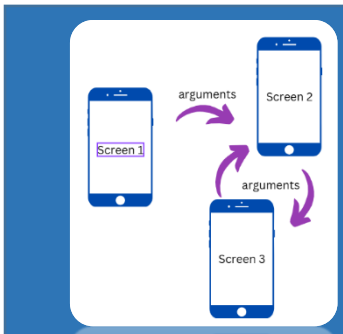
Universidad Nacional Altiplano
FACULTAD DE INGENIERÍA MECÁNICA ELÉCTRICA, ELECTRÓNICA Y SISTEMAS
ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMAS



GUÍA DE LABORATORIO

"Navegación en Jetpack Compose"

Curso	Desarrollo de Plataformas II
Guía de Laboratorio	Navegación en Compose
Semestre	2024 II
Docentes	Ing. Donia Alizandra Ruelas Acero Ing. Miguel Romilio Aceituno Rojo

GUÍA DE LABORATORIO

"Navegación En Jetpack Compose"

Introducción:

Una de las características clave de una aplicación móvil es la **navegación**, y Jetpack Compose introduce un enfoque novedoso para gestionar la navegación entre pantallas, componentes y fragmentos de forma declarativa.

Esta Guía de Laboratorio explora los fundamentos de la navegación en Jetpack Compose, sus características, patrones y cómo implementarla efectivamente.

Objetivos:

Implementar una estructura de navegación fluida y eficiente en aplicaciones Android utilizando el sistema de navegación específico de Jetpack Compose.

Objetivos Específicos:

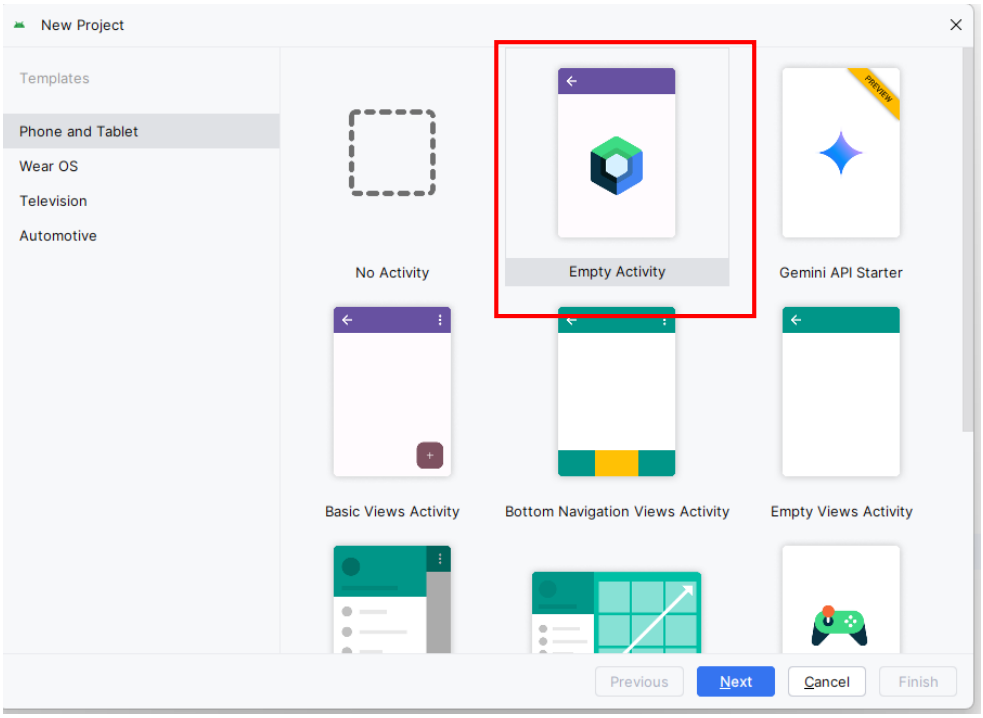
- Preparar la Navegación
- Navegar sin parámetros
- Navegar con parámetros

Requisitos:

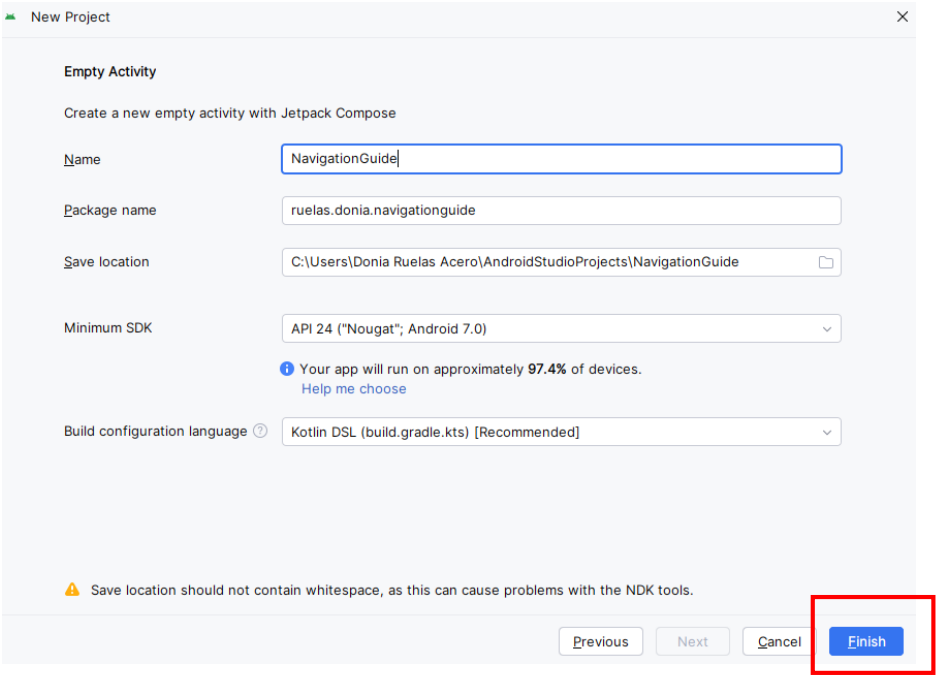
- Android Studio Instalado y actualizado: Esta guía se trabajó con la versión 17.0 Se recomienda que este actualizado para utilizar las últimas dependencias que provee el sistema.
- Computador con las siguientes características:
  - *Procesador:* Al menos un procesador Intel Core i5 o equivalente, aunque se recomienda i7 o superior.
  - *Memoria RAM*: Al menos 8 GB de RAM, se e recomienda 16 GB a más para los emuladores y un rendimiento adecuado.
  - *Tarjeta gráfica:* GPU integrada o dedicada, con soporte para OpenGL ES 2.0 o superior (para mejor rendimiento de emuladores).
  - *Espacio en disco:* Al menos 10 GB de espacio libre (20 GB recomendado).
- *Conexión a Internet estable*, para las actualizaciones de Android Studio, bibliotecas y dependencias de Gradle, y otros recursos necesarios

ACTIVIDAD 1: CREACIÓN DE UN NUEVO PROYECTO

Creamos un nuevo proyecto en Compose haciendo clic en **Empty Activity.**



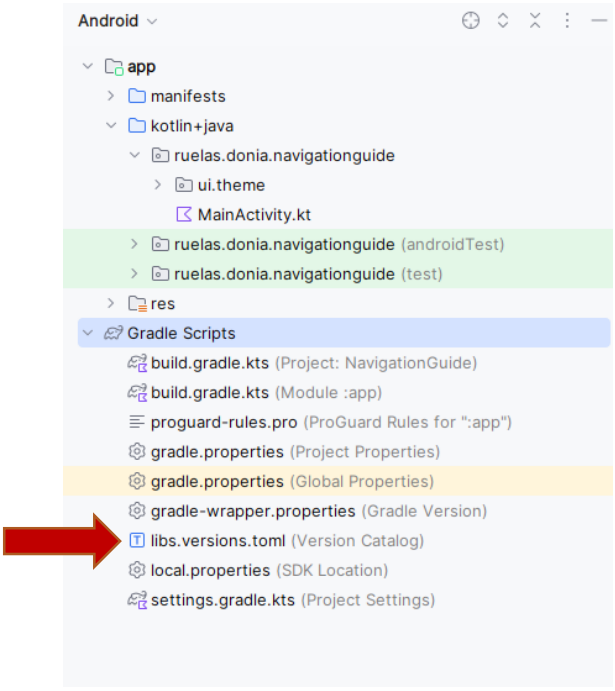
A continuación, ingresamos el nombre del proyecto, seleccionamos el nombre adecuado para el paquete, elegimos la ubicación donde se guardará el proyecto, la versión de la API y el lenguaje de programación.



ACTIVIDAD 2: CONFIGURACIÓN DE DEPENDENCIAS DEL PROYECTO

Vamos a agregar las dependencias para nuestra aplicación para lo cual modificaremos los archivos **libs.versions.toml** y **build.gradle.kts**

- **En el archivo libs.versions.toml**



Agregaremos las versiones de las librerías para la navegación:

Seguidamente agregamos los módulos donde están las librerías:

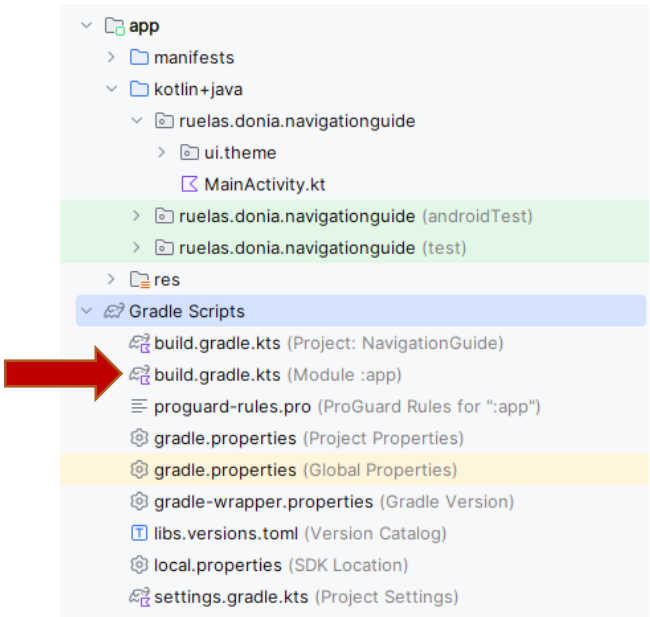
```
[libraries]
androidx-core-ktx = { group = "androidx.core", name = "core-ktx", version.ref = "coreKtx" }
junit = { group = "junit", name = "junit", version.ref = "junit" }
androidx-junit = { group = "androidx.test.ext", name = "junit", version.ref = "junitVersion" }
androidx-espresso-core = { group = "androidx.test.espresso", name = "espresso-core", version.ref = "espressoCore" }
androidx-lifecycle-runtime-ktx = { group = "androidx.lifecycle", name = "lifecycle-runtime-ktx", version.ref = "lifecycleRuntimeKt
androidx-activity-compose = { group = "androidx.activity", name = "activity-compose", version.ref = "activityCompose" }
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" }
androidx-navigation-compose = {module = "androidx.navigation:navigation-compose", version.ref ="navigationCompose"}
kotlinx-serialization-json = {module = "org.jetbrains.kotlinx:kotlinx-serialization-json", version.ref= "kotlinxSerialization"}
```

Así mismo agregamos en enlace del recurso del plugin:

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

A continuación, sincronizamos Gradle para que descargue las dependencias.

- **En el archivo build.gradle.kts**



Finalmente, en el archivo ***build.gradle.kts*** agregamos las dependencias.

```
52  dependencies {
53
54      implementation(libs.androidx.core.ktx)
55      implementation(libs.androidx.lifecycle.runtime.ktx)
56      implementation(libs.androidx.activity.compose)
57      implementation(platform(libs.androidx.compose.bom))
58      implementation(libs.androidx.ui)
59      implementation(libs.androidx.ui.graphics)
60      implementation(libs.androidx.ui.tooling.preview)
61      implementation(libs.androidx.material3)
62
63      implementation(libs.androidx.navigation.compose)
64      implementation(libs.kotlinx.serialization.json)
65
66      testImplementation(libs.junit)
67      androidTestImplementation(libs.androidx.junit)
```

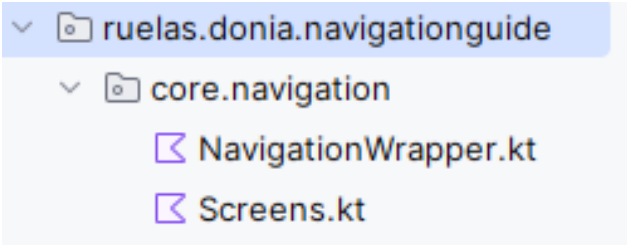
Y el plugin:

```
1   plugins {
2       alias(libs.plugins.android.application)
3       alias(libs.plugins.kotlin.android)
4       alias(libs.plugins.kotlin.serialization)
5   }
```

ACTIVIDAD 3: PREPARAR LA NAVEGACIÓN

Para la navegación entre las pantallas (*screens*) utilizaremos una función de navegación principal.

Para lo cual crearemos dentro del *Package* del proyecto una carpeta con el nombre core.navigation, y dentro de ella crearemos el archivo kotlin: NavigationWrapper.kt y el archivo Screens.kt.



En el archivo Screens.kt, creamos los objetos que se utilizarán en la función de navegación(NavigationWrapper()).

Screens.kt

```kotlin
package ruelas.donia.navigationguide.core.navigation
import kotlinx.serialization.Serializable

@Serializable
object Login

@Serializable
object Home
```

@Serializable: La etiqueta @Serializable es parte de la biblioteca kotlinx.serialization, la cual permite convertir objetos de Kotlin en formatos de datos como JSON, XML, o CBOR, y viceversa. Al marcar Login como @Serializable, se indica que el objeto puede ser serializado y deserializado automáticamente. Esto es útil para transmitir datos entre diferentes partes de una aplicación o para guardar datos en formatos que puedan almacenarse o compartirse fácilmente.

object: La palabra clave object en Kotlin define un objeto singleton, es decir, una instancia única que se crea de forma estática y existe en un solo lugar en el código. Un object en Kotlin es útil cuando deseas que exista una única instancia de una clase en toda la aplicación, lo cual ahorra memoria y simplifica el acceso.

Login: Login es el nombre del objeto creado. Es utilizarlo para almacenar o gestionar información o comportamientos relacionados con el "inicio de sesión" de la aplicación.

En el archivo NavigationWrapper.kt, creamos la función de navegación, que se encargará de toda la navegación.

NavigationWrapper.kt

```
@Composable
fun  NavigationWrapper() {
    val navController = rememberNavController()
    NavHost(navController = navController, startDestination = Login) {
        composable<Login> {
            LoginScreen { navController.navigate(Home) }
        }
        composable<Home> {
            HomeScreen()
        }
    }
}
```

NavControler: Es un variable para crear y recordar una instancia del controlador de navegación (NavController) que gestiona el historial de la navegación y realizar la navegación entre pantallas en una aplicación de Android.

NavHost: Es un contenedor que define el gráfico de navegación de la aplicación, es decir, especifica las pantallas (o destinos) entre las cuales se puede navegar. Define una estructura de navegación básica, se utiliza para establecer la navegación entre las pantallas de la interfaz de usuario.

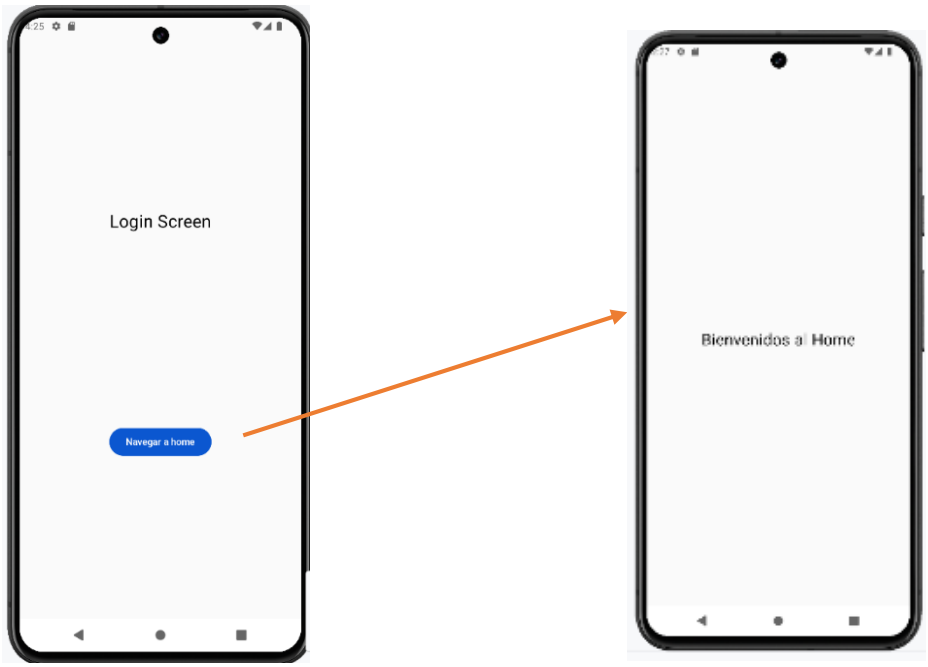
startDestination: Es la propiedad que define la primera pantalla que se muestra cuando la aplicación carga el NavHost. En este caso, Login es la pantalla de inicio.

```
NavHost(navController = navController, startDestination = Login)
```

composable<Login> { ... }: Este bloque define una pantalla o destino dentro del NavHost. La función composable crea una pantalla de tipo Login en el flujo de navegación de la aplicación. La función composable es utilizada para especificar cada pantalla a la que se puede navegar.

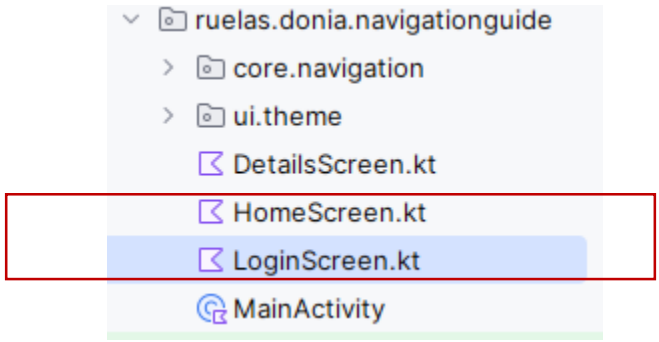
- **<Login>**: Aquí se usa Login como un identificador del destino para la pantalla de inicio de sesión.
- **LoginScreen()**: Dentro del bloque { ... }, llamamos a LoginScreen(), que representa la interfaz o contenido de la pantalla de Login. Es un *Composable* (es decir, una función que describe la UI en Jetpack Compose).

ACTIVIDAD 4: NAVEGACIÓN SIN PARÁMETROS



Una vez preparado la navegación, crearemos las interfaces de Login y Home para poder navegar entre ellas.

Primero creamos la pantalla Login y la pantalla Home, para lo cual creamos un archivo *kotlin*, llamado ***LoginScreen.kt y HomeScreen*** dentro del *package* del proyecto.



Para la implementación de la interfaz Login y Home utilizando los componentes *Spacer*, *Text* y *Button*.

LoginScreen.kt

```
@Composable
fun LoginScreen(navigateToHome: ()->Unit){
    Column(modifier = Modifier.fillMaxSize(),
        horizontalAlignment = Alignment.CenterHorizontally
    ){
        Spacer(modifier = Modifier.weight(1f))
        Text( text: "Login Screen", fontSize = 25.sp )
        Spacer(modifier = Modifier.weight(1f))
        Button(onClick = { navigateToHome()}) {
            Text( text: "Navegar a home")
        }
        Spacer(modifier = Modifier.weight(1f))
    }
}
```

La función LoginScreen () recibe como parámetro otra función que se ejecutará al hacer clic en el botón "Navegar a Home".

HomeScreen.kt

```
@Composable
fun HomeScreen(){
    Column(
        modifier = Modifier.fillMaxSize(),
        horizontalAlignment = Alignment.CenterHorizontally
    ){
        Spacer(modifier = Modifier.weight(1f))
        Text( text: "Bienvenidos al Home", fontSize = 25.sp)
        Spacer(modifier = Modifier.weight(1f))
    }
}
```

Para poder visualizar, en el MainActivity llamamos a la función de navegación creada, NavigationWrapper().

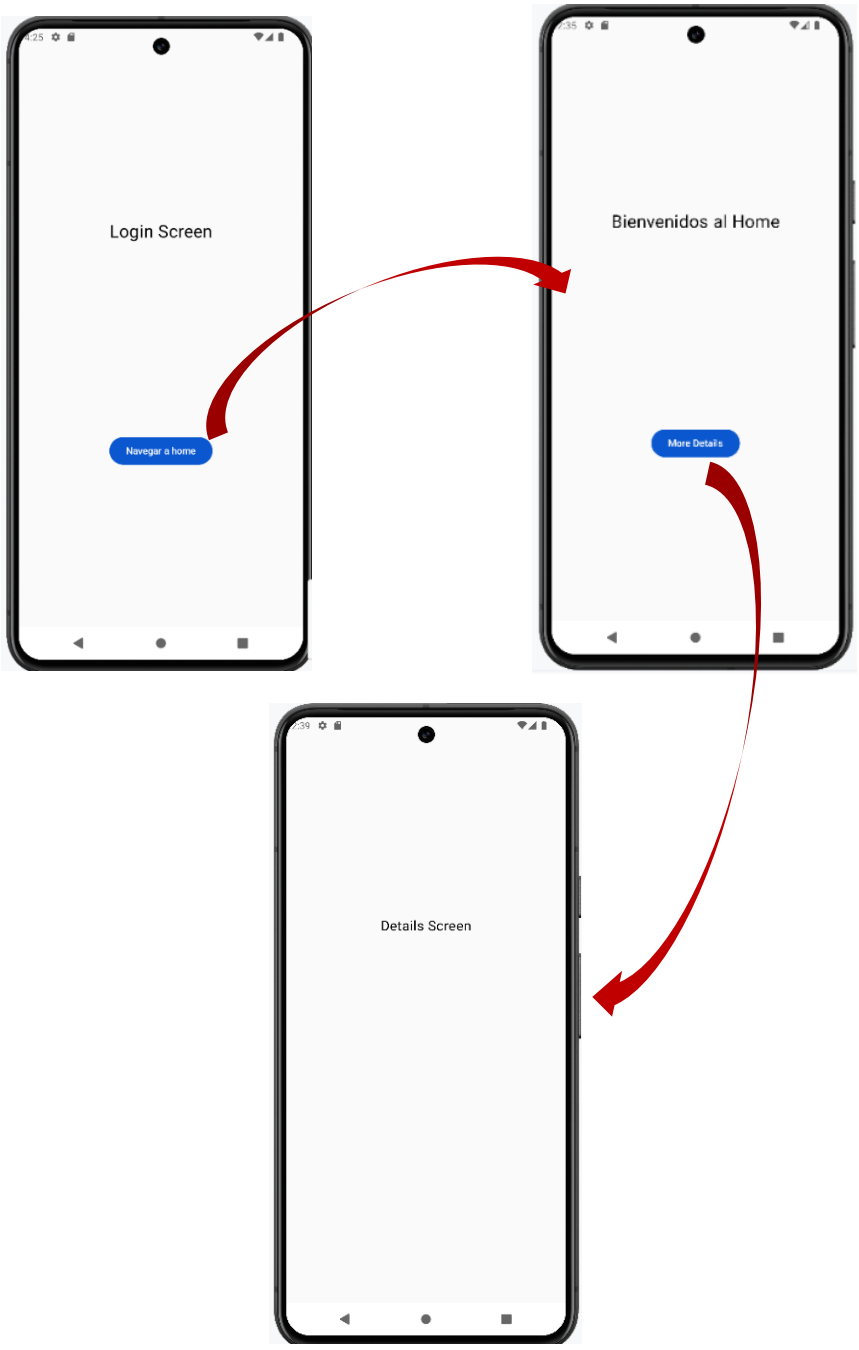
MainActivity

```
class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        enableEdgeToEdge()
        setContent {
            NavigationGuideTheme {
                NavigationWrapper()
            }
        }
    }
}
```

Finalmente ejecutamos la aplicación.

RETO 1:

Elbora una interfaz **DetailsScreen.kt** y agregue la navegación de la pantalla **Home** a dicha interfaz.



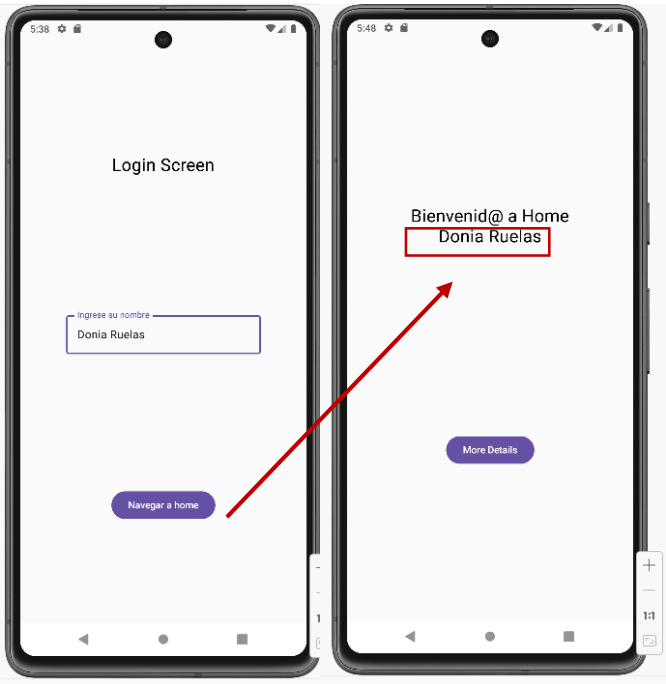
En Jetpack Compose, la navegación entre pantallas se realiza mediante el componente NavHost, que permite estructurar rutas y destinos dentro de una aplicación.	

ACTIVIDAD 5: NAVEGACIÓN CON PARÁMETROS

Además de navegar de una pantalla a otra, Jetpack Compose también admite la navegación con parámetros, una funcionalidad esencial para pasar datos entre *composables* (pantallas o componentes).

La navegación con parámetros permite enviar información específica desde una pantalla de origen a una pantalla de destino. Esto es útil, por ejemplo, al querer mostrar detalles de un elemento seleccionado en una lista o enviar un identificador para cargar información específica en la siguiente pantalla.

En la siguiente imagen se muestra la navegación desde la pantalla de ***Login*** hacia la pantalla de ***Home***, donde se transfiere el texto ingresado.



Primeramente, editaremos la función **LoginScreen.**

1. Agregamos el parámetro a la función.
2. Agregamos el componente *OutlinedTextField*, para ingresar un texto y enviarlo como parámetro a la pantalla Home.
3. Pasamos el parámetro *text,* que contiene el texto ingresado, a la función *navigateToHome.*

LoginScreen.kt

```
@Composable
fun LoginScreen(navigateToHome: (String)->Unit){
    var text by remember { mutableStateOf( value: "")}
    Column(modifier = Modifier.fillMaxSize(),
        horizontalAlignment = Alignment.CenterHorizontally
    ){
        Spacer(modifier = Modifier.weight(1f))
        Text( text: "Login Screen", fontSize = 25.sp )
        Spacer(modifier = Modifier.weight(1f))
        OutlinedTextField(value = text,
            label= {Text( text: "Ingrese su nombre")},
            onValueChange = { text=it }
        )
        Spacer(modifier = Modifier.weight(1f))
        Button(onClick = { navigateToHome(text)}) {
            Text( text: "Navegar a home")
        }
        Spacer(modifier = Modifier.weight(1f))
    }
```

Seguidamente, editaremos la función **HomeScreen.**

1. Agregamos el parámetro que recibirá.
2. Mostramos el texto recibido.

HomeScreen.kt

```
@Composable
fun HomeScreen(name: String, navigationToDetails:()->Unit){

    Column(
        modifier = Modifier.fillMaxSize(),
        horizontalAlignment = Alignment.CenterHorizontally
    ){
        Spacer(modifier = Modifier.weight(1f))
        Text( text: "Bienvenid@ a Home", fontSize = 25.sp)
        Text( text: "$name", fontSize = 25.sp)
        Spacer(modifier = Modifier.weight(1f))
        Button(onClick = {navigationToDetails() }){
            Text( text: "More Details")
        }
        Spacer(modifier = Modifier.weight(1f))
    }
}
```

Posteriormente, configuramos el archivo **Screen.kt**, para editar el objeto Home a una ***data class*** que recibe un parámetro de tipo *String*.

Screens.kt

```
4    @Serializable
5    object Login
6
7    @Serializable
8    data class Home(val name:String)
9
10   @Serializable
11   object Details
```

Finalmente, realizamos la configuración de nuestra función de navegación NavigationWrapper(), editaremos el contenedor de rutas de navegación *NavHost* y el controlador *NavController* para manejar las transiciones entre las pantallas *Login* y *Home*.

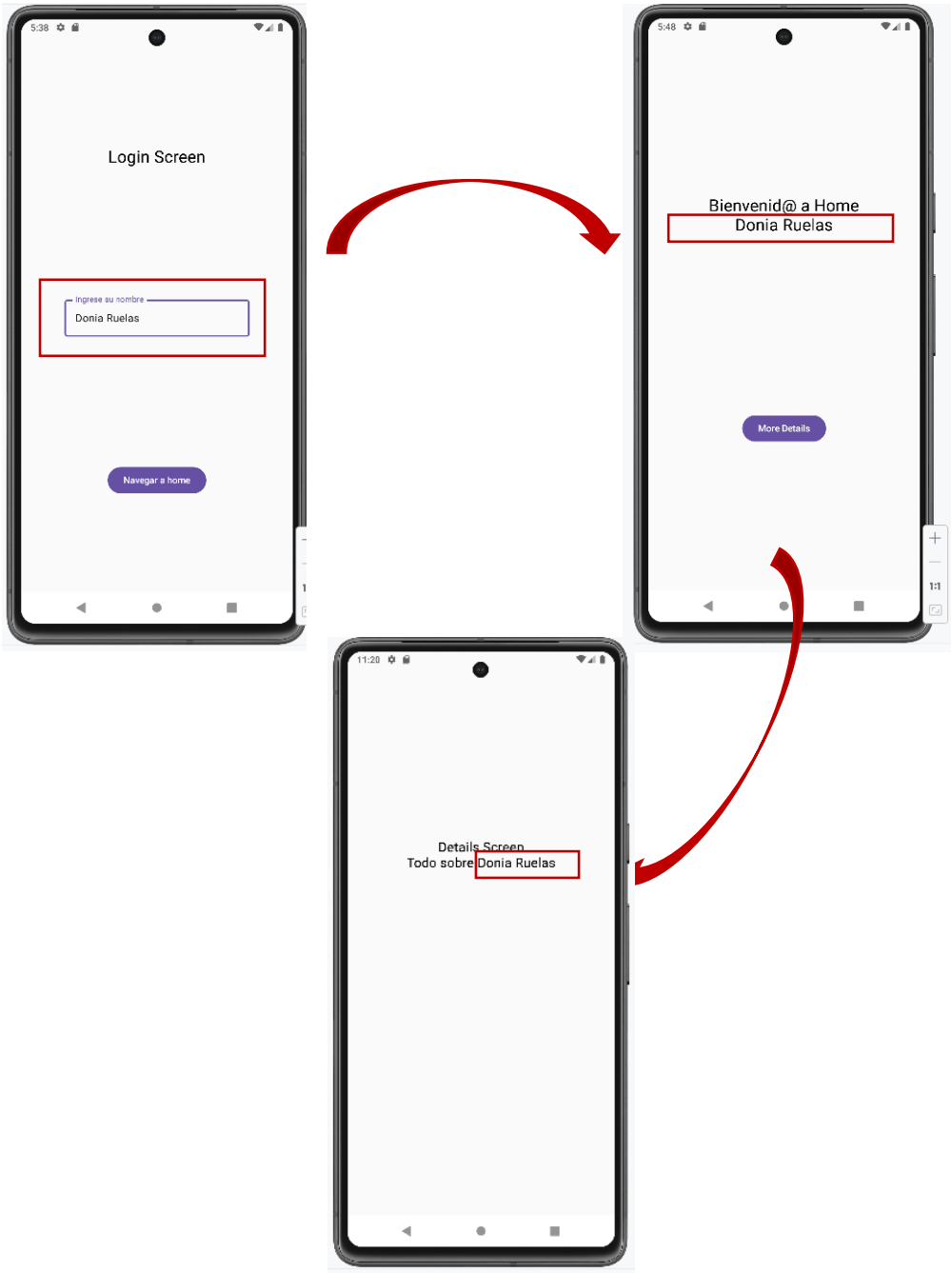
1. Especificaremos la navegación hacia la pantalla principal (*Home*) al capturar un parámetro *name*.
2. Extraemos datos de la ruta de navegación usando un backStackEntry, que representa el elemento de la pila de navegación (back stack), que contiene detalles sobre el estado de esta pantalla cuando se navega hacia ella.

NavigationWrapper.kt

```
12   @Composable
13   fun  NavigationWrapper() {
14       val navController = rememberNavController()
15       NavHost(navController = navController, startDestination = Login) {
16           composable<Login> {
17               LoginScreen { name->navController.navigate(Home(name=name)) }
18           }
19           composable<Home> { backStackEntry ->
20               val detail:Home = backStackEntry.toRoute()
21               HomeScreen(detail.name) { navController.navigate(Details)}
22
23           }
24           composable<Details> {
25               DetailScreen()
26           }
27       }
28   }
```

RETO 2:

Elabora la siguiente navegación de Login -> Home-> Details, enviando el parámetro ingresado en Login (Su primer nombre y primer apellido).



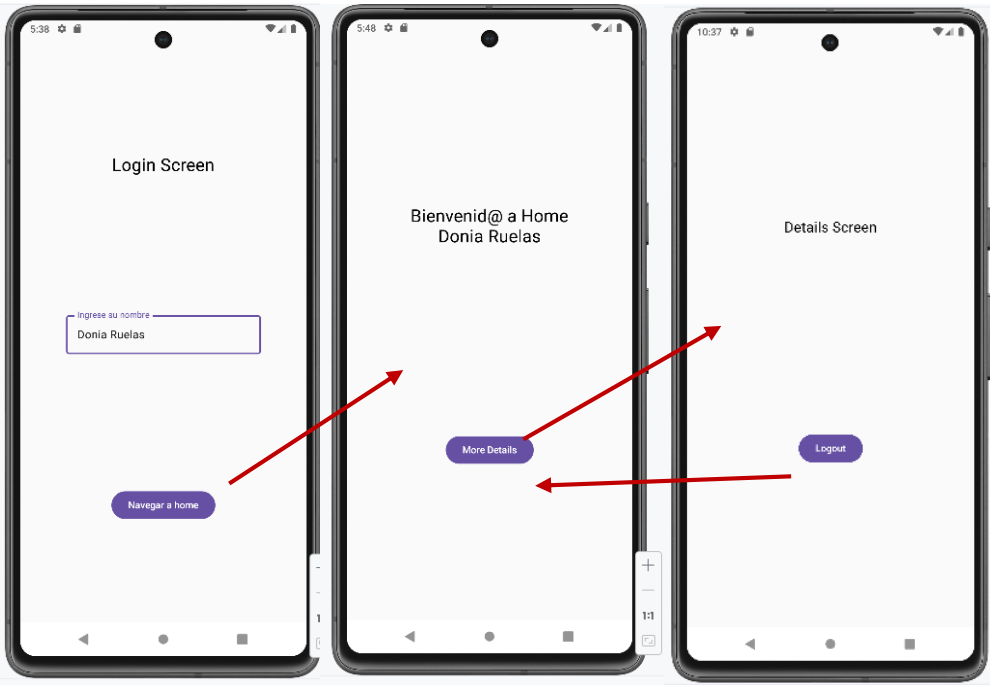
Google y otros expertos en desarrollo de software suelen recomendar no pasar parámetros complejos, como objetos de datos completos, entre componentes de la interfaz de usuario. En lugar de eso, se sugiere pasar únicamente los identificadores necesarios o datos básicos. Este enfoque ayuda a mejorar la eficiencia y la mantenibilidad de las aplicaciones.

ACTIVIDAD 6: NAVEGACIÓN PARA ATRÁS

Además de navegar de una pantalla a otra, *Jetpack Compose* también admite la navegación con parámetros, una funcionalidad esencial para pasar datos entre *composables* (pantallas o componentes).

La navegación con parámetros permite enviar información específica desde una pantalla de origen a una pantalla de destino. Esto es útil, por ejemplo, al querer mostrar detalles de un elemento seleccionado en una lista o enviar un identificador para cargar información específica en la siguiente pantalla.

En la siguiente imagen se muestra la navegación desde la pantalla de ***Login*** hacia la pantalla de ***Home***, donde se transfiere el texto ingresado.



Primeramente, editaremos la función **DetailsScreen:**

1. Agregamos el parámetro a la función.
2. Agregamos el componente Button, que permite regresar enviando como parámetro la función *navigateBack()*.

DetailsScreen.kt

```
@Composable
fun DetailScreen(navigateBack:()->Unit){
    Column(
        modifier = Modifier.fillMaxSize(),
        horizontalAlignment = Alignment.CenterHorizontally
    ) {
        Spacer(modifier = Modifier.weight(1f))
        Text( text: "Details Screen", fontSize = 20.sp)
        Spacer(modifier = Modifier.weight(1f))
        Button(onClick = {navigateBack()}) {
            Text( text: "Atrás")
        }
        Spacer(modifier = Modifier.weight(1f))
    }
}
```

Seguidamente, editaremos la función de navegación **NavigationWrapper.**

1. Pasamos como parámetro una acción para navegar hacia atrás en la pila de navegación, para ello utilizamos la función *navigateUp().*

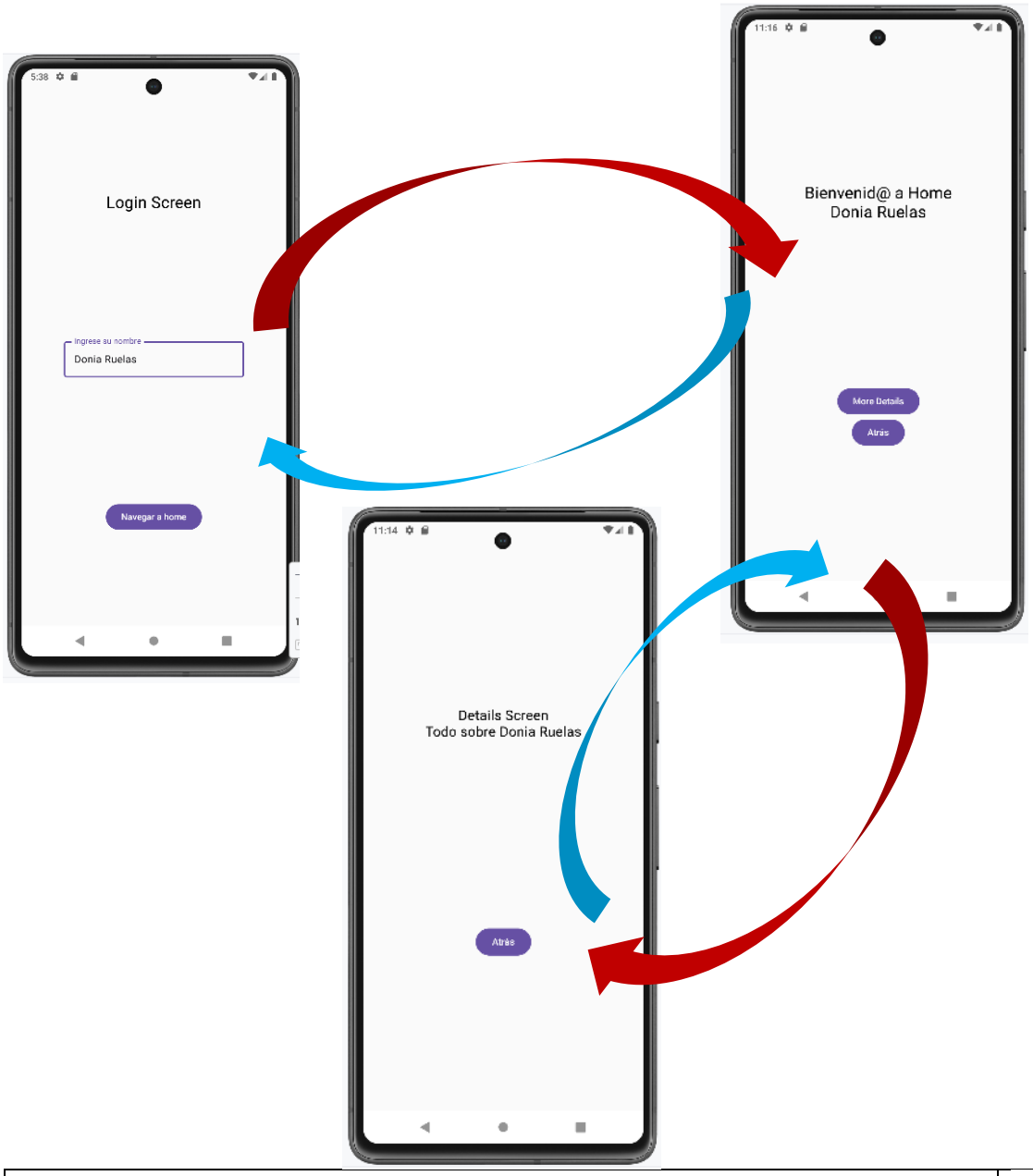
NavigationWrapper.kt

```
@Composable
fun  NavigationWrapper() {
    val navController = rememberNavController()
    NavHost(navController = navController, startDestination = Login) {
        composable<Login> {
            LoginScreen { name->navController.navigate(Home(name = name)) }
        }
        composable<Home> { backStackEntry ->
            val home:Home = backStackEntry.toRoute()
            HomeScreen(home.name) { navController.navigate(Details)}

        }
        composable<Details> {
            DetailScreen{ navController.navigateUp() }
        }
    }
}
```

RETO 3:

Elbora la siguiente navegación: Login->Home->Details, considerando el parámetro a enviar, y vuelta atrás de navegación.



En Jetpack Compose, la navegación entre pantallas se realiza mediante el componente NavHost, que permite estructurar rutas y destinos dentro de una aplicación.