Mobile Application Development

Week7　App Navigation

Yi SUN
Kobe Institute of Computing

What is Navigation

- Navigation refers to the interactions that let users navigate across, into, and back out from the different pieces of content within your app.
- Android Jetpack's Navigation component includes the Navigation library, Safe Args Gradle plug-in, and tooling to help you implement app navigation. The Navigation component handles diverse navigation use cases, from straightforward button clicks to more complex patterns, such as app bars and the navigation drawer.

https://developer.android.com/guide/navigation

Key concepts

Concept	Purpose	Type
Host	A UI element that contains the current navigation destination. That is, when a user navigates through an app, the app essentially swaps destinations in and out of the navigation host.	•Compose: NavHost •Fragments: NavHostFragment
Graph	A data structure that defines all the navigation destinations within the app and how they connect together.	NavGraph
Controller	The central coordinator for managing navigation between destinations. The controller offers methods for navigating between destinations, handling deep links, managing the back stack, and more.	NavController
Destination	A node in the navigation graph. When the user navigates to this node, the host displays its content.	NavDestination Typically created when constructing the navigation graph.
Route	Uniquely identifies a destination and any data required by it. You can navigate using routes. Routes take you to destinations.	Any serializable data type.

Benefits and features

- **Animations and transitions:** Provides standardized resources for animations and transitions.
- **Deep linking:** Implements and handles deep links that take the user directly to a destination.
- **UI patterns:** Supports patterns such as navigation drawers and bottom navigation with minimal additional work.
- **Type safety:** Includes support for passing data between destinations with type safety.
- **ViewModel support:** Enables scoping a ViewModel to a navigation graph to share UI-related data between the graph's destinations.
- **Fragment transactions:** Fully supports and handles fragment transactions.
- **Back and up:** Handles back and up actions correctly by default.

Understand Navigation by simple sample

- We are building a simple greeting app.

1. Open AndroidStudio
2. Select `File > New > NewProject`
3. Choose `Empty Activity`
4. Name: NavigationSample

Prepare the First Screen

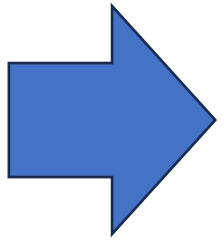
- New Kotlin file FirstScreen.kt

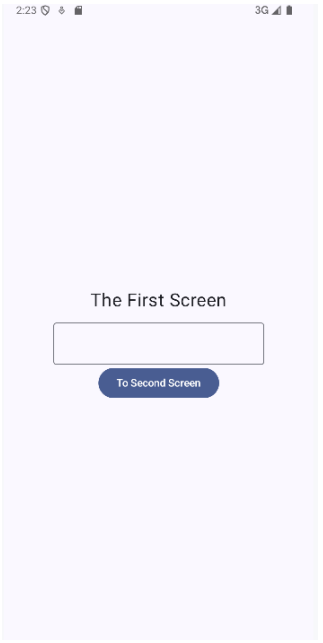
FirstScreen.kt

```
@Composable
fun FirstScreen(modifier: Modifier){
    val name = remember { mutableStateOf("") }
    Column(modifier = Modifier.fillMaxSize().padding(16.dp),
        verticalArrangement = Arrangement.Center,
        horizontalAlignment = Alignment.CenterHorizontally){
        Text("The First Screen", fontSize = 24.sp)
        Spacer(modifier = Modifier.height(16.dp))
        OutlinedTextField(value = name.value, onValueChange = {
            name.value = it
        })
        Button(onClick = {}) {
            Text("To Second Screen")
        }
    }
}
```

MainActivity.kt

```
NavigationSampleTheme {
    Scaffold(modifier = Modifier.fillMaxSize()) { innerPadding ->
        FirstScreen(
            modifier = Modifier.padding(innerPadding)
        )
    }
}
```





Prepare the Second Screen

- New Kotlin file SecondScreen.kt

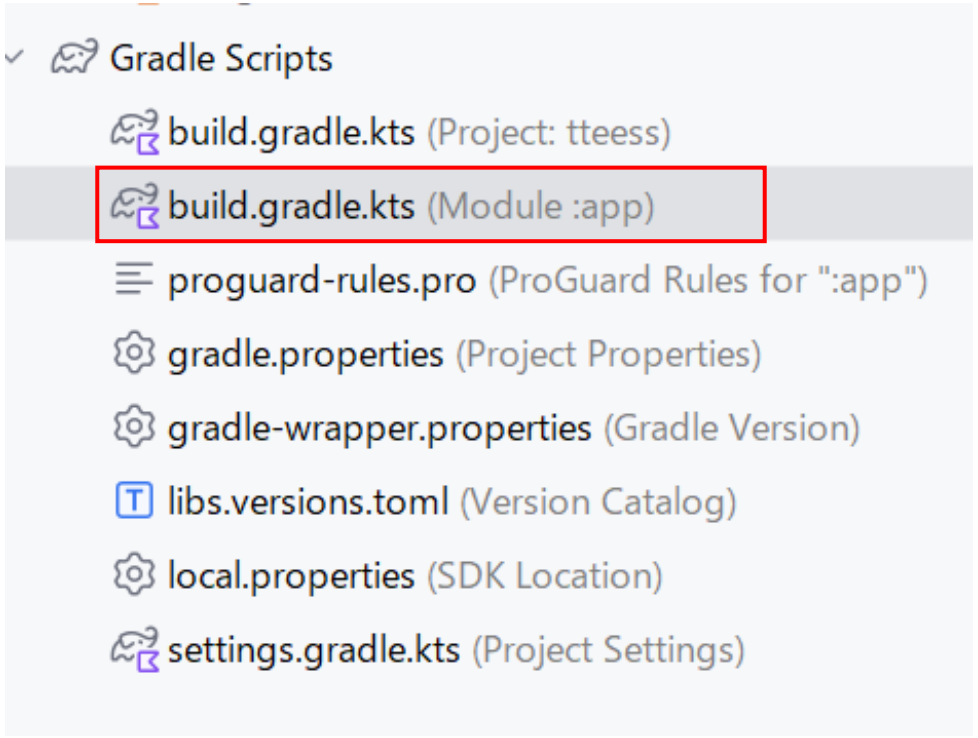
SecondScreen.kt

```
@Composable
fun SecondScreen(modifier: Modifier){

    Column(modifier = Modifier.fillMaxSize().padding(16.dp),
        verticalArrangement = Arrangement.Center,
        horizontalAlignment = Alignment.CenterHorizontally){
        Text("The Second Screen", fontSize = 24.sp)
        Text("Welcome to Second Screen", fontSize = 24.sp)
        Button(onClick = {}) {
            Text("To First Screen")
        }
    }
}
```

Add the navigation dependencies

- Make sure you've added the correct dependency in your build.gradle (app-level) file:




```
dependencies {
    val nav_version = "2.8.8"

    implementation("androidx.navigation:navigation-compose:$nav_version")
    implementation(libs.androidx.core.ktx)
    implementation(libs.androidx.lifecycle.runtime.ktx)
```

- Sync the Project:After making these changes, sync your project in Android Studio to download the dependency.
- Cache/Index Issues:If everything seems correct but the dependency still isn't found, try cleaning the project and rebuilding it. You can also invalidate caches via File > Invalidate Caches / Restart... in Android Studio.

Add the navigation function to the Screen

FirstScreen.kt

```
@Composable
fun FirstScreen(modifier: Modifier, navigationToSecondScreen:()->Unit){
    val name = remember { mutableStateOf("") }
    Column(modifier = Modifier.fillMaxSize().padding(16.dp),
                ....
        Button(onClick = {
            navigationToSecondScreen()
        }) {
            Text("To Second Screen")
        }
    }
}
```

SecondScreen.kt

```
@Composable
fun SecondScreen(modifier: Modifier, navigateToFirstScreen:()->Unit){
    ...
        Text("Welcome to Second Screen", fontSize = 24.sp)
        Button(onClick = {
            navigateToFirstScreen()
        }) {
            Text("To First Screen")
        }
    }
}
```

Prepare the NavHost & NavController

Initializes a navigation controller with rememberNavController()

Create a navHost and use the navController to manage navigation between composable screens.

MainActivity.kt

```
fun Navitest(){
    val navController = rememberNavController()
    NavHost(navController = navController, startDestination = "first_screen"){
        composable("first_screen"){}
        composable("second_screen"){}
    }
}
```

Define two routes

Define the navigations for each screen

Add modifier parameter, and remove the modifier parameter in FirstScreen() and SecondScreen()

MainActivity.kt

```
@Composable
fun Navitest(modifier: Modifier){
    val navController = rememberNavController()
    NavHost(navController = navController, startDestination = "first_screen"){
        composable("first_screen"){
            FirstScreen {
                navController.navigate("second_screen")
            }
        }
        composable("second_screen"){
            SecondScreen {
                navController.navigate("first_screen")
            }
        }
    }
}
```

Passed a lambda function as an argument. This lambda defines what should happen when a user triggers an event (like a button click) inside FirstScreen

MainActivity.kt

```
NavigationSampleTheme {
    Scaffold(modifier = Modifier.fillMaxSize()) {
innerPadding ->
        Navitest(
            modifier = Modifier.padding(innerPadding)
        )
    }
}
```

Modify the function parameters to tranfer the data by Navigation (1/3)

FirstScreen.kt

```
@Composable
fun FirstScreen(navigationToSecondScreen:(String)->Unit){
    val name = remember { mutableStateOf("") }
    Column(modifier = Modifier.fillMaxSize().padding(16.dp),
        verticalArrangement = Arrangement.Center,
        horizontalAlignment = Alignment.CenterHorizontally){
        Text("The First Screen", fontSize = 24.sp)
        Spacer(modifier = Modifier.height(16.dp))
        OutlinedTextField(value = name.value, onValueChange = {
            name.value = it
        })
        Button(onClick = {
            navigationToSecondScreen(name.value)
        }) {
            Text("To Second Screen")
        }
    }
}
```

This is a lambda function that takes a String as input and returns Unit (i.e., no return value). This lambda is used to handle navigation to the second screen, passing the entered text along.

When the button is clicked, it triggers the navigationToSecondScreen lambda, passing the current value of name.value. This is how the user's input is sent to the next screen.

Modify the function parameters to tranfer the data by Navigation (2/3)

SecondScreen.kt

```
@Composable
fun SecondScreen(name:String, navigateToFirstScreen:()->Unit){

    Column(modifier = Modifier.fillMaxSize().padding(16.dp),
        verticalArrangement = Arrangement.Center,
        horizontalAlignment = Alignment.CenterHorizontally)
    {
        Text("The Second Screen", fontSize = 24.sp)
        Text("Welcome $name to Second Screen", fontSize = 24.sp)
        Button(onClick = {
            navigateToFirstScreen()
        }) {
            Text("To First Screen")
        }
    }
}
```

This parameter receives a string, the user's input from the first screen

A lambda function that, when invoked, will navigate the user back to the first screen.

Use string interpolation to display a personalized welcome message that includes the value of name.

calls the navigateToFirstScreen lambda, which is intended to handle the navigation back to the first screen.

Modify the function parameters to tranfer the data by Navigation (3/3)

MainActivity.kt

```
@Composable
fun Navitest(modifier: Modifier){
    val navController = rememberNavController()
    NavHost(navController = navController, startDestination = "first_screen"){
        composable("first_screen"){
            FirstScreen {name->
                navController.navigate("second_screen/$name")
            }
        }
        composable('second_screen/{name}'){
            val name = it.arguments?.getString("name") ?: "no name"
            SecondScreen(name) {
                navController.navigate("first_screen")
            }
        }
    }
}
```

FirstScreen Composable: The function FirstScreen is invoked here. It is passed a lambda that takes a name parameter.

This navigates to the route "second_screen/{name}", replacing {name} with the actual value provided

The route is defined with a placeholder {name}, indicating that a string value is expected as part of the route.

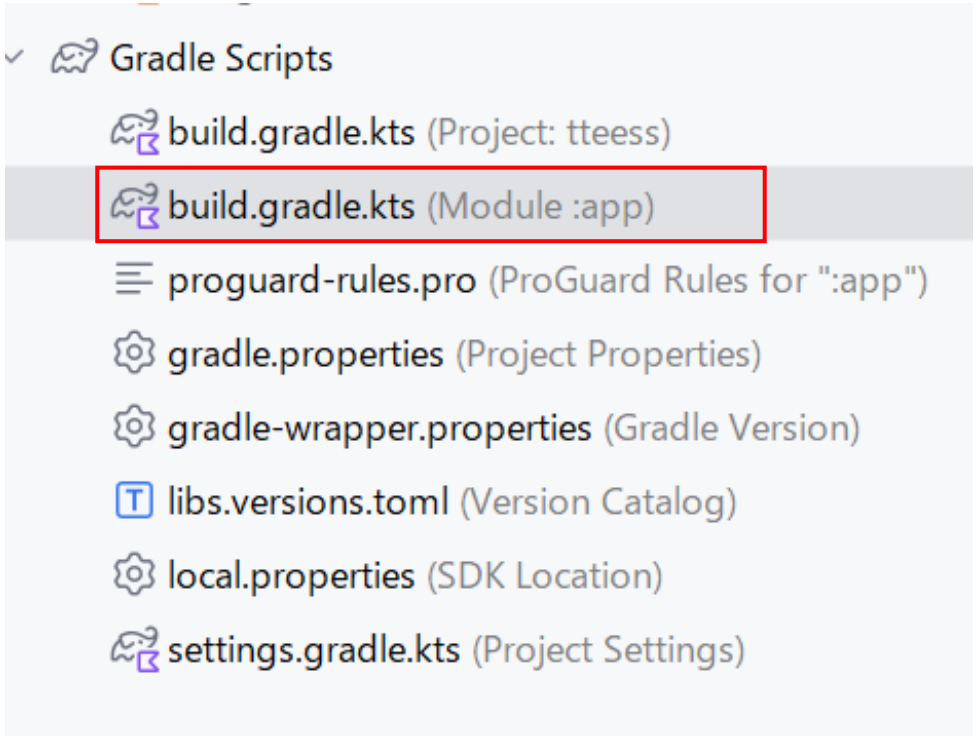
the navigation argument "name" is retrieved. If it's not found, it defaults to "no name".

Add the details Screen for the Shopping App

1. Open the Shoppingapp Project
2. Add the navigation dependencies
3. Prepare the UI for Navigation
4. set up the Routes
5. Implement the Navigation and pass the data
6. Serialization and Deserialization with Parceable

Add the navigation dependencies

- Make sure you've added the correct dependency in your build.gradle (app-level) file:




```
dependencies {
    val nav_version = "2.8.8"

    implementation("androidx.navigation:navigation-compose:$nav_version")
    implementation(libs.androidx.core.ktx)
    implementation(libs.androidx.lifecycle.runtime.ktx)
```

- Sync the Project:After making these changes, sync your project in Android Studio to download the dependency.
- Cache/Index Issues:If everything seems correct but the dependency still isn't found, try cleaning the project and rebuilding it. You can also invalidate caches via File > Invalidate Caches / Restart... in Android Studio.

Setup the product item can be clickable (1/2)

ShoppingScreen.kt

```
@Composable
fun ProductItem(product: Product,
                navigationToDetail: (Product) -> Unit
                ){
    Column(modifier = Modifier.padding(8.dp)
                        .fillMaxSize()
                        .clickable { navigationToDetail(product) },
        horizontalAlignment =  Alignment.CenterHorizontally)
    {
        Image(
            painter = rememberAsyncImagePainter(product.image),
            contentDescription = "the picture of our products",
            modifier = Modifier.fillMaxSize().aspectRatio(1f)
        )

        Text(
```

Add a lambda function that accepts a Product object. This function is called when the product item is clicked,

Wraps the entire column in a clickable area. When the column is clicked, it triggers the provided navigation lambda, passing the current product.

Setup the product item can be clickable (2/2)

ShoppingScreen.kt

```
fun ProductScreen(products: List<Product>,
                  navigationToDetail: (Product) -> Unit){
    LazyVerticalGrid(GridCells.Fixed(2), modifier = Modifier.fillMaxSize()) {
        items(products){
            product ->
            ProductItem(product = product, navigationToDetail)
…
```

Add the lambda function that takes a Product and handles navigation to a detailed view of that product.

ShoppingScreen.kt

```
fun ShoppingScreen(modifier: Modifier = Modifier,
                   navigationToDetail: (Product) -> Unit){
    val productViewModel: MainViewModel = viewModel()
    …
            else ->{
                ProductScreen(products = viewstate.list,navigationToDetail)
            }
        }
    }
}
```

Passed down to handle clicks on a product item, which will navigate to its detailed view.

Set up the Routes with a Sealed Class

Add a sealed class to define the route for navigation

Screen.kt

```
sealed class Screen(val route:String) {
    object ShoppingScreen:Screen("main_screen")
    object ItemDetailScreen:Screen("detail_screen")
}
```

By naming the screens (ShoppingScreen and ItemDetailScreen) and associating them with specific route strings, it becomes clear what each screen represents and how they are navigated to within the app.

Prepare the item detail screen

ProductDetailScreen.kt

```
@Composable
fun ProductDetailScreen(product: Product){
    Column(modifier = Modifier.fillMaxSize()
                        .padding(16.dp),
           horizontalAlignment = Alignment.CenterHorizontally
    ) {
        Text(text = product.title, textAlign= TextAlign.Center)
        Image(
            painter = rememberAsyncImagePainter( product.image),
            contentDescription = "${product.title} image",
            modifier = Modifier.wrapContentSize().aspectRatio(1f)
        )
        Text(text= product.description,
            textAlign = TextAlign.Justify,
            modifier =  Modifier.verticalScroll(rememberScrollState()))
    }
}
```

Implement the navigation and pass the data

ShoppingScreen.kt

```
fun ShoppingScreen(modifier: Modifier = Modifier,
                   viewstate:MainViewModel.ProductState,
                   navigationToDetial: (Product) -> Unit){
    val productViewModel: MainViewModel = viewModel()
    val viewstate by productViewModel.productState
```

ShoppingApp.kt

```
@Composable
fun ShoppingApp(navController: NavHostController, modifier: Modifier){
    val productViewModel: MainViewModel = viewModel()
    val viewstate by productViewModel.productState

    NavHost(navController = navController, startDestination = Screen.ShoppingScreen.route){
        composable(route=Screen.ShoppingScreen.route){
            ShoppingScreen(viewstate = viewstate, navigationToDetail = {
                navController.currentBackStackEntry?.savedStateHandle?.set("item", it)
                navController.navigate((Screen.ItemDetailScreen.route))
            } )
        }
        composable(route = Screen.ItemDetailScreen.route) {
            val product = navController.previousBackStackEntry?.savedStateHandle?.
            get<Product>("item") ?: Product(0,"",0.0,"","")
            ProductDetailScreen(product=product)
        }
    }
}
```

Add the viewstate parameter to ShoppingScreen()

The code observes the productState from the view model using the by delegate. The state (stored in viewstate) likely contains information such as a list of products, loading status, and error information.

Continue to explain the navigation

ShoppingApp.kt

```
@Composable
fun ShoppingApp(navController: NavHostController, modifier: Modifier){
    val productViewModel: MainViewModel = viewModel()
    val viewstate by productViewModel.productState

    NavHost(navController = navController, startDestination = Screen.ShoppingScreen.route){
        composable(route=Screen.ShoppingScreen.route){
            ShoppingScreen(viewstate = viewstate, navigationToDetail = {
                navController.currentBackStackEntry?.savedStateHandle?.set("item", it)
                navController.navigate((Screen.ItemDetailScreen.route))
            } )
        }
        composable(route = Screen.ItemDetailScreen.route) {
            val product = navController.previousBackStackEntry?.savedStateHandle?.
            get<Product>("item") ?: Product(0,"",0.0,"","")
            ProductDetailScreen(product=product)
        }
    }
}
```

This is the container that manages the navigation graph. It uses the provided navController and sets the start destination to the route defined by Screen.ShoppingScreen.route.

Inside the lambda, the selected product is stored in the current back stack entry's savedStateHandle with the key "item".Then, the app navigates to the ItemDetailScreen by calling navController.navigate(Screen.ItemDetailScreen.route).

The retrieved product is then passed to the ProductDetailScreen composable, which displays its details.

The detail screen retrieves the product data from the savedStateHandle of the previous back stack entry using the key "item".If no product is found (for example, if something went wrong), a default Product is created as a fallback.

Add the navigation to MainActivity.kt

```
class MainActivity : ComponentActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        enableEdgeToEdge()
        setContent {
            val navController = rememberNavController()
            ShoppingAppTheme {
                Scaffold(modifier = Modifier.fillMaxSize()) { innerPadding ->
                    ShoppingApp(navController,
                        modifier = Modifier.padding(innerPadding)
                    )
}
```

Initializes and remembers a navigation controller, which manages the app's navigation state and back stack. It is later passed to the ShoppingApp composable to enable navigation between screens.

Serialization and Deserialization with Parcelable

- Add the plugins for Parcelabe

build.gradle

```
plugins {
    alias(libs.plugins.android.application)
    alias(libs.plugins.kotlin.android)
    alias(libs.plugins.kotlin.compose)
    id("kotlin-parcelize")
}
```

Implementing Parcelable allows instances of the Product class to be serialized.

```
@Parcelize
data class Product(
    val id: Int,
    val title: String,
    val price: Double,
    val description: String,
    val image: String
):Parcelable
```

@Parcelize Annotation:This is a Kotlin-specific annotation that automatically generates the necessary boilerplate code for the Parcelable interface. By using @Parcelize, you don't have to manually implement methods like writeToParcel() and describeContents().

Parcelable Interface:Implementing Parcelable allows instances of the Product class to be serialized. This is particularly useful in Android for passing objects between activities or fragments via Intent extras or Bundles.