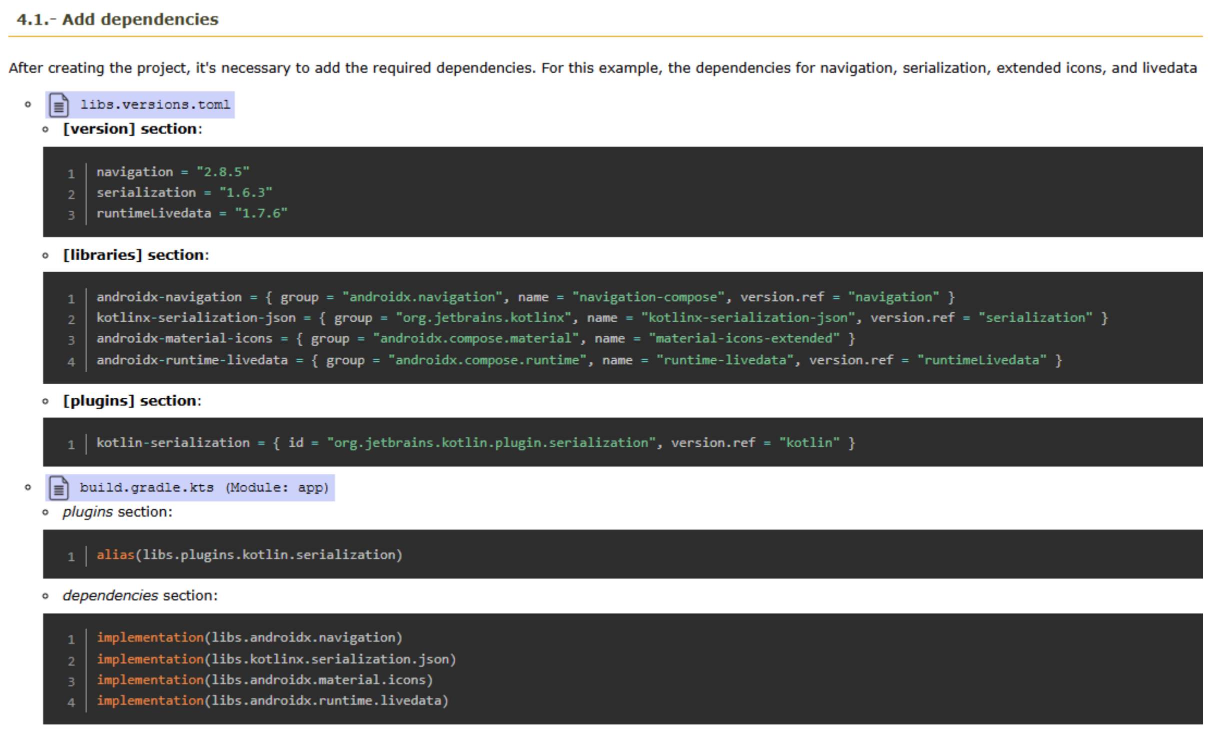
Varias dependencias juntas(faltan room y data store)
Ud9.4



4.1.- Add dependencies

After creating the project, it's necessary to add the required dependencies. For this example, the dependencies for navigation, serialization, extended icons, and livedata

- `libs.versions.toml`
  - **[version] section**:

```
navigation = "2.8.5"
serialization = "1.6.3"
runtimeLivedata = "1.7.6"
```

  - **[libraries] section**:

```
androidx-navigation = { group = "androidx.navigation", name = "navigation-compose", version.ref = "navigation" }
kotlinx-serialization-json = { group = "org.jetbrains.kotlinx", name = "kotlinx-serialization-json", version.ref = "serialization" }
androidx-material-icons = { group = "androidx.compose.material", name = "material-icons-extended" }
androidx-runtime-livedata = { group = "androidx.compose.runtime", name = "runtime-livedata", version.ref = "runtimeLivedata" }
```

  - **[plugins] section**:

```
kotlin-serialization = { id = "org.jetbrains.kotlin.plugin.serialization", version.ref = "kotlin" }
```

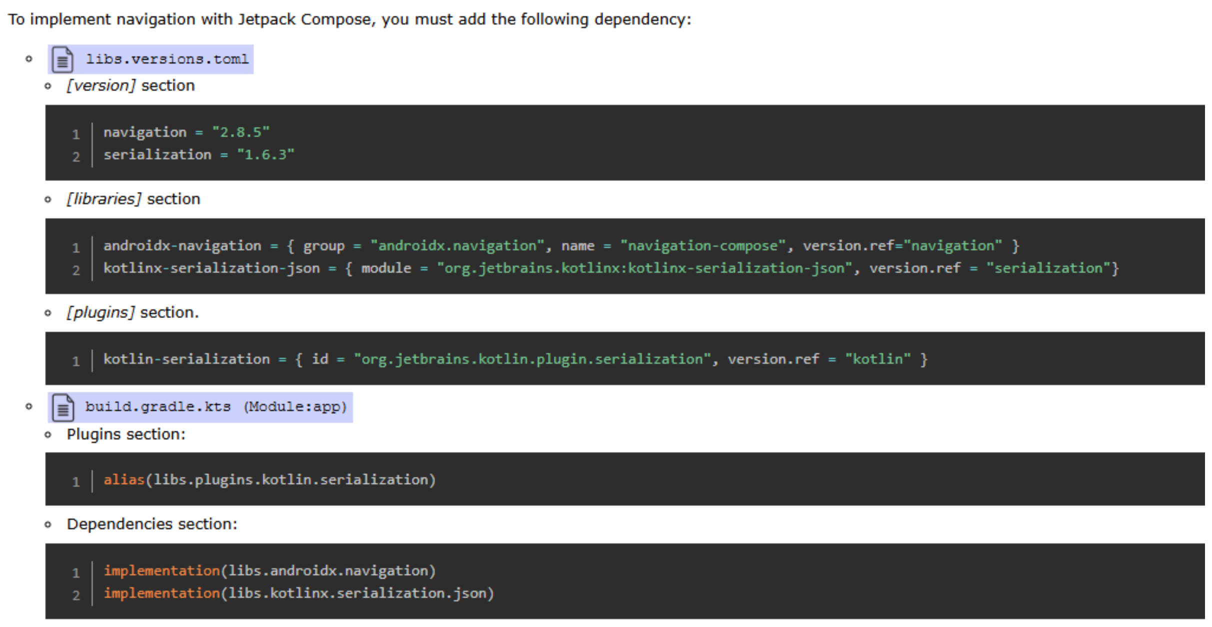
- `build.gradle.kts (Module: app)`
  - *plugins* section:

```
alias(libs.plugins.kotlin.serialization)
```

  - *dependencies* section:

```
implementation(libs.androidx.navigation)
implementation(libs.kotlinx.serialization.json)
implementation(libs.androidx.material.icons)
implementation(libs.androidx.runtime.livedata)
```

Para poder usar navigation
Está en la unidad 8 importante importar todo eso



To implement navigation with Jetpack Compose, you must add the following dependency:

- `libs.versions.toml`
  - *[version]* section

```
navigation = "2.8.5"
serialization = "1.6.3"
```

  - *[libraries]* section

```
androidx-navigation = { group = "androidx.navigation", name = "navigation-compose", version.ref="navigation" }
kotlinx-serialization-json = { module = "org.jetbrains.kotlinx:kotlinx-serialization-json", version.ref = "serialization"}
```

  - *[plugins]* section.

```
kotlin-serialization = { id = "org.jetbrains.kotlin.plugin.serialization", version.ref = "kotlin" }
```

- `build.gradle.kts (Module:app)`
  - Plugins section:

```
alias(libs.plugins.kotlin.serialization)
```

  - Dependencies section:

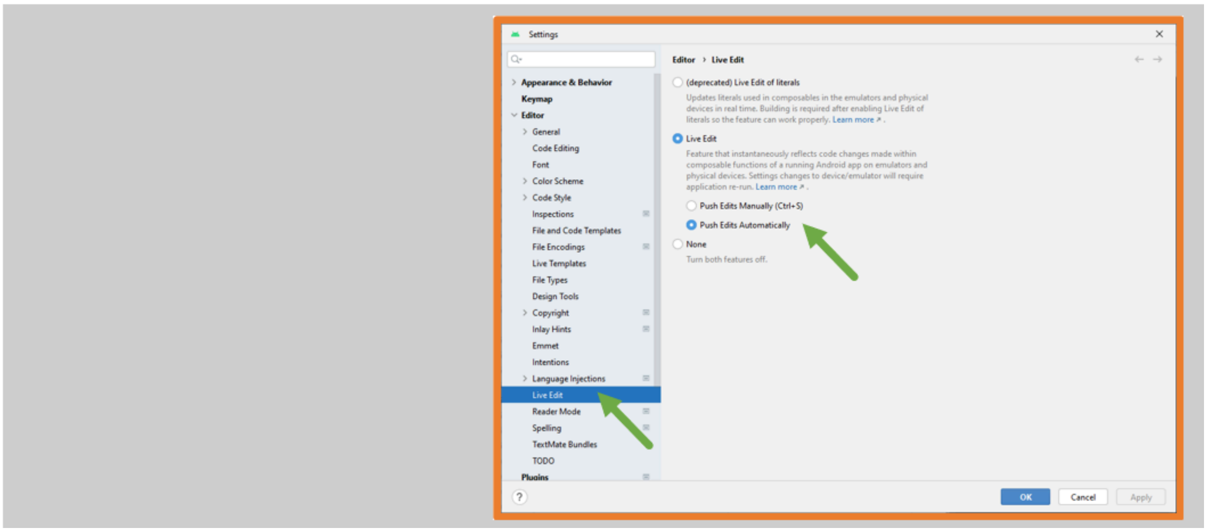
```
implementation(libs.androidx.navigation)
implementation(libs.kotlinx.serialization.json)
```

Actualizar cambios automaticamente

9.3.- Live Edit

To automatically update changes in the emulator, configure Android Studio's **Live Edit** option.

File -> Settings (CONTROL+ALT+S)



Settings

Editor › Live Edit

( ) (deprecated) Live Edit of literals

Updates literals used in composables in the emulators and physical devices in real time. Building is required after enabling Live Edit of literals so the feature can work properly. Learn more ↗ .

(●) Live Edit

Feature that instantaneously reflects code changes made within composable functions of a running Android app on emulators and physical devices. Settings changes to device/emulator will require application re-run. Learn more ↗ .

( ) Push Edits Manually (Ctrl+S)

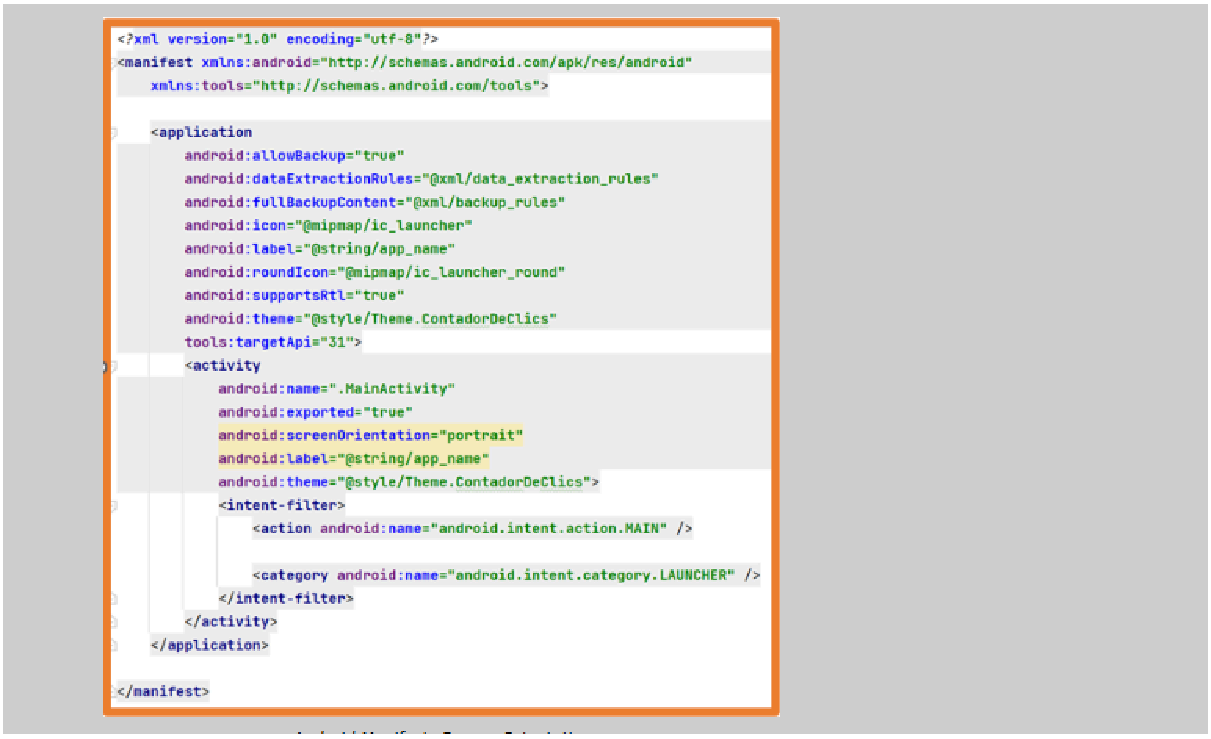
(●) Push Edits Automatically

( ) None

Turn both features off.

Cambiar la aplicación de orientación

llow orientation changes. To achieve this, add the following property to the **Activity** in the `AndroidManifest.xml`



```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools">

    <application
        android:allowBackup="true"
        android:dataExtractionRules="@xml/data_extraction_rules"
        android:fullBackupContent="@xml/backup_rules"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.ContadorDeClics"
        tools:targetApi="31">
        <activity
            android:name=".MainActivity"
            android:exported="true"
            android:screenOrientation="portrait"
            android:label="@string/app_name"
            android:theme="@style/Theme.ContadorDeClics">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

Para añadir acción a cualquier elemento

```kotlin
modifier = Modifier
    .clickable {
        println("¡Se hizo clic!")
    }
```

Añadir dependencia para los iconos

With the **Icons** class, you can use system icons.

The usual scenario is using *Material Design* vector icons.

Android Studio only includes some icons by default. If you need the entire icon set, add the following dependency in:

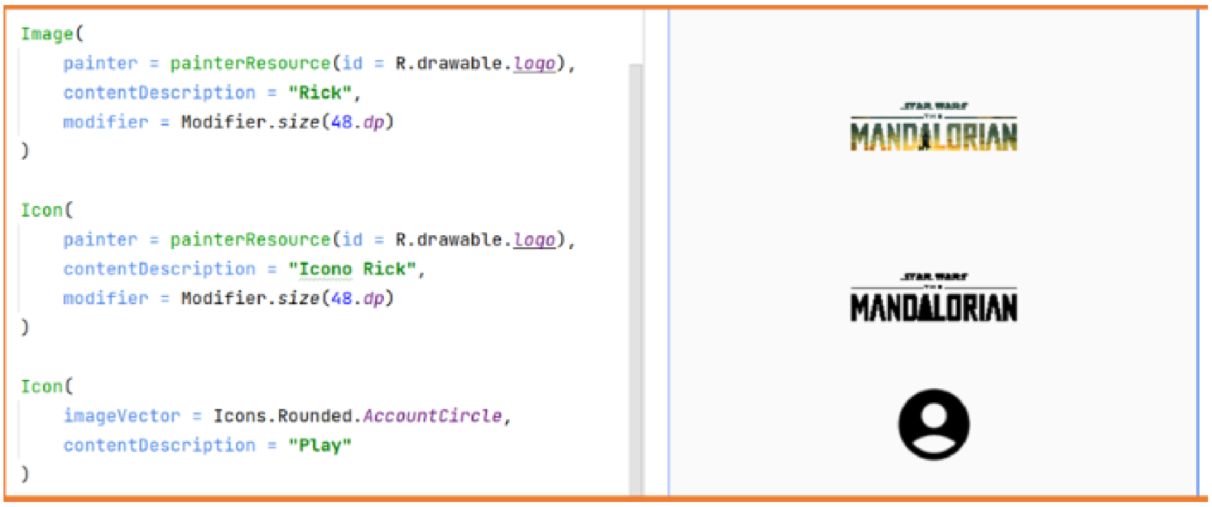
- `libs.versions.toml` *[libraries]* section,

```toml
androidx-material-icons-extended = { group = "androidx.compose.material", name = "material-icons-extended" }
```

- `build.graddle.kts (Module: app)` and synchronize:

```kotlin
implementation(libs.androidx.material.icons.extended)
```

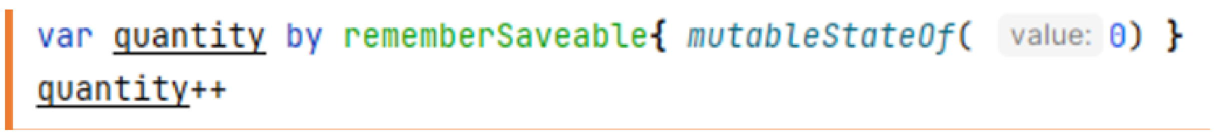
Iconos e imágenes



```kotlin
Image(
    painter = painterResource(id = R.drawable.logo),
    contentDescription = "Rick",
    modifier = Modifier.size(48.dp)
)

Icon(
    painter = painterResource(id = R.drawable.logo),
    contentDescription = "Icono Rick",
    modifier = Modifier.size(48.dp)
)

Icon(
    imageVector = Icons.Rounded.AccountCircle,
    contentDescription = "Play"
)
```

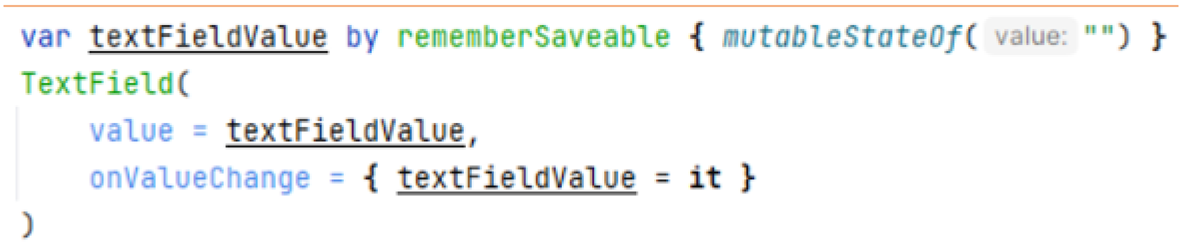
```kotlin
Icon(
    imageVector = Icons.Default.PlayArrow,
    contentDescription = "Play",
    tint = Color.Red
)
```

Variables normales



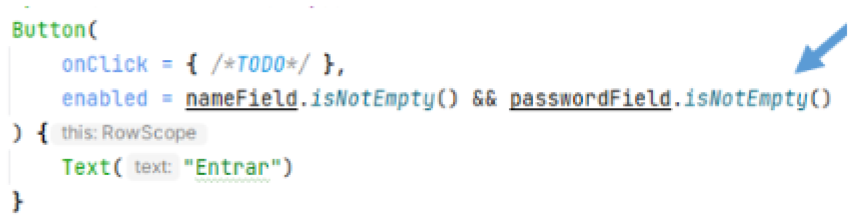
```
var quantity by rememberSaveable{ mutableStateOf( value: 0) }
quantity++
```

Introducción de datos



```
var textFieldValue by rememberSaveable { mutableStateOf( value: "") }
TextField(
    value = textFieldValue,
    onValueChange = { textFieldValue = it }
)
```

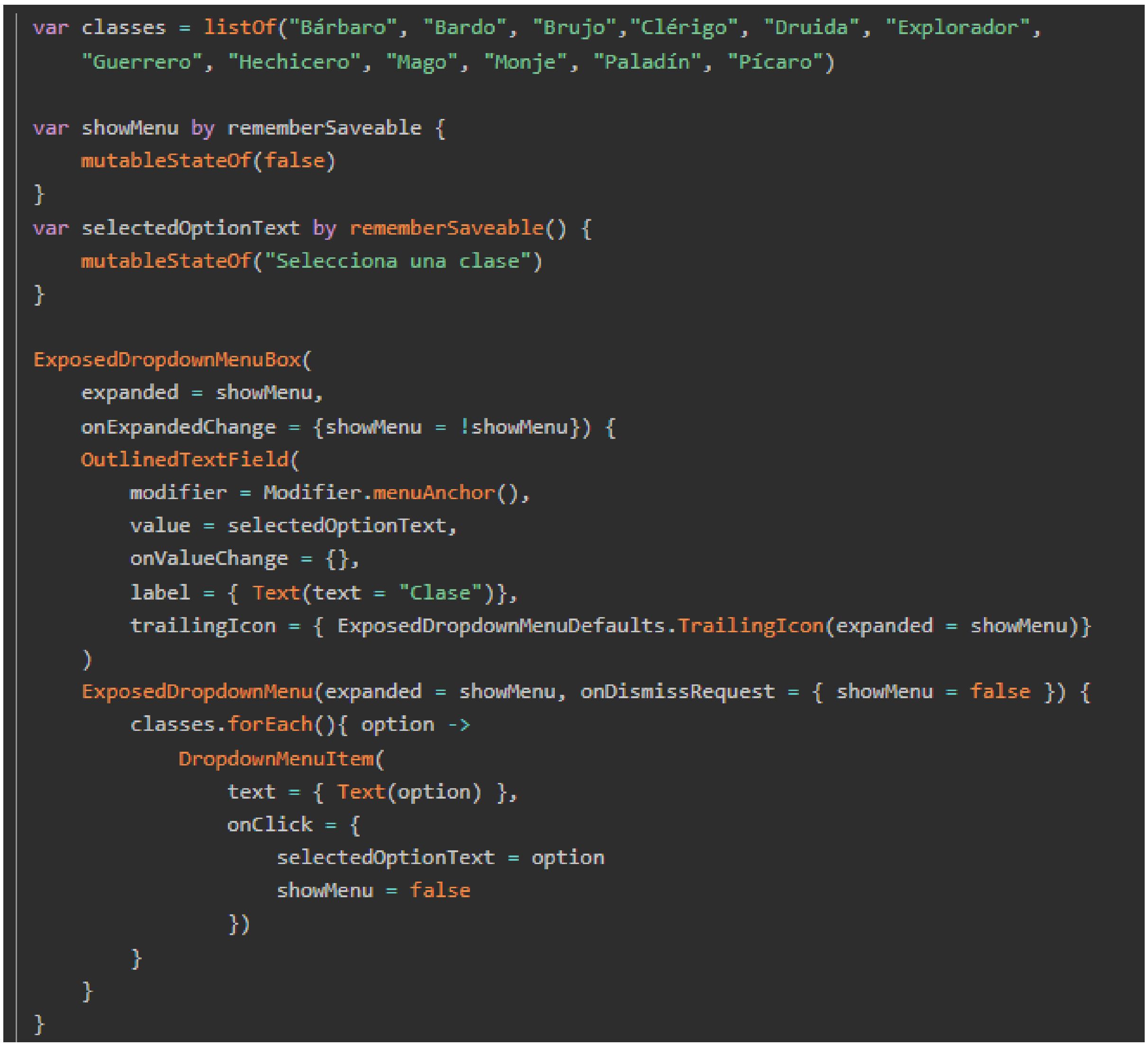
Activar o desactivar un botón:



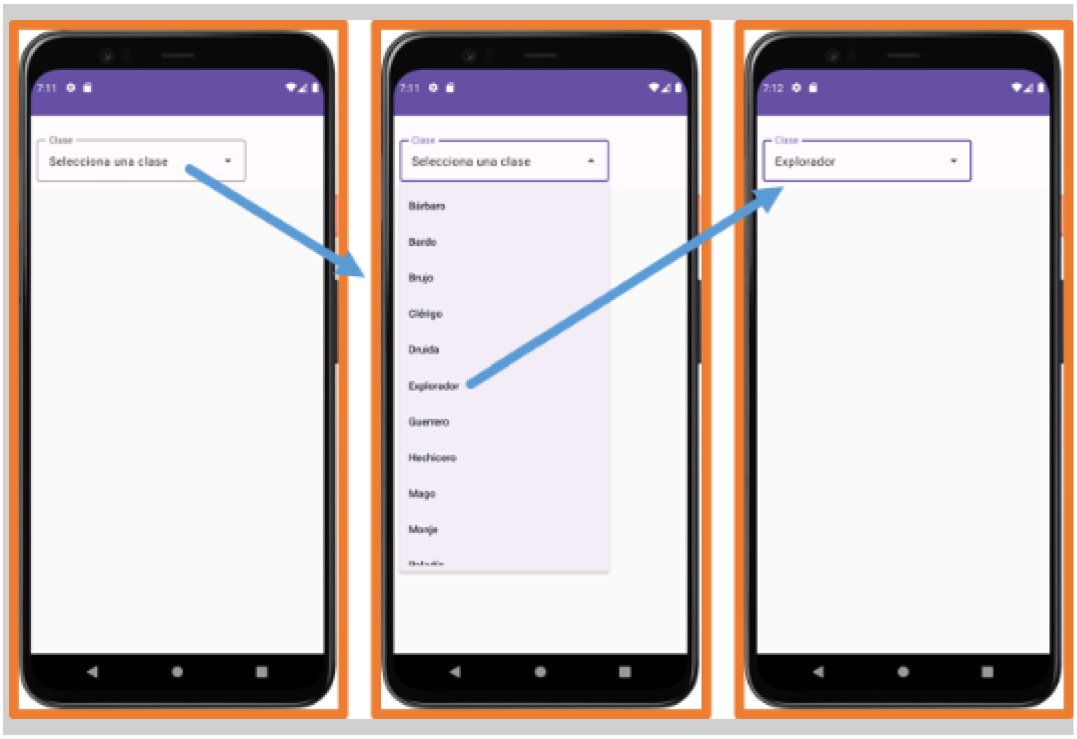
```
Button(
    onClick = { /*TODO*/ },
    enabled = nameField.isNotEmpty() && passwordField.isNotEmpty()
) { this: RowScope
    Text( text: "Entrar")
}
```

Drop down menu

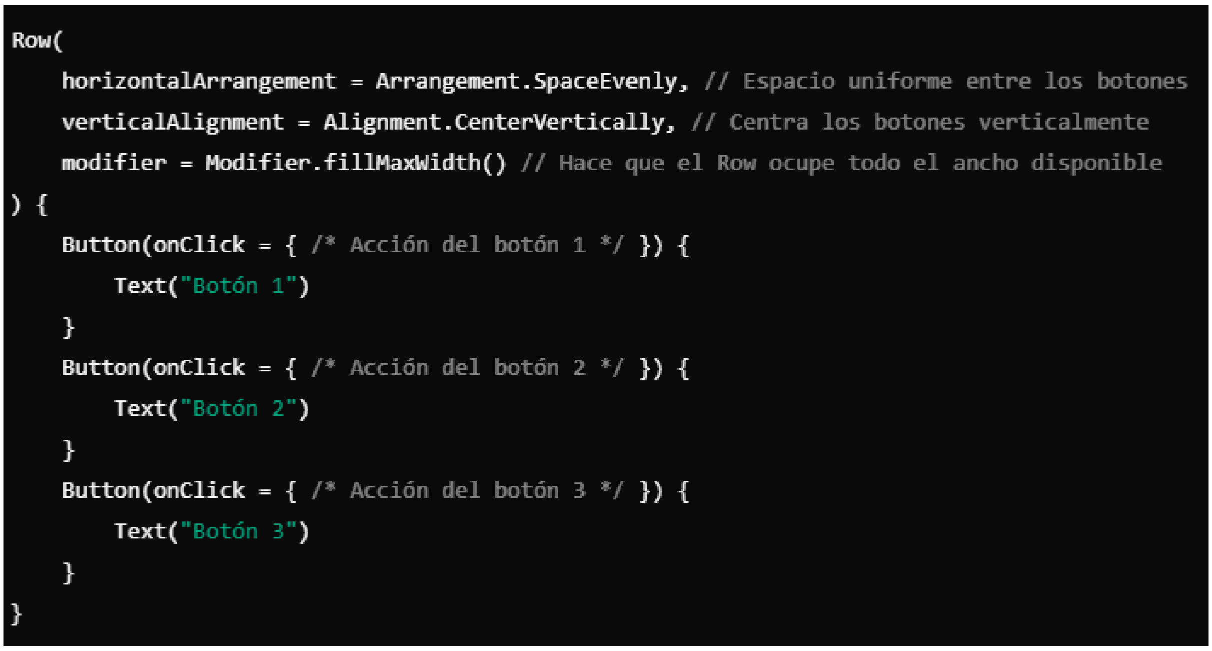
```
var classes = listOf("Bárbaro", "Bardo", "Brujo","Clérigo",
"Druida", "Explorador", "Guerrero", "Hechicero", "Mago", "Monje",
"Paladín", "Pícaro") var showMenu by rememberSaveable {
mutableStateOf(false) } var selectedOptionText by
rememberSaveable() { mutableStateOf("Selecciona una clase") }
ExposedDropdownMenuBox( expanded = showMenu, onExpandedChange =
{showMenu = !showMenu}) { OutlinedTextField( modifier =
Modifier.menuAnchor(), value = selectedOptionText, onValueChange =
{}, label = { Text(text = "Clase")}, trailingIcon = {
ExposedDropdownMenuDefaults.TrailingIcon(expanded = showMenu)} )
ExposedDropdownMenu(expanded = showMenu, onDismissRequest = {
showMenu = false }) { classes.forEach(){ option ->
DropdownMenuItem( text = { Text(option) }, onClick = {
selectedOptionText = option showMenu = false }) } } }
```



```kotlin
var classes = listOf("Bárbaro", "Bardo", "Brujo","Clérigo", "Druida", "Explorador",
    "Guerrero", "Hechicero", "Mago", "Monje", "Paladín", "Pícaro")

var showMenu by rememberSaveable {
    mutableStateOf(false)
}
var selectedOptionText by rememberSaveable() {
    mutableStateOf("Selecciona una clase")
}

ExposedDropdownMenuBox(
    expanded = showMenu,
    onExpandedChange = {showMenu = !showMenu}) {
    OutlinedTextField(
        modifier = Modifier.menuAnchor(),
        value = selectedOptionText,
        onValueChange = {},
        label = { Text(text = "Clase")},
        trailingIcon = { ExposedDropdownMenuDefaults.TrailingIcon(expanded = showMenu)}
    )
    ExposedDropdownMenu(expanded = showMenu, onDismissRequest = { showMenu = false }) {
        classes.forEach(){ option ->
            DropdownMenuItem(
                text = { Text(option) },
                onClick = {
                    selectedOptionText = option
                    showMenu = false
                })
        }
    }
}
```

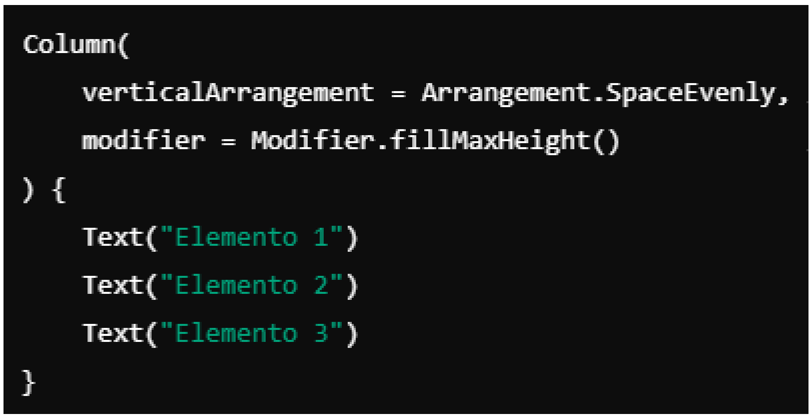


Centrar cosas en un row



```
Row(
    horizontalArrangement = Arrangement.SpaceEvenly, // Espacio uniforme entre los botones
    verticalAlignment = Alignment.CenterVertically, // Centra los botones verticalmente
    modifier = Modifier.fillMaxWidth() // Hace que el Row ocupe todo el ancho disponible
) {
    Button(onClick = { /* Acción del botón 1 */ }) {
        Text("Botón 1")
    }
    Button(onClick = { /* Acción del botón 2 */ }) {
        Text("Botón 2")
    }
    Button(onClick = { /* Acción del botón 3 */ }) {
        Text("Botón 3")
    }
}
```

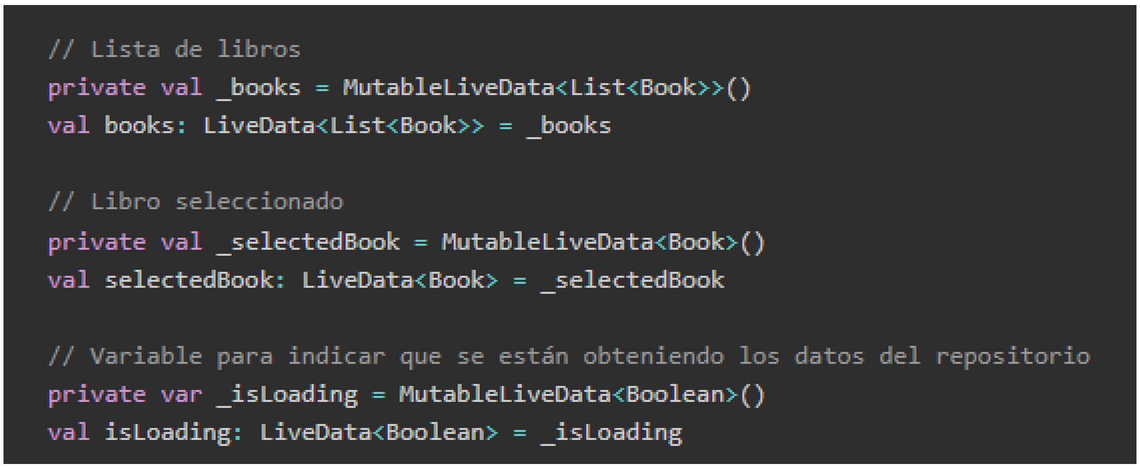
Centrar cosas en un column



```
Column(
    verticalArrangement = Arrangement.SpaceEvenly,
    modifier = Modifier.fillMaxHeight()
) {
    Text("Elemento 1")
    Text("Elemento 2")
    Text("Elemento 3")
}
```

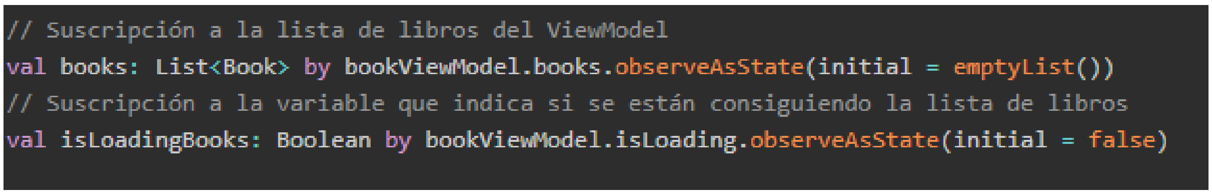
Snack bar es un mensaje temporal, acuerdate si lo piden está en la unidad 7

Live Data
En el view model:



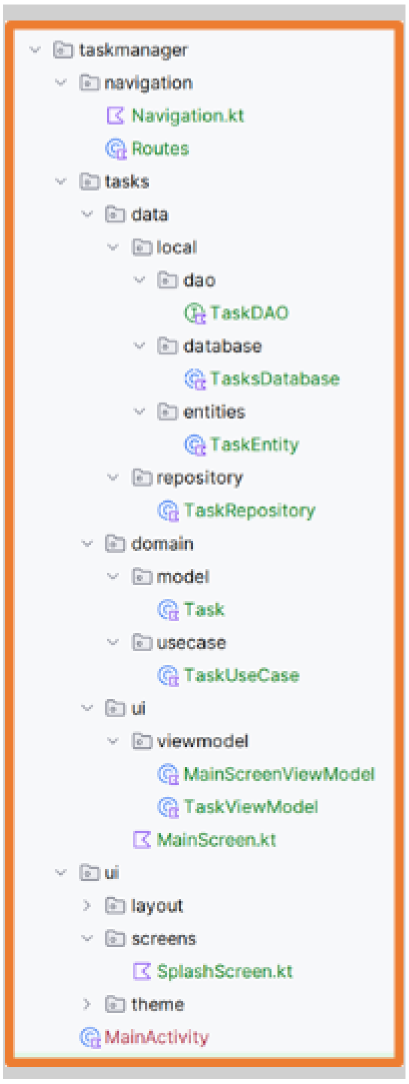
```kotlin
// Lista de libros
private val _books = MutableLiveData<List<Book>>()
val books: LiveData<List<Book>> = _books

// Libro seleccionado
private val _selectedBook = MutableLiveData<Book>()
val selectedBook: LiveData<Book> = _selectedBook

// Variable para indicar que se están obteniendo los datos del repositorio
private var _isLoading = MutableLiveData<Boolean>()
val isLoading: LiveData<Boolean> = _isLoading
```

La suscripción en el composable



```kotlin
// Suscripción a la lista de libros del ViewModel
val books: List<Book> by bookViewModel.books.observeAsState(initial = emptyList())
// Suscripción a la variable que indica si se están consiguiendo la lista de libros
val isLoadingBooks: Boolean by bookViewModel.isLoading.observeAsState(initial = false)
```

Esto son puntitos



```
taskmanager
  navigation
    Navigation.kt
    Routes
  tasks
    data
      local
        dao
          TaskDAO
        database
          TasksDatabase
        entities
          TaskEntity
      repository
        TaskRepository
    domain
      model
        Task
      usecase
        TaskUseCase
    ui
      viewmodel
        MainScreenViewModel
        TaskViewModel
      MainScreen.kt
  ui
    layout
    screens
      SplashScreen.kt
    theme
  MainActivity
```

vista vistamodelo usecase dao vistamodelo vista