }
}

@Preview
@Composable
fun StringColumnPreview(){
    StringColumn(text = "I have my horse I have my pony")
}
```



StringColumnPreview

```
I
have
my
horse
I
have
my
pony
```

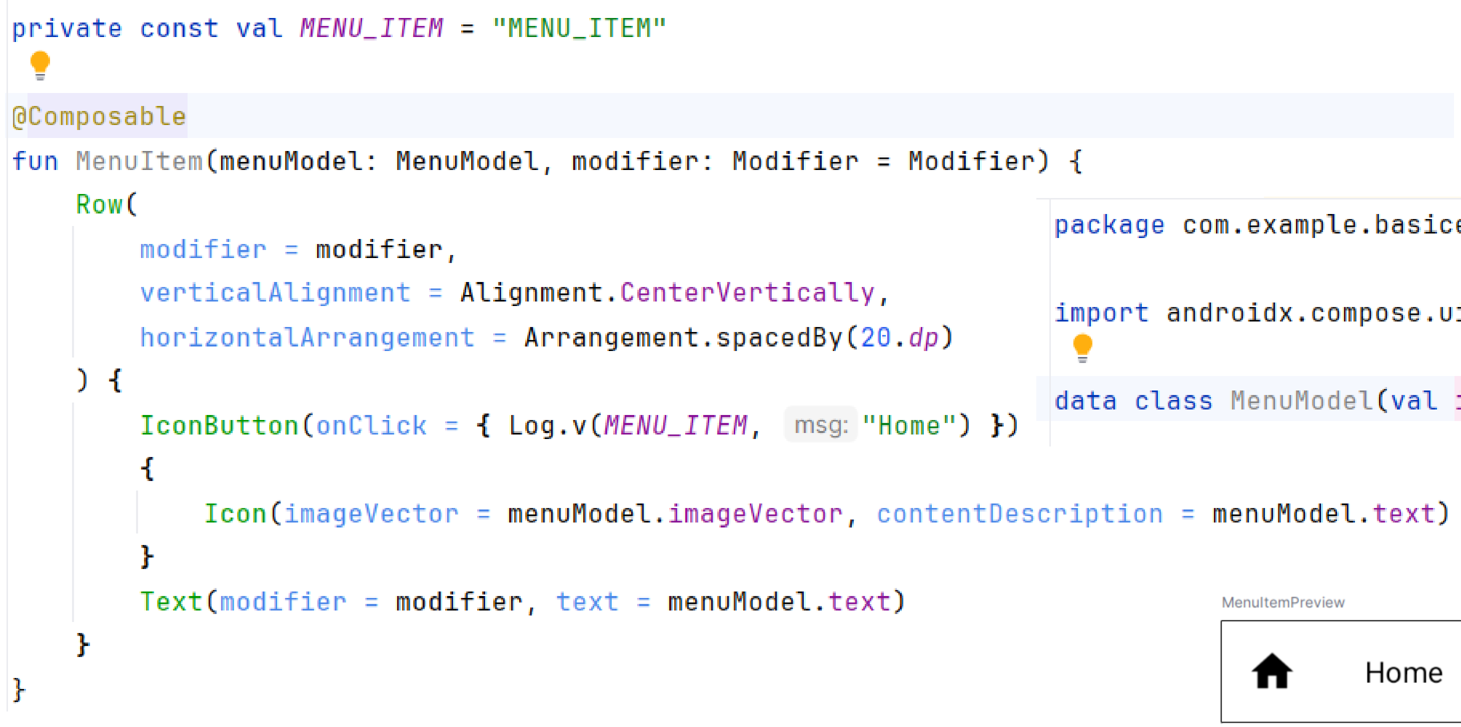
REUSE COMPOSABLES

We saw in the menu composable that there was a lot of code repetition

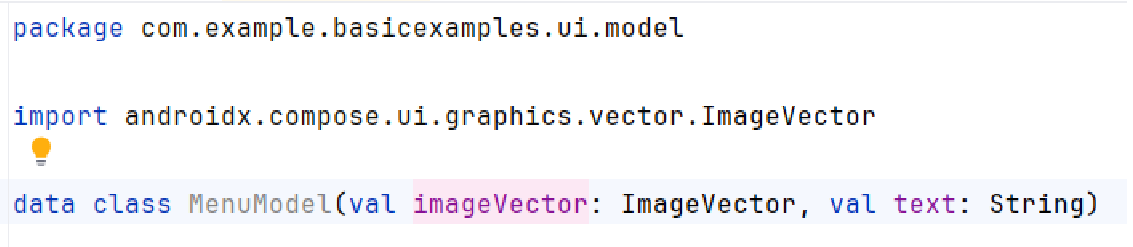
And custom composables are exactly for getting this repetition down to a minimum

MENU ITEM

It is obvious that the row element is duplicated for each menu item, so this is a candidate for a separate composable. And that the parameters should be structured

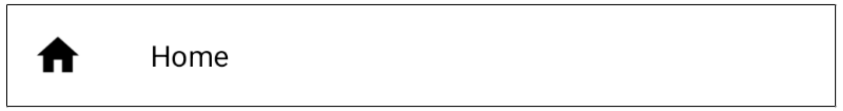


```kotlin
private const val MENU_ITEM = "MENU_ITEM"

@Composable
fun MenuItem(menuModel: MenuModel, modifier: Modifier = Modifier) {
    Row(
        modifier = modifier,
        verticalAlignment = Alignment.CenterVertically,
        horizontalArrangement = Arrangement.spacedBy(20.dp)
    ) {
        IconButton(onClick = { Log.v(MENU_ITEM, msg: "Home") })
        {
            Icon(imageVector = menuModel.imageVector, contentDescription = menuModel.text)
        }
        Text(modifier = modifier, text = menuModel.text)
    }
}
```



```kotlin
package com.example.basicexamples.ui.model

import androidx.compose.ui.graphics.vector.ImageVector

data class MenuModel(val imageVector: ImageVector, val text: String)
```

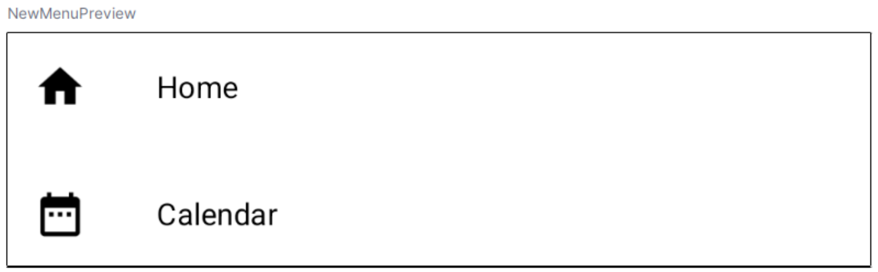
MenuItemPreview



THE NEW MENU

Now we must compose all these composables into a single menu with some structured parameter.

```
@Composable
fun NewMenu(models: List<MenuModel>, modifier: Modifier = Modifier) {
    Column(verticalArrangement = Arrangement.spacedBy(10.dp)) {
        models.forEach { MenuItem(menuModel = it, modifier = modifier) }
    }
}

@Preview
@Composable
fun NewMenuPreview() {
    val menuItemModels = listOf(
        MenuModel(imageVector = Icons.Default.Home,  text: "Home"),
        MenuModel(imageVector = Icons.Default.DateRange,  text: "Calendar")
    )
    NewMenu(models = menuItemModels, modifier = Modifier.fillMaxWidth())
}
```



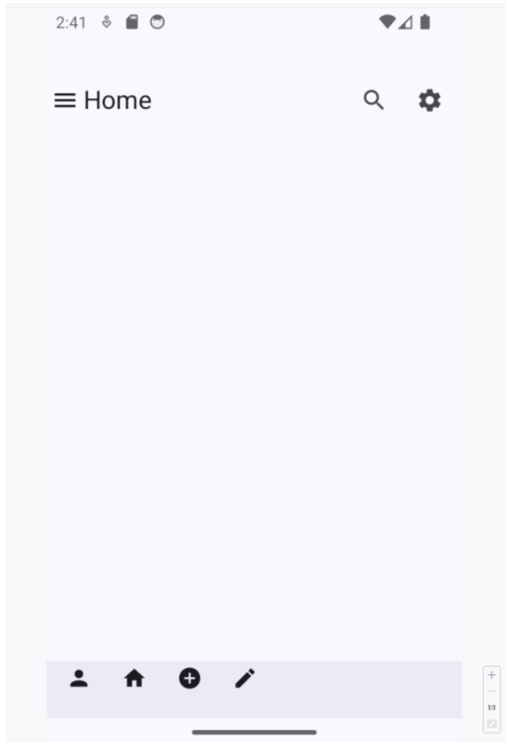
MENUS AND STUFF

In the next examples we will look at features of scaffolding and menu bars and menus

SCAFFOLD

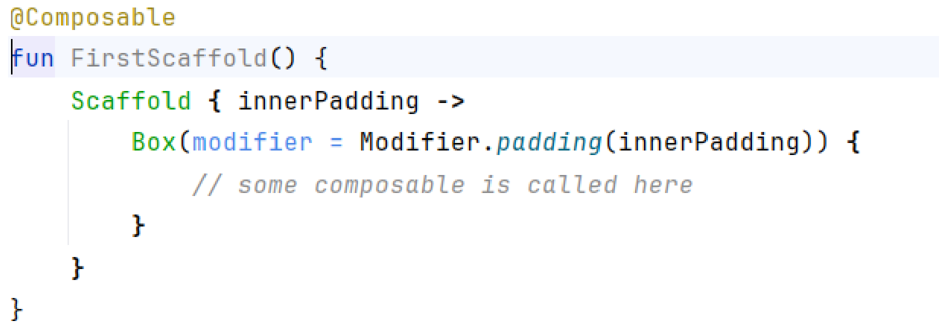
Scaffold is a built in composable that offers the easy construction of top and bottom bars.

It is default present is the empty-activity template



SCAFFOLD

First step is to use the scaffold composable



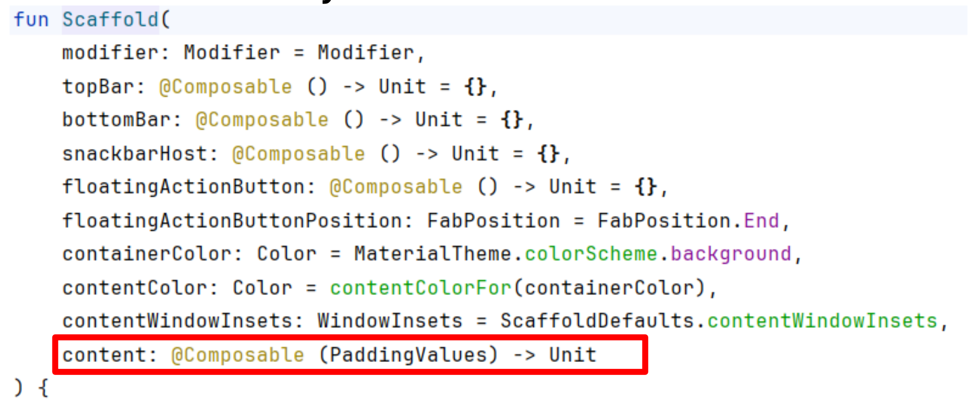
```kotlin
@Composable
fun FirstScaffold() {
    Scaffold { innerPadding ->
        Box(modifier = Modifier.padding(innerPadding)) {
            // some composable is called here
        }
    }
}
```

The content of the scaffold (in the middle of it) goes into the curly braces

The content in the curly braces should be a

function of Paddingvalues calling a composable

And we provide a Box(kind of like a Div in html)



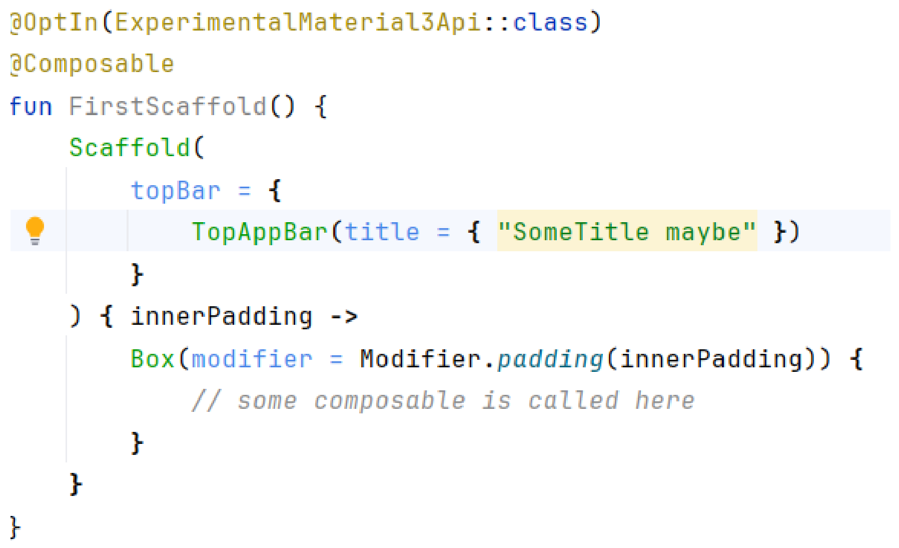
```kotlin
fun Scaffold(
    modifier: Modifier = Modifier,
    topBar: @Composable () -> Unit = {},
    bottomBar: @Composable () -> Unit = {},
    snackbarHost: @Composable () -> Unit = {},
    floatingActionButton: @Composable () -> Unit = {},
    floatingActionButtonPosition: FabPosition = FabPosition.End,
    containerColor: Color = MaterialTheme.colorScheme.background,
    contentColor: Color = contentColorFor(containerColor),
    contentWindowInsets: WindowInsets = ScaffoldDefaults.contentWindowInsets,
    content: @Composable (PaddingValues) -> Unit
) {
```

SCAFFOLD TOP BAR

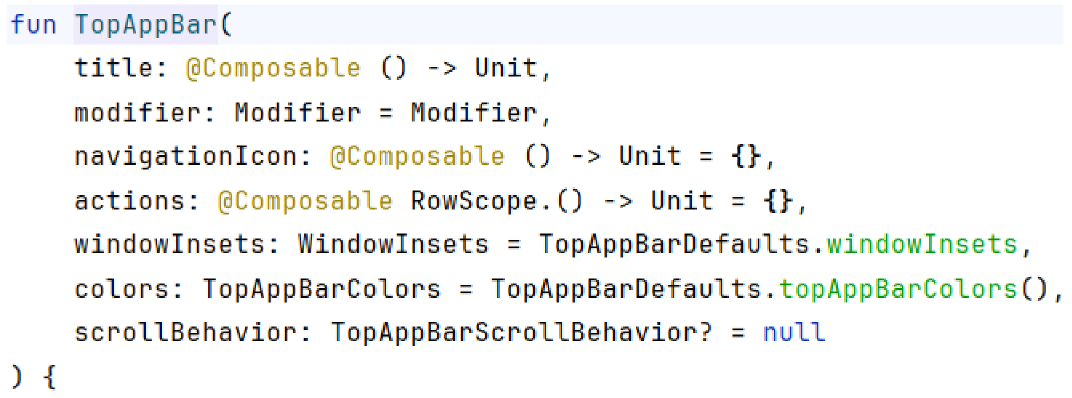
Now we want to put in a top bar in the scaffold

Notice it is experimental

The IDE will ask you to opt in and add the annotation



```
@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun FirstScaffold() {
    Scaffold(
        topBar = {
            TopAppBar(title = { "SomeTitle maybe" })
        }
    ) { innerPadding ->
        Box(modifier = Modifier.padding(innerPadding)) {
            // some composable is called here
        }
    }
}
```

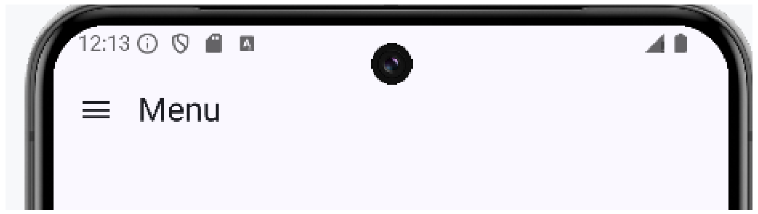


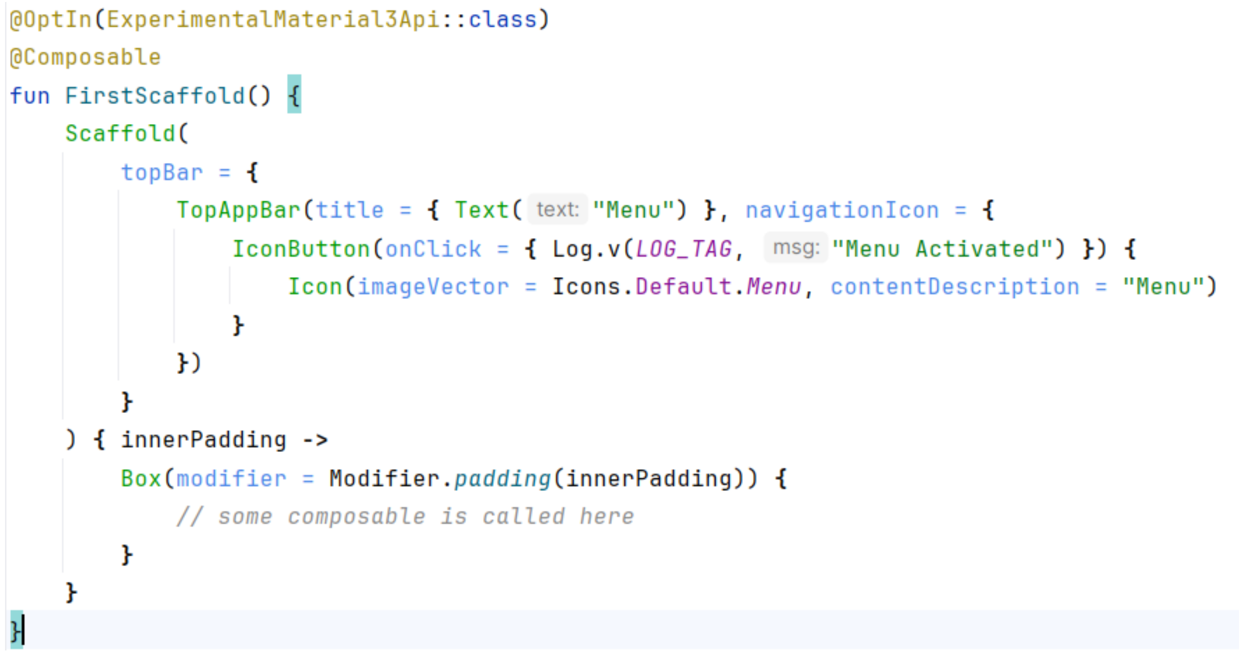
```
fun TopAppBar(
    title: @Composable () -> Unit,
    modifier: Modifier = Modifier,
    navigationIcon: @Composable () -> Unit = {},
    actions: @Composable RowScope.() -> Unit = {},
    windowInsets: WindowInsets = TopAppBarDefaults.windowInsets,
    colors: TopAppBarColors = TopAppBarDefaults.topAppBarColors(),
    scrollBehavior: TopAppBarScrollBehavior? = null
) {
```

TOP BAR

The Navigation composable is usually some icon placed in the left of the top bar

Let us insert a menu as this navigation Icon, and a title say “Menu”



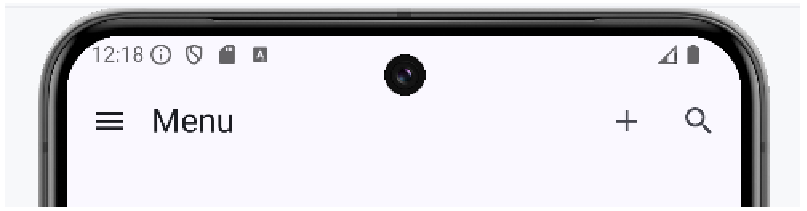


```
@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun FirstScaffold() {
    Scaffold(
        topBar = {
            TopAppBar(title = { Text( text: "Menu") }, navigationIcon = {
                IconButton(onClick = { Log.v(LOG_TAG,  msg: "Menu Activated") }) {
                    Icon(imageVector = Icons.Default.Menu, contentDescription = "Menu")
                }
            })
        }
    ) { innerPadding ->
        Box(modifier = Modifier.padding(innerPadding)) {
            // some composable is called here
        }
    }
}
```

ACTIONS (FURTHER BUTTONS)

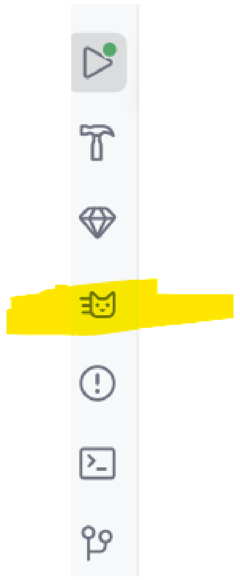
Maybe we want some extra actions in the form og Icon buttons

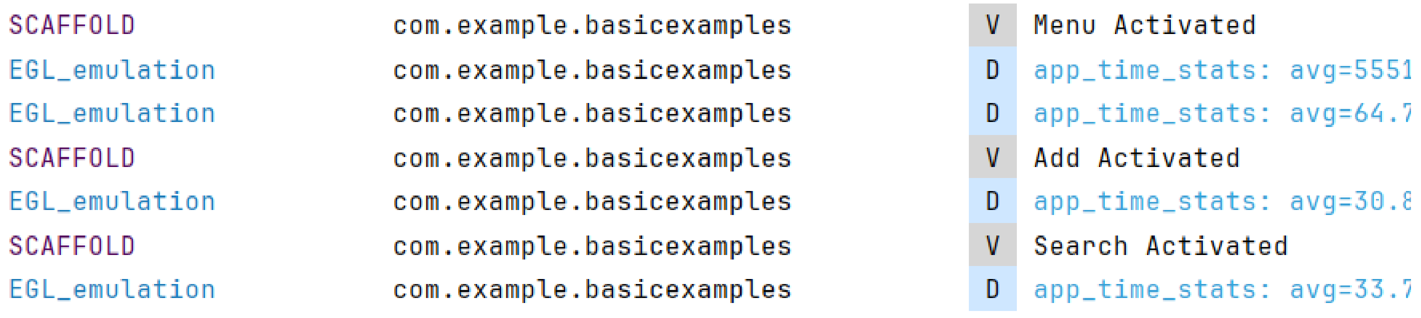
```
@Composable
fun FirstScaffold() {
    Scaffold(
        topBar = {
            TopAppBar(title = { Text( text: "Menu") }, navigationIcon = {
                IconButton(onClick = { Log.v(LOG_TAG,  msg: "Menu Activated") }) {
                    Icon(imageVector = Icons.Default.Menu, contentDescription = "Menu")
                }
            },
                actions = {
                    IconButton(onClick = { Log.v(LOG_TAG,  msg: "Add Activated") }) {
                        Icon(imageVector = Icons.Default.Add, contentDescription = "Add")
                    }
                    IconButton(onClick = { Log.v(LOG_TAG,  msg: "Search Activated") }) {
                        Icon(imageVector = Icons.Default.Search, contentDescription = "Search")
                    }
                })
        }
    ) { innerPadding ->
```



ONCLICK

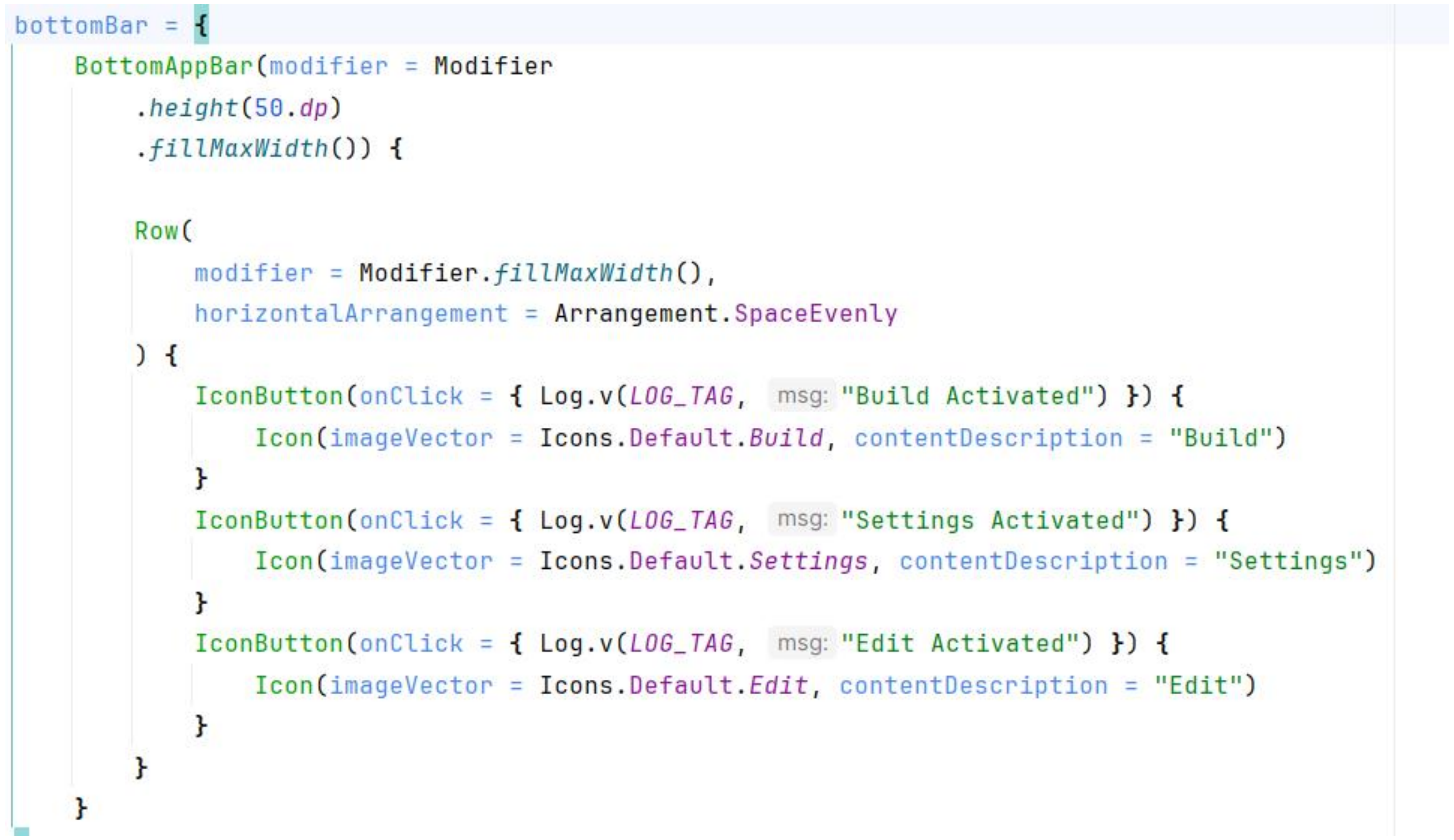
For the moment let us just log the click in the developer log called logcat



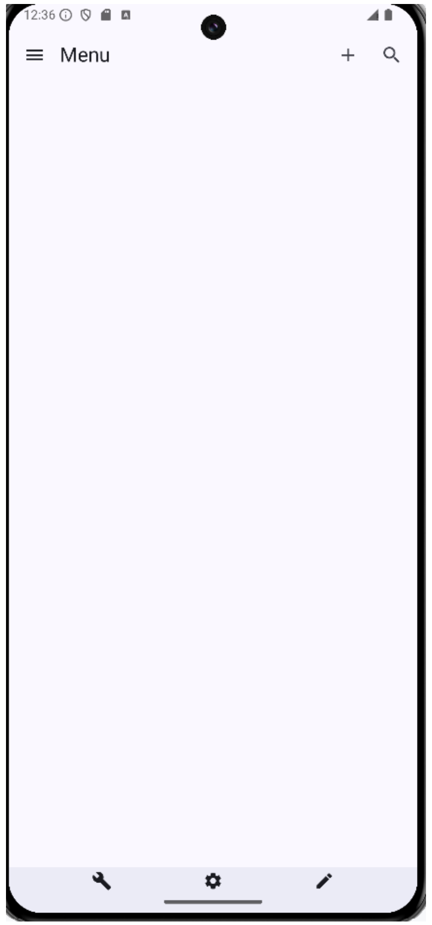


```
SCAFFOLD          com.example.basicexamples     V  Menu Activated
EGL_emulation     com.example.basicexamples     D  app_time_stats: avg=5551
EGL_emulation     com.example.basicexamples     D  app_time_stats: avg=64.7
SCAFFOLD          com.example.basicexamples     V  Add Activated
EGL_emulation     com.example.basicexamples     D  app_time_stats: avg=30.8
SCAFFOLD          com.example.basicexamples     V  Search Activated
EGL_emulation     com.example.basicexamples     D  app_time_stats: avg=33.7
```

BOTTOM BAR

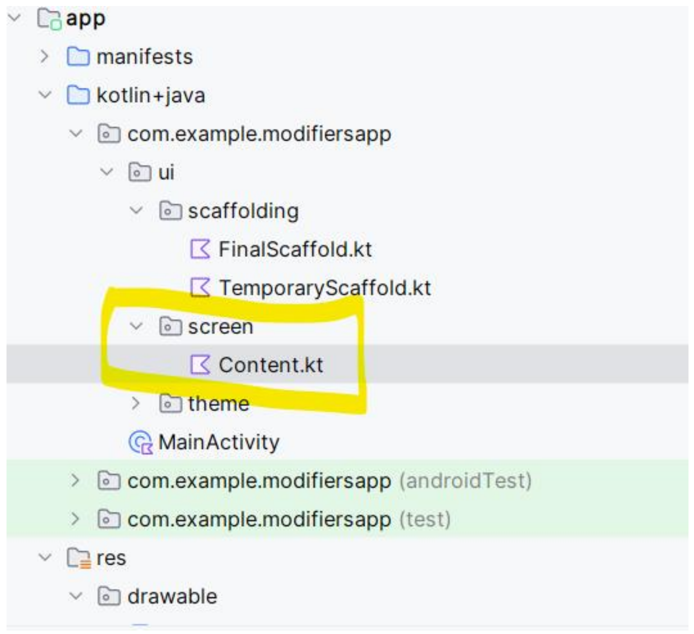


```
bottomBar = {
    BottomAppBar(modifier = Modifier
        .height(50.dp)
        .fillMaxWidth()) {

        Row(
            modifier = Modifier.fillMaxWidth(),
            horizontalArrangement = Arrangement.SpaceEvenly
        ) {
            IconButton(onClick = { Log.v(LOG_TAG,  msg: "Build Activated") }) {
                Icon(imageVector = Icons.Default.Build, contentDescription = "Build")
            }
            IconButton(onClick = { Log.v(LOG_TAG,  msg: "Settings Activated") }) {
                Icon(imageVector = Icons.Default.Settings, contentDescription = "Settings")
            }
            IconButton(onClick = { Log.v(LOG_TAG,  msg: "Edit Activated") }) {
                Icon(imageVector = Icons.Default.Edit, contentDescription = "Edit")
            }
        }
    }
```

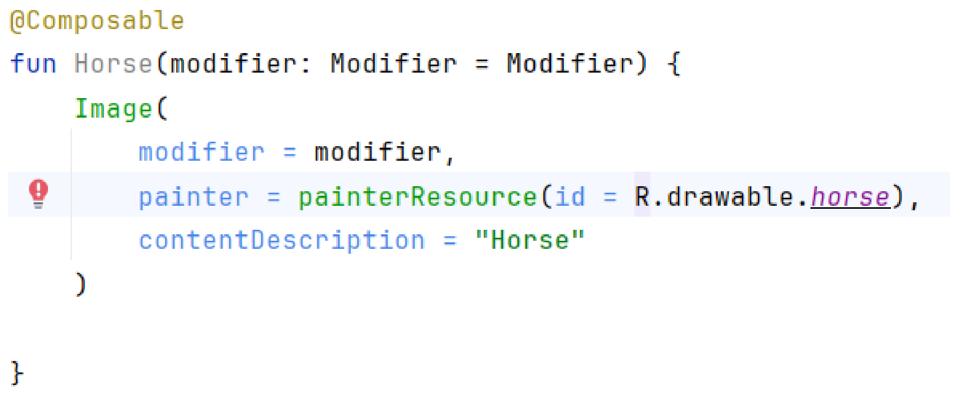


CONTENT IN THE SCAFFOLD

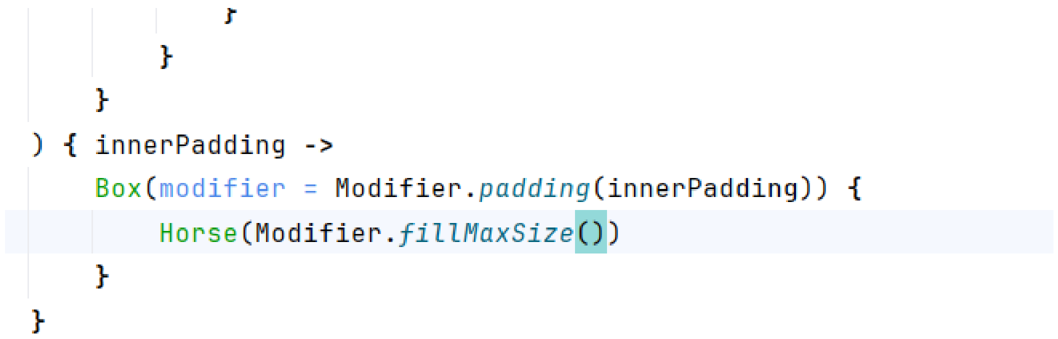
We define a composable for the main content



```
app
  manifests
  kotlin+java
    com.example.modifiersapp
      ui
        scaffolding
          FinalScaffold.kt
          TemporaryScaffold.kt
        screen
          Content.kt
        theme
      MainActivity
    com.example.modifiersapp (androidTest)
    com.example.modifiersapp (test)
  res
    drawable
```

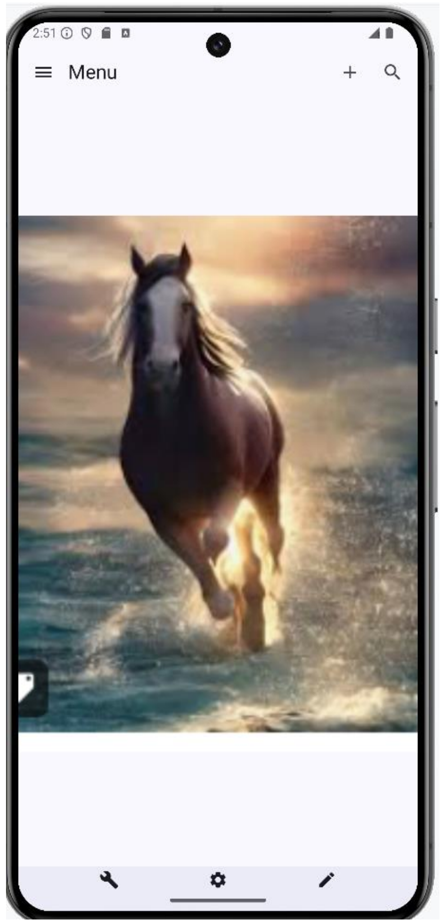


```kotlin
@Composable
fun Horse(modifier: Modifier = Modifier) {
    Image(
        modifier = modifier,
        painter = painterResource(id = R.drawable.horse),
        contentDescription = "Horse"
    )

}
```



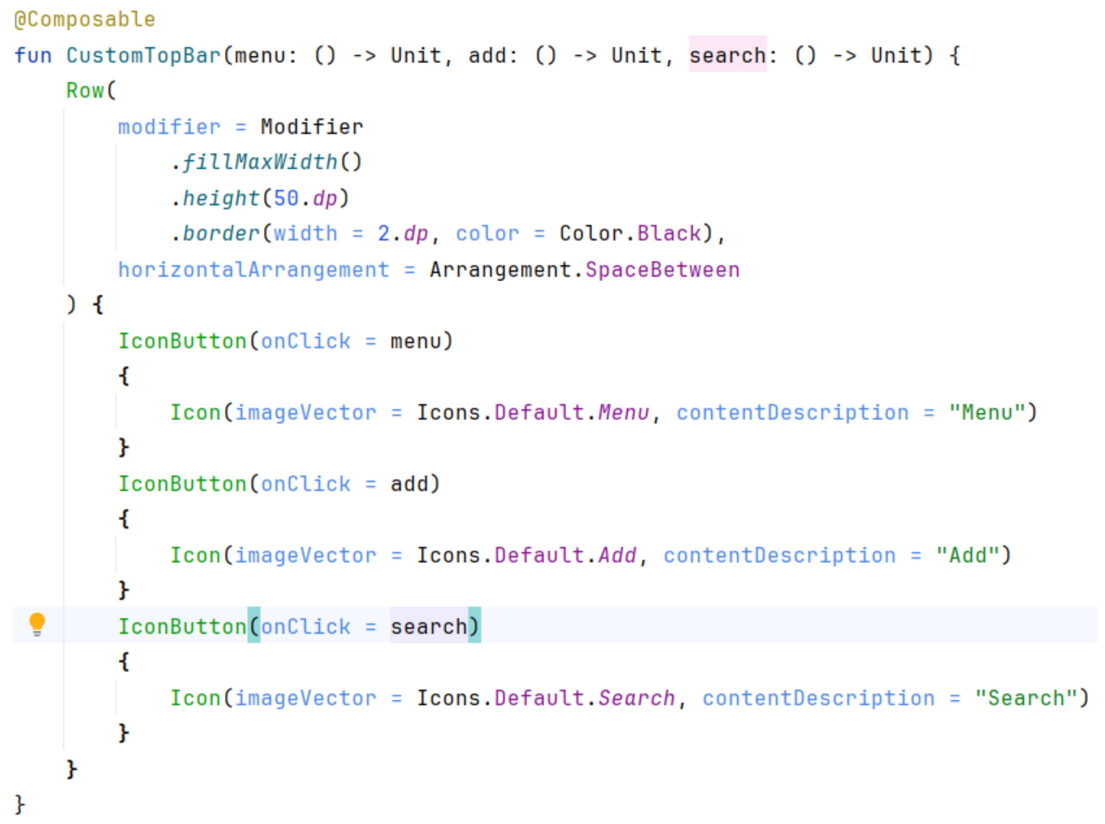
```kotlin
                ,
            }
        }
) { innerPadding ->
    Box(modifier = Modifier.padding(innerPadding)) {
        Horse(Modifier.fillMaxSize())
    }
}
```

CONTENT IN THE SCAFFOLD

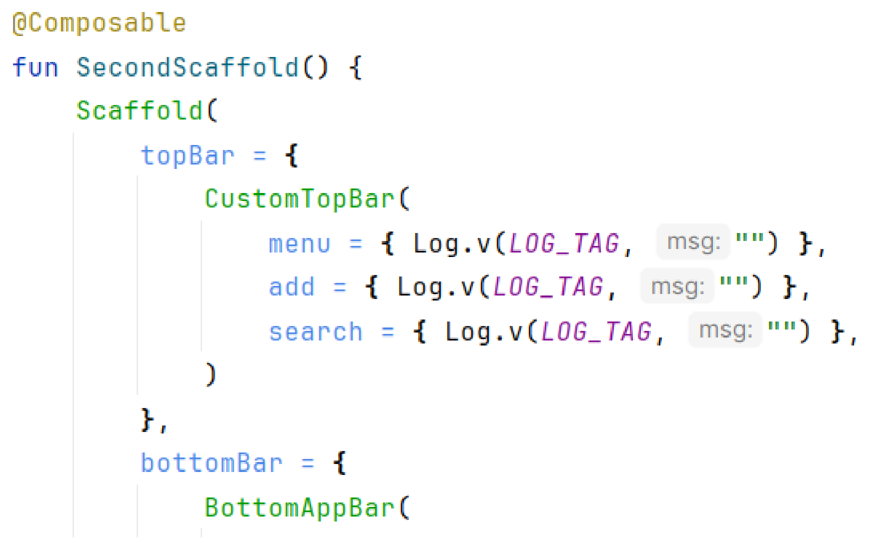


CUSTOM MENU BARS IF YOU WANT

You can of course make your own top bar if you want



```kotlin
@Composable
fun CustomTopBar(menu: () -> Unit, add: () -> Unit, search: () -> Unit) {
    Row(
        modifier = Modifier
            .fillMaxWidth()
            .height(50.dp)
            .border(width = 2.dp, color = Color.Black),
        horizontalArrangement = Arrangement.SpaceBetween
    ) {
        IconButton(onClick = menu)
        {
            Icon(imageVector = Icons.Default.Menu, contentDescription = "Menu")
        }
        IconButton(onClick = add)
        {
            Icon(imageVector = Icons.Default.Add, contentDescription = "Add")
        }
        IconButton(onClick = search)
        {
            Icon(imageVector = Icons.Default.Search, contentDescription = "Search")
        }
    }
}
```

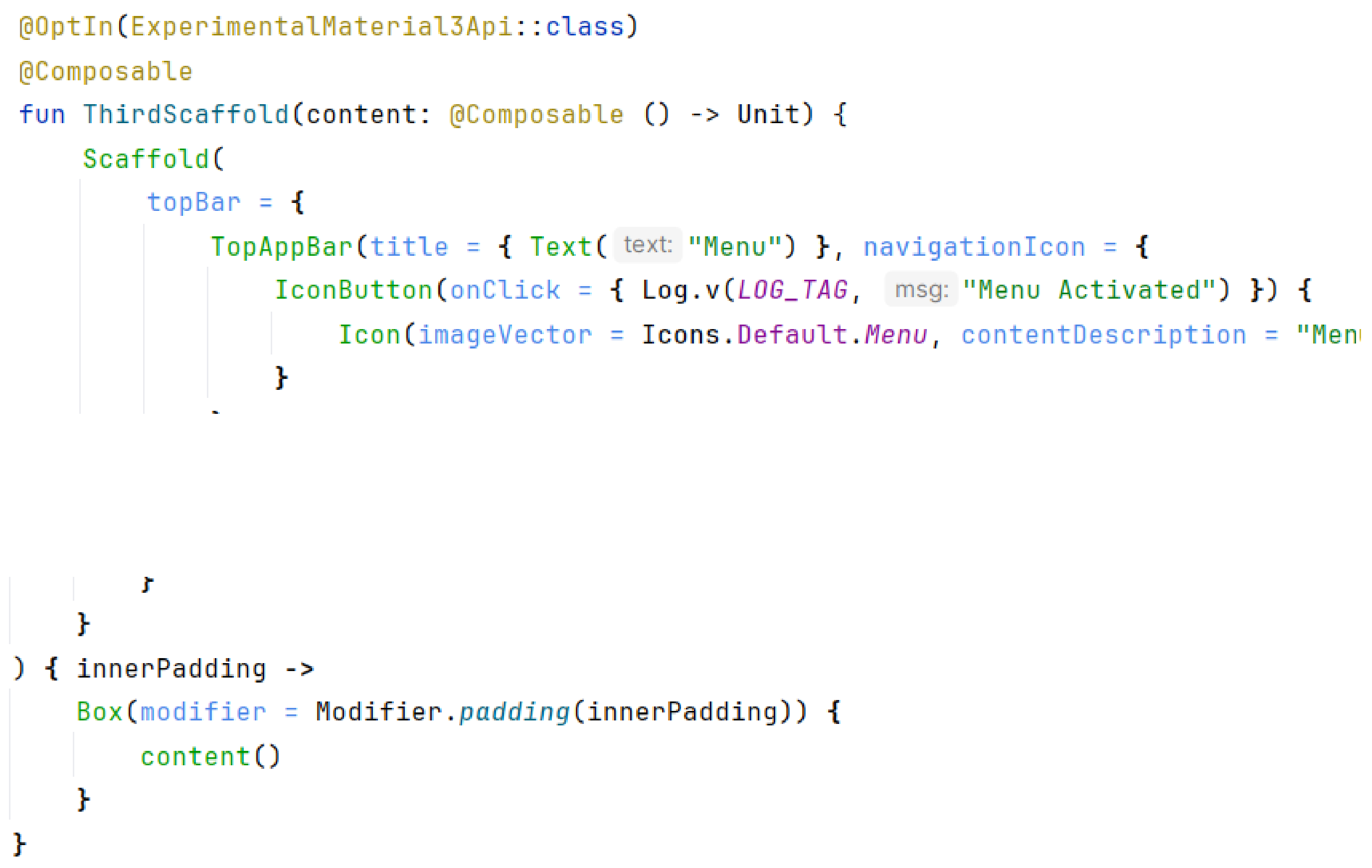


```kotlin
@Composable
fun SecondScaffold() {
    Scaffold(
        topBar = {
            CustomTopBar(
                menu = { Log.v(LOG_TAG, msg: "") },
                add = { Log.v(LOG_TAG, msg: "") },
                search = { Log.v(LOG_TAG, msg: "") },
            )
        },
        bottomBar = {
            BottomAppBar(
```

COMPOSABLE WITH COMPOSABLE PARAMETER

Sometimes it is desirable to implement a composable that can take another composable as parameter. Typically, in scaffolding composables.

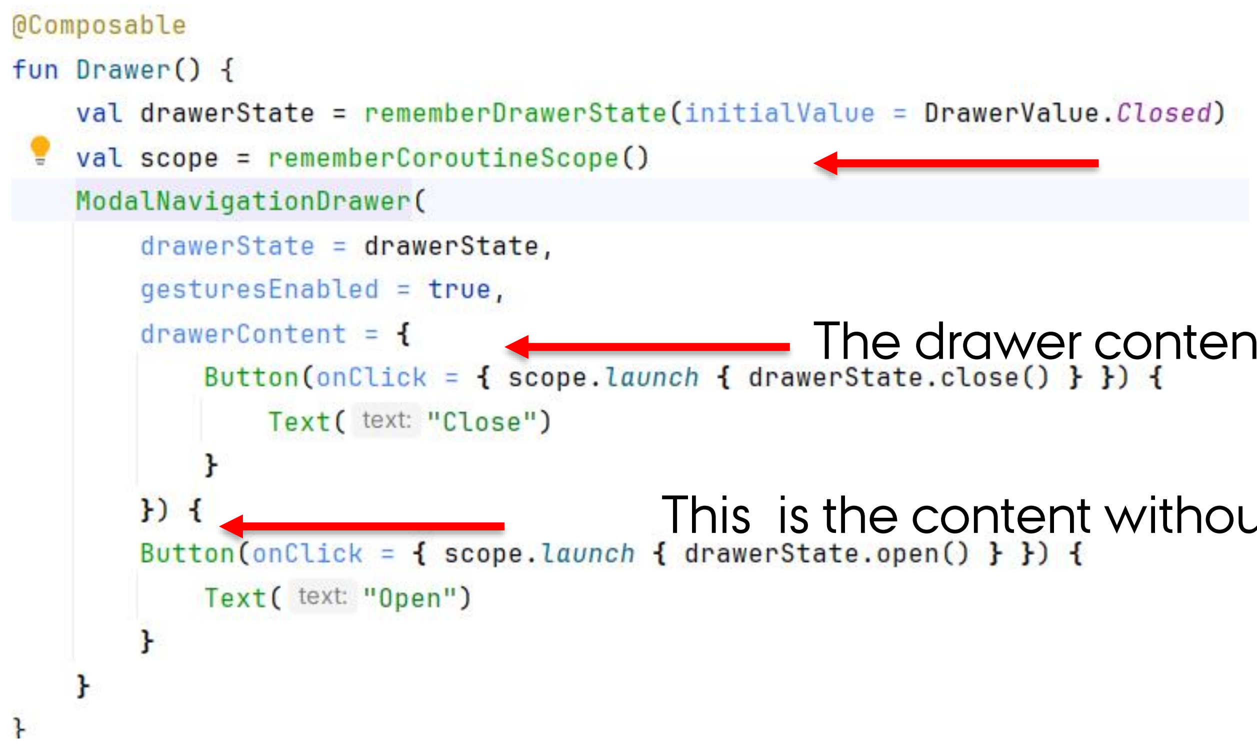
COMPOSABLE AS A PARAMETER



```kotlin
@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun ThirdScaffold(content: @Composable () -> Unit) {
    Scaffold(
        topBar = {
            TopAppBar(title = { Text( text: "Menu") }, navigationIcon = {
                IconButton(onClick = { Log.v(LOG_TAG,  msg: "Menu Activated") }) {
                    Icon(imageVector = Icons.Default.Menu, contentDescription = "Men
                }
```

```kotlin
    }
) { innerPadding ->
    Box(modifier = Modifier.padding(innerPadding)) {
        content()
    }
}
```

A SPECIAL COMPOSABLE
MODALNAVIGATIONDRAWER



```
@Composable
fun Drawer() {
    val drawerState = rememberDrawerState(initialValue = DrawerValue.Closed)
    val scope = rememberCoroutineScope()
    ModalNavigationDrawer(
        drawerState = drawerState,
        gesturesEnabled = true,
        drawerContent = {
            Button(onClick = { scope.launch { drawerState.close() } }) {
                Text( text: "Close")
            }
        }) {
        Button(onClick = { scope.launch { drawerState.open() } }) {
            Text( text: "Open")
        }
    }
}
```

The drawer content is where we typically put menus

This is the content without open drawer

THE DRAWER STATE

```
val drawerState = rememberDrawerState(initialValue = DrawerValue.Closed)
```

An object for holding the state of the drawer. When it changes the ModalNavigationDrawer Is re-rendered

THE COROUTINE SCOPE

The coroutine is somewhat special. It is an object that enables asynchronous calls

```
val scope = rememberCoroutineScope()
```

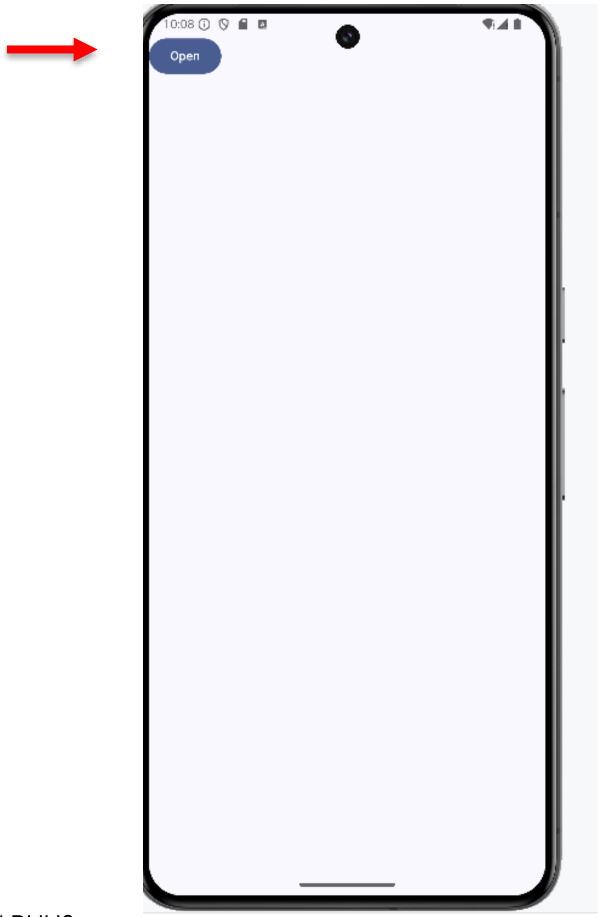
This statement creates an object that is accessible in the current composable and is used for running asynchronous calls.

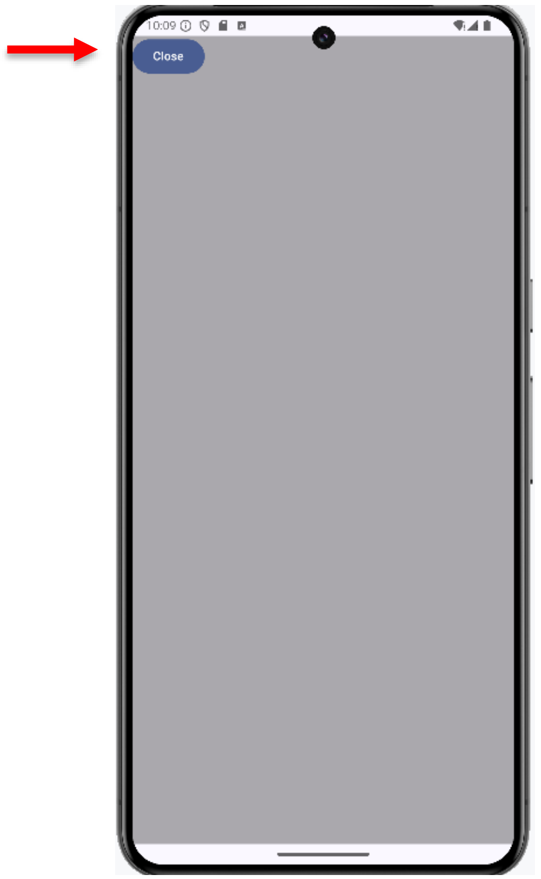
This can be done by the launch method on the scope

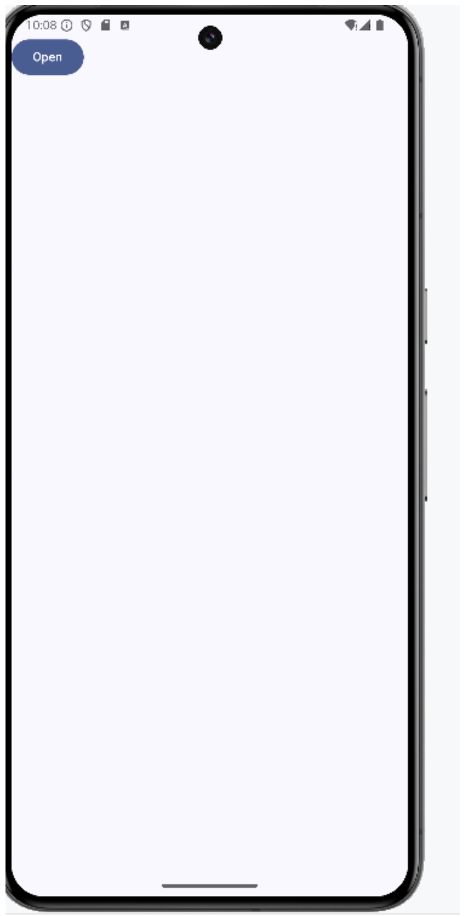
```
scope.launch {
   drawerState.open()
}
```

This is necessary because drawerstate .open() is a suspend function and MUST be executed in a coroutine scope.

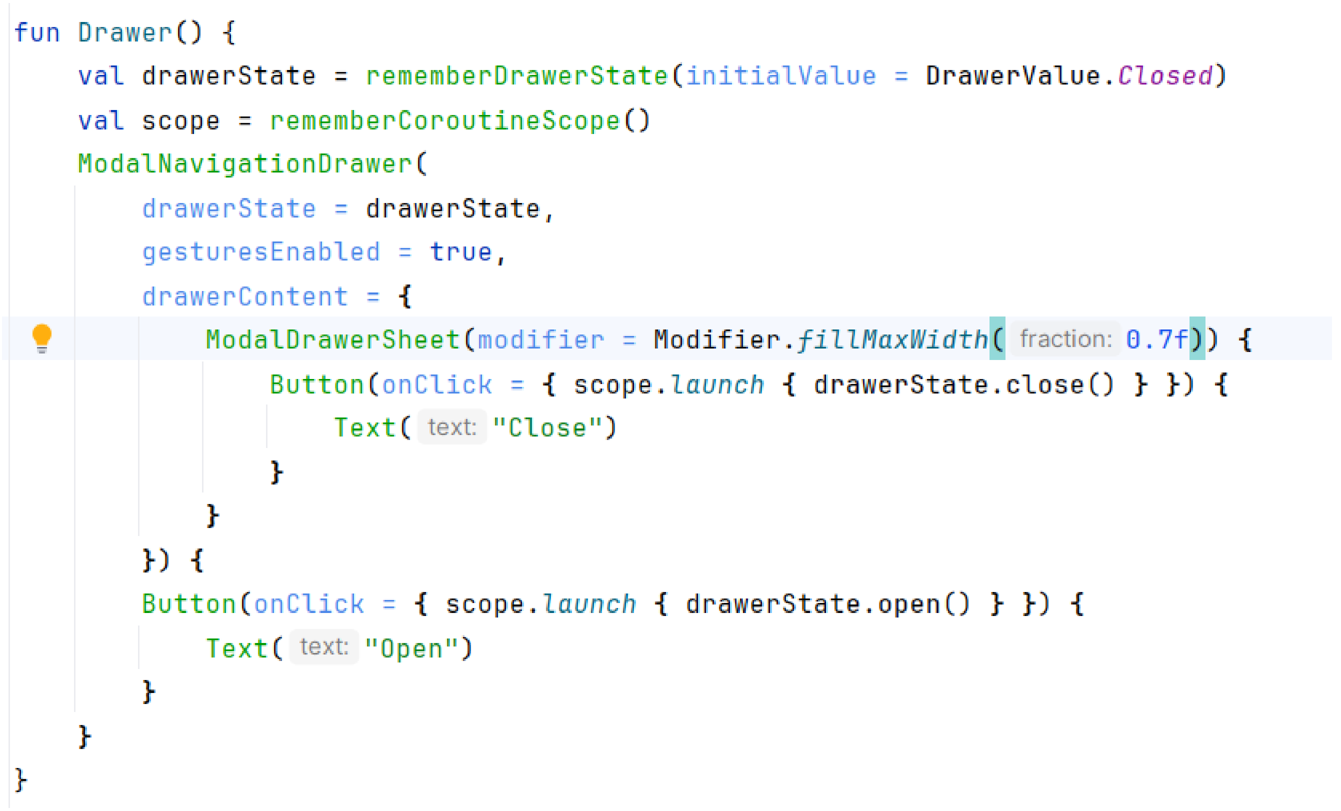
THE DRAWER



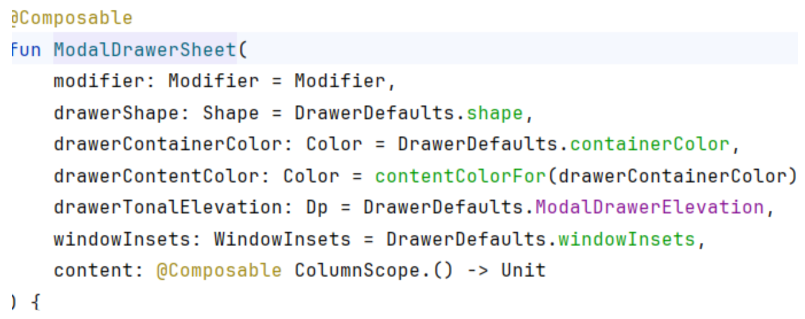




DRAWER SHEET

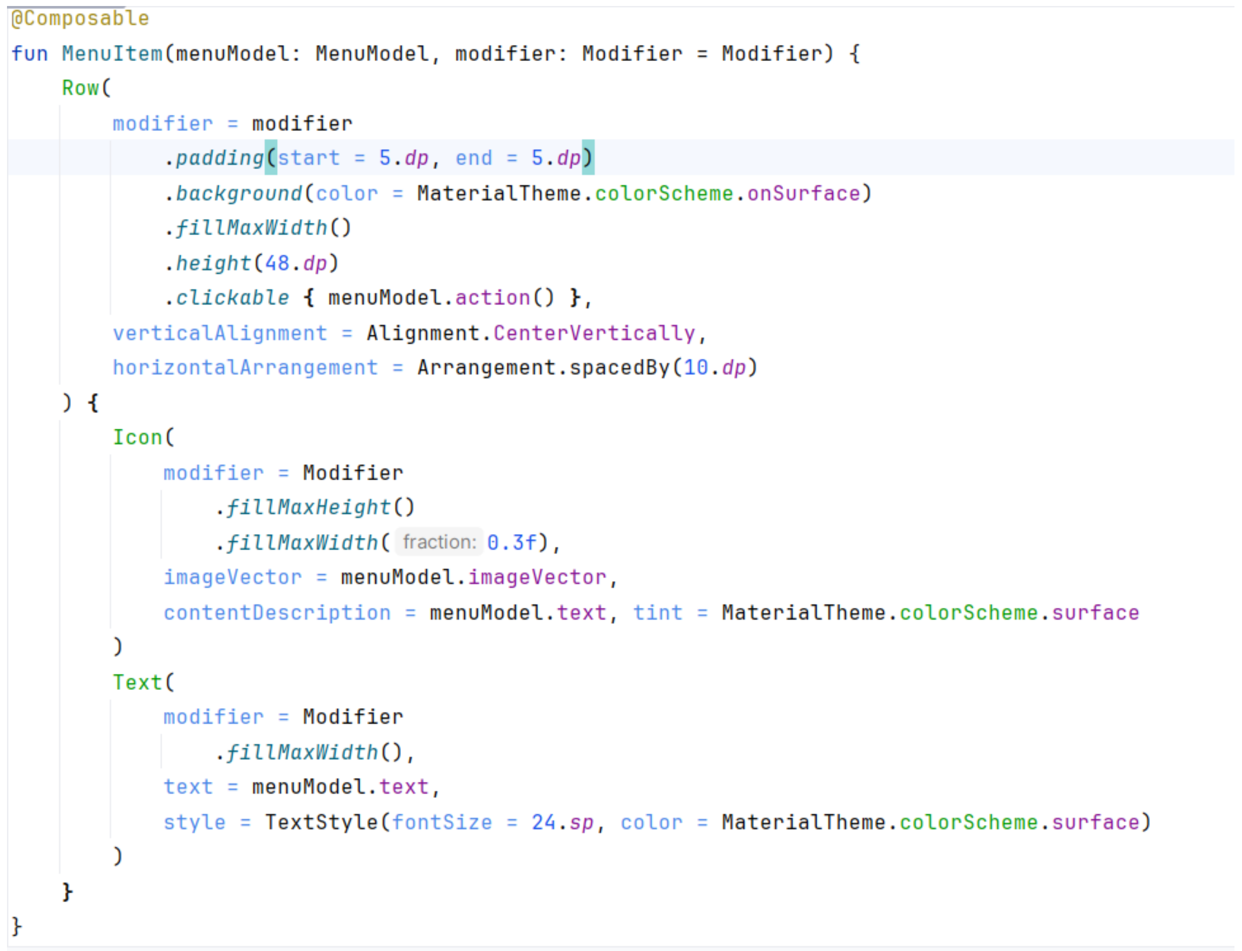


```kotlin
fun Drawer() {
    val drawerState = rememberDrawerState(initialValue = DrawerValue.Closed)
    val scope = rememberCoroutineScope()
    ModalNavigationDrawer(
        drawerState = drawerState,
        gesturesEnabled = true,
        drawerContent = {
            ModalDrawerSheet(modifier = Modifier.fillMaxWidth( fraction: 0.7f)) {
                Button(onClick = { scope.launch { drawerState.close() } }) {
                    Text( text: "Close")
                }
            }
        }) {
        Button(onClick = { scope.launch { drawerState.open() } }) {
            Text( text: "Open")
        }
    }
}
```



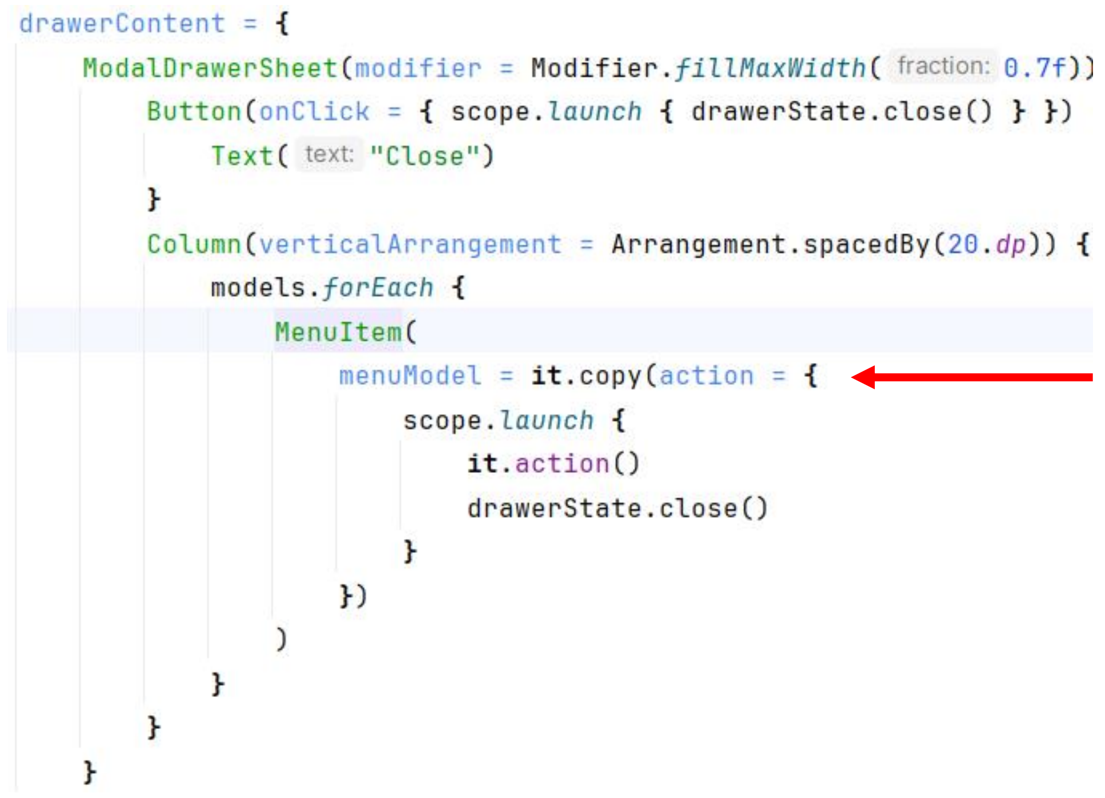
```kotlin
@Composable
fun ModalDrawerSheet(
    modifier: Modifier = Modifier,
    drawerShape: Shape = DrawerDefaults.shape,
    drawerContainerColor: Color = DrawerDefaults.containerColor,
    drawerContentColor: Color = contentColorFor(drawerContainerColor)
    drawerTonalElevation: Dp = DrawerDefaults.ModalDrawerElevation,
    windowInsets: WindowInsets = DrawerDefaults.windowInsets,
    content: @Composable ColumnScope.() -> Unit
) {
```

PUT IN SOME MENUS



```kotlin
@Composable
fun MenuItem(menuModel: MenuModel, modifier: Modifier = Modifier) {
    Row(
        modifier = modifier
            .padding(start = 5.dp, end = 5.dp)
            .background(color = MaterialTheme.colorScheme.onSurface)
            .fillMaxWidth()
            .height(48.dp)
            .clickable { menuModel.action() },
        verticalAlignment = Alignment.CenterVertically,
        horizontalArrangement = Arrangement.spacedBy(10.dp)
    ) {
        Icon(
            modifier = Modifier
                .fillMaxHeight()
                .fillMaxWidth( fraction: 0.3f),
            imageVector = menuModel.imageVector,
            contentDescription = menuModel.text, tint = MaterialTheme.colorScheme.surface
        )
        Text(
            modifier = Modifier
                .fillMaxWidth(),
            text = menuModel.text,
            style = TextStyle(fontSize = 24.sp, color = MaterialTheme.colorScheme.surface)
        )
    }
}
```

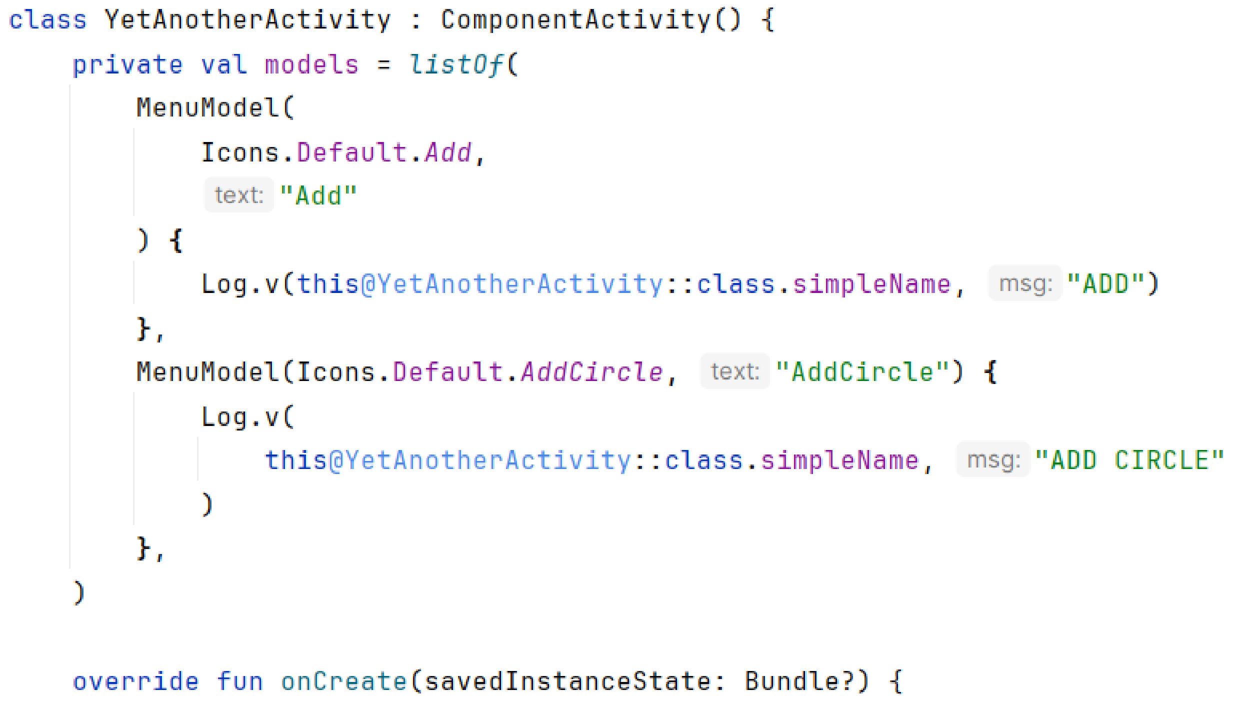
IN THE DRAWER



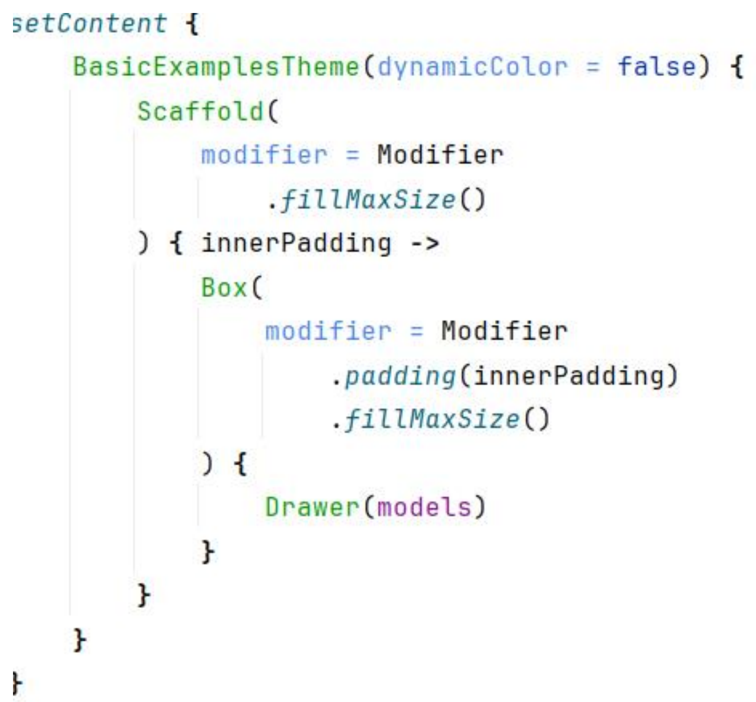
```
drawerContent = {
    ModalDrawerSheet(modifier = Modifier.fillMaxWidth( fraction: 0.7f))
        Button(onClick = { scope.launch { drawerState.close() } })
            Text( text: "Close")
        }
        Column(verticalArrangement = Arrangement.spacedBy(20.dp)) {
            models.forEach {
                MenuItem(
                    menuModel = it.copy(action = {
                        scope.launch {
                            it.action()
                            drawerState.close()
                        }
                    })
                )
            }
        }
    }
```

A trick

ACTIVITY

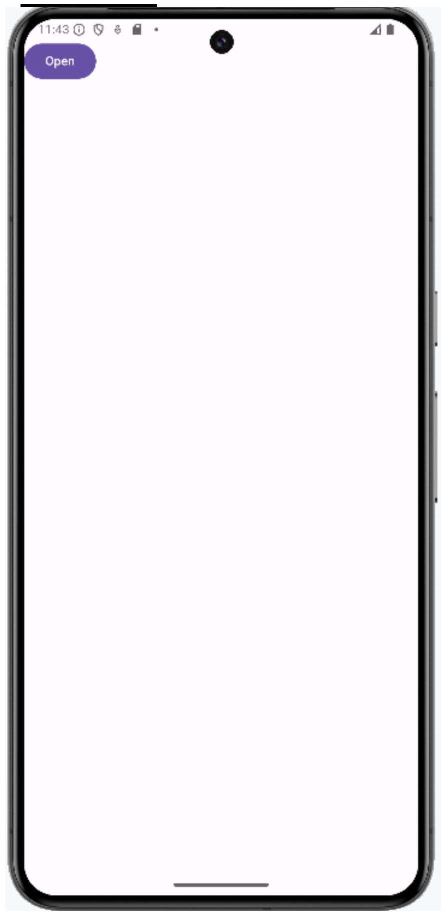


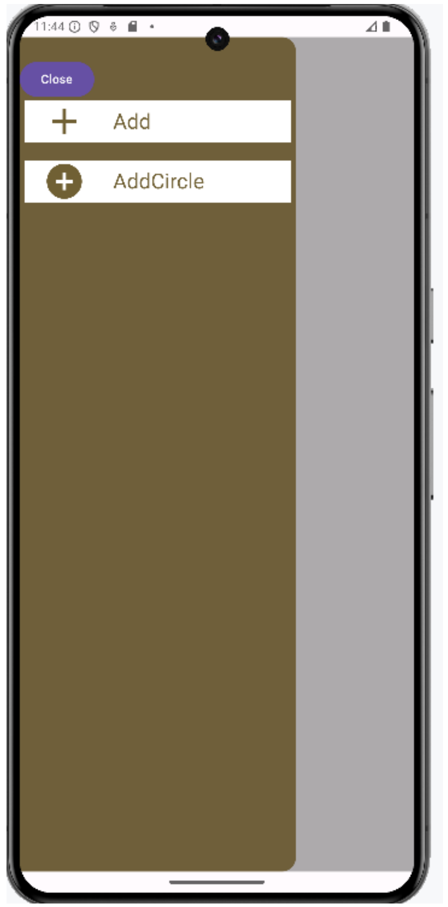
```
class YetAnotherActivity : ComponentActivity() {
    private val models = listOf(
        MenuModel(
            Icons.Default.Add,
            text: "Add"
        ) {
            Log.v(this@YetAnotherActivity::class.simpleName, msg: "ADD")
        },
        MenuModel(Icons.Default.AddCircle, text: "AddCircle") {
            Log.v(
                this@YetAnotherActivity::class.simpleName, msg: "ADD CIRCLE"
            )
        },
    )

    override fun onCreate(savedInstanceState: Bundle?) {
```



```
setContent {
    BasicExamplesTheme(dynamicColor = false) {
        Scaffold(
            modifier = Modifier
                .fillMaxSize()
        ) { innerPadding ->
            Box(
                modifier = Modifier
                    .padding(innerPadding)
                    .fillMaxSize()
            ) {
                Drawer(models)
            }
        }
    }
}
```

IN ACTION

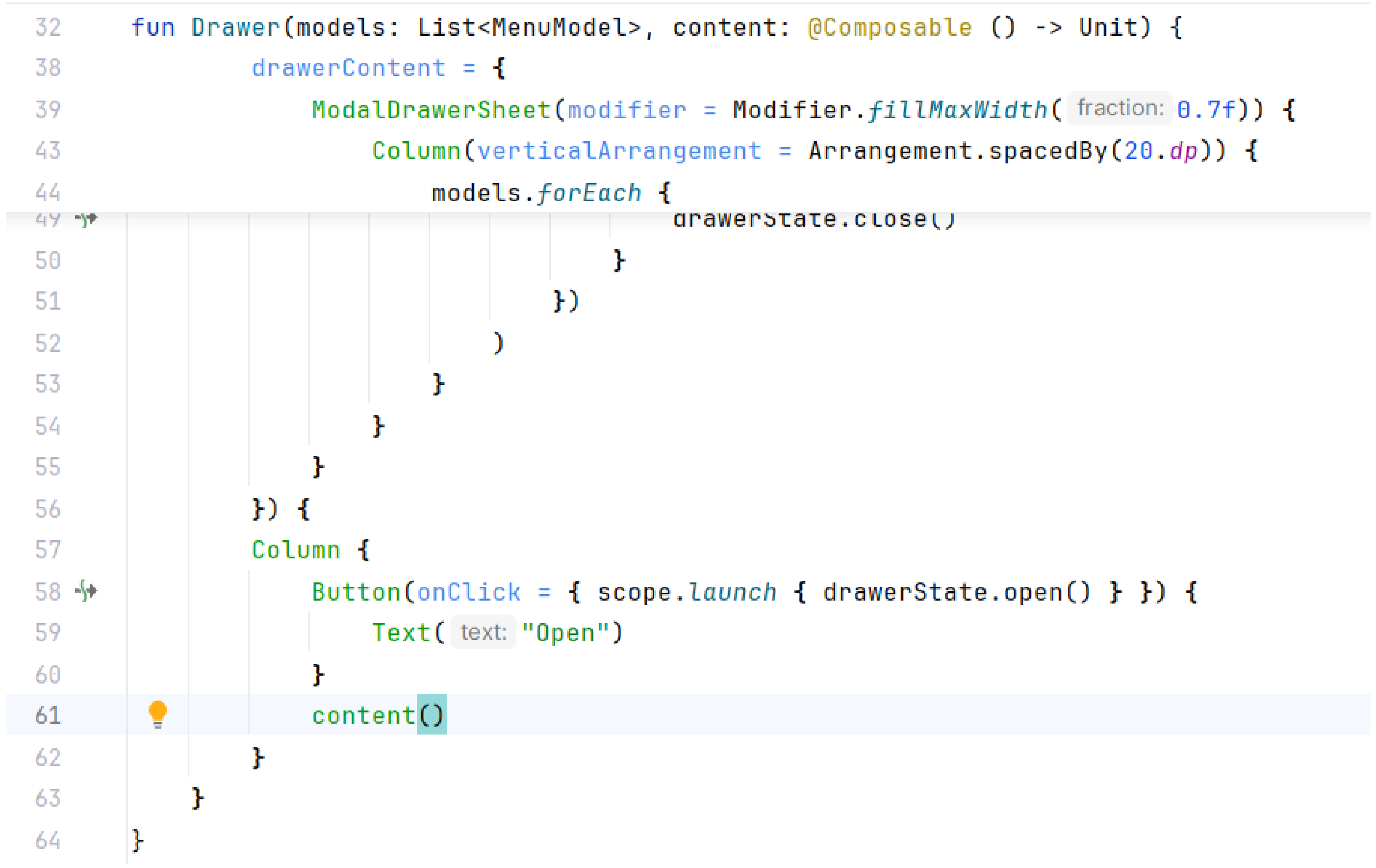




```
YetAnotherActivity    com.example.basicexamples    V  ADD
EGL_emulation         com.example.basicexamples    D  app_time_stats:
YetAnotherActivity    com.example.basicexamples    V  ADD CIRCLE
EGL_emulation         com.example.basicexamples    D  app_time_stats:
```

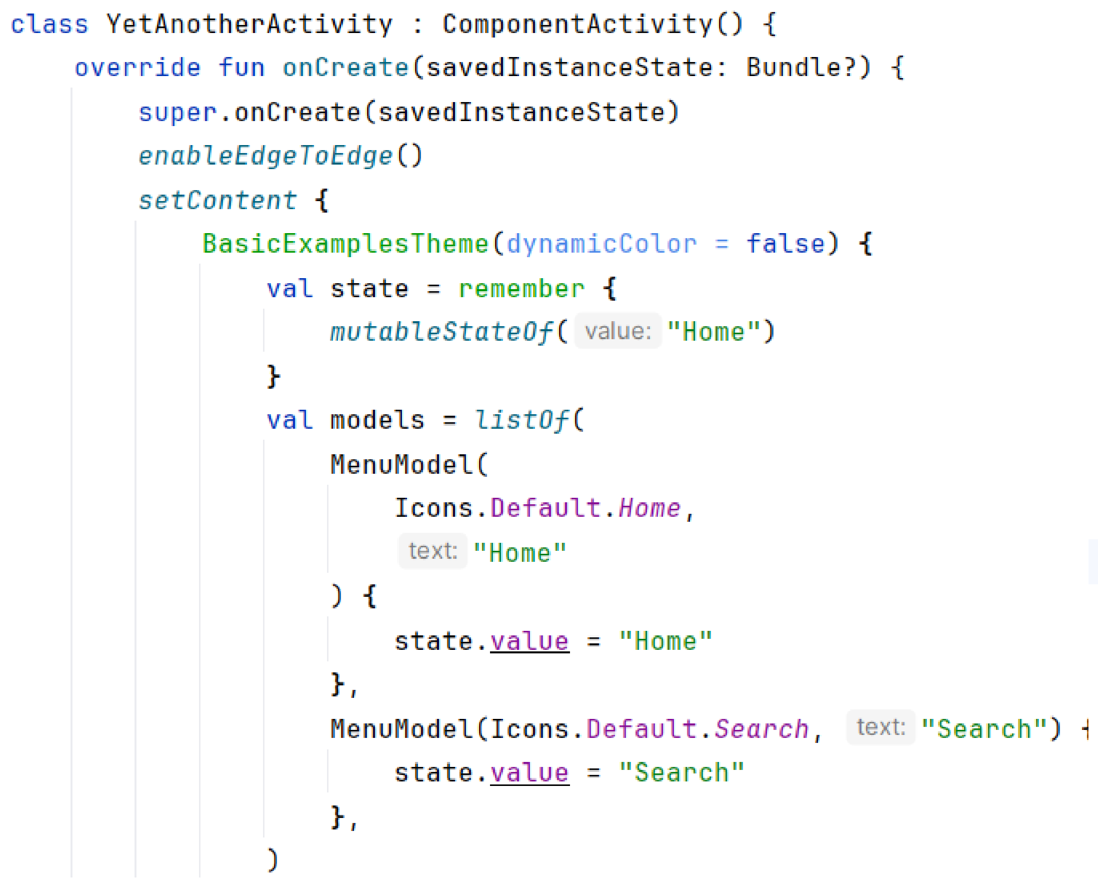
MAKING THE DRAWER CONTENT A PARAMETER

If we imagine that the content can change->

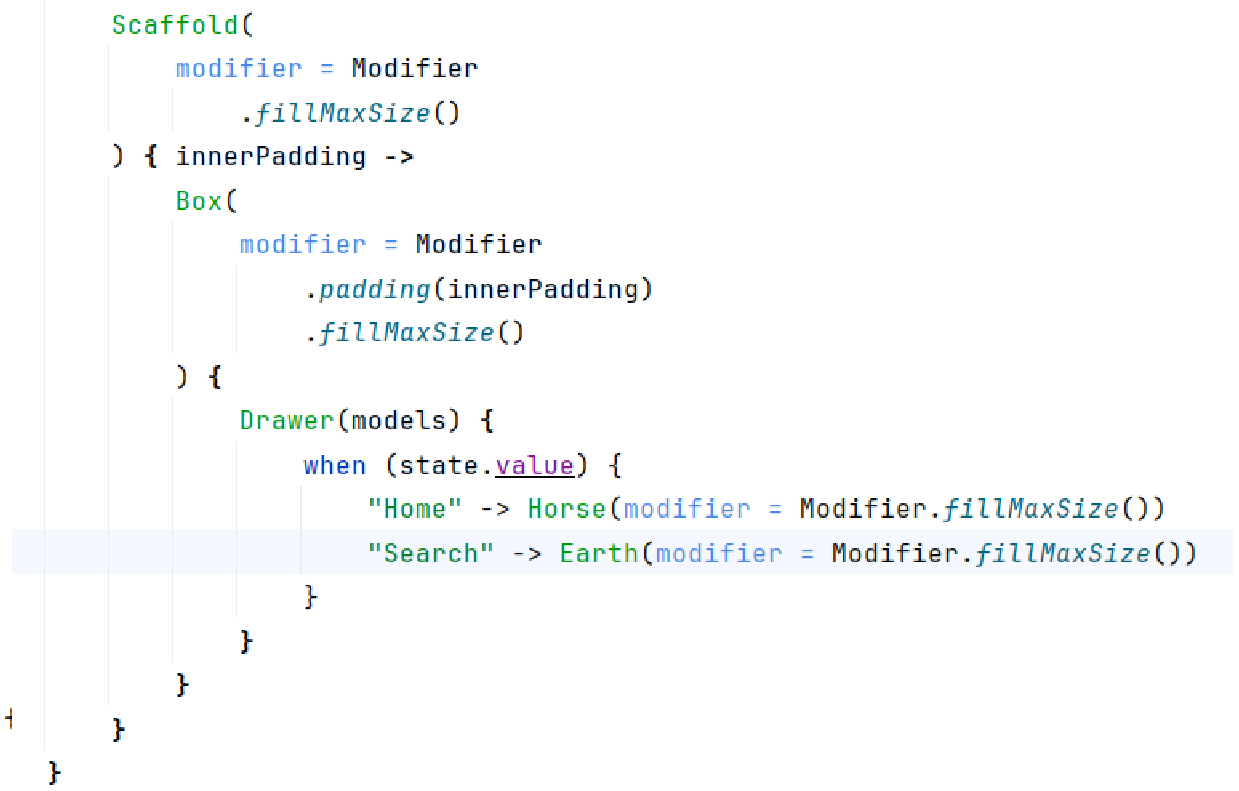


```
fun Drawer(models: List<MenuModel>, content: @Composable () -> Unit) {
        drawerContent = {
            ModalDrawerSheet(modifier = Modifier.fillMaxWidth( fraction: 0.7f)) {
                Column(verticalArrangement = Arrangement.spacedBy(20.dp)) {
                    models.forEach {
                                    drawerState.close()
                                }
                            })
                        )
                    }
                }
            }
        }) {
        Column {
            Button(onClick = { scope.launch { drawerState.open() } }) {
                Text( text: "Open")
            }
            content()
        }
    }
}
```

CONTENT CHANGE BY SIMPLE! STATE

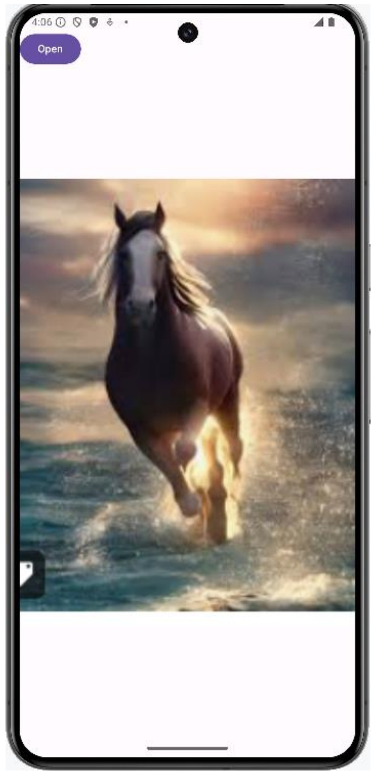


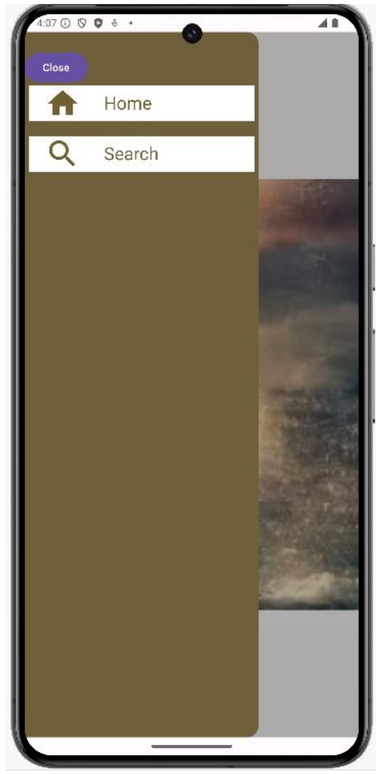
```kotlin
class YetAnotherActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        enableEdgeToEdge()
        setContent {
            BasicExamplesTheme(dynamicColor = false) {
                val state = remember {
                    mutableStateOf( value: "Home")
                }
                val models = listOf(
                    MenuModel(
                        Icons.Default.Home,
                        text: "Home"
                    ) {
                        state.value = "Home"
                    },
                    MenuModel(Icons.Default.Search,  text: "Search") {
                        state.value = "Search"
                    },
                )
```

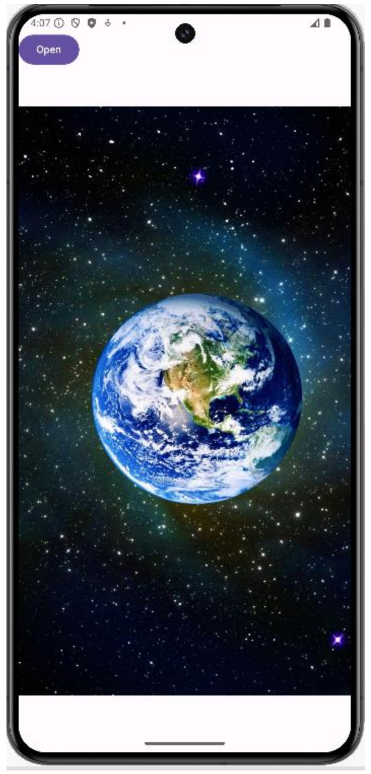


```kotlin
                Scaffold(
                    modifier = Modifier
                        .fillMaxSize()
                ) { innerPadding ->
                    Box(
                        modifier = Modifier
                            .padding(innerPadding)
                            .fillMaxSize()
                    ) {
                        Drawer(models) {
                            when (state.value) {
                                "Home" -> Horse(modifier = Modifier.fillMaxSize())
                                "Search" -> Earth(modifier = Modifier.fillMaxSize())
                            }
                        }
                    }
                }
            }
```

CONTENT CHANGE BY SIMPLE STATE







AARHUS
UNIVERSITY