Type-Safe Navigation in Jetpack Compose

Navigate from Source to Destination

nav·i·gate

1. plan and direct the route of a ship, aircraft, or other form of transportation, especially by using instruments or maps
2. travel on a desired course after planning a route
3. guide over a specific route or terrain
4. *move from one accessible page, section, or view of a file or website to another*

Activity Navigation + Compose

- Type-Safe Navigation
  - No “unknown” destination at runtime
  - All destination objects are known at compile-time
- Navigation Between Composable within a Single Activity
  - Transition from one screen to another
  - Passing data payload forward (from source to destination)
  - Returning data payload backward (from destination to source)
- ~~Navigation Across Activities~~ (not included here)
  - Transition from one Activity to another
  - The UI of each activity is built using Jetpack Compose

UI Navigation in Jetpack Compose

Project Setup

```
// Edit your Module's build.gradle.kts
// Add new plugin
plugins {
  // WARNING: there was potentially a build issue when the plugin version
  // does not match the Kotlin compiler version. Most of the time the fix
  // is to use a slightly older version of the Kotlin compiler
  kotlin("plugin.serialization") version "2.0.0"
}
android {
  compileSdk = 35
}
// WARNING: the serialization plugin (above) and the JSON serialization
// library (below) must be carefully picked to avoid incompatibility!!!
// Kotlin Serialization
implementation("org.jetbrains.kotlinx:kotlinx-serialization-json:1.8.0")

// Add Library Dependency (2.8.0 or newer. 2.9.x requires Kotlin 2.0)
implementation("androidx.navigation:navigation-compose:2.8.0") // or newer
implementation("androidx.navigation:navigation-runtime-ktx:2.8.0")
```

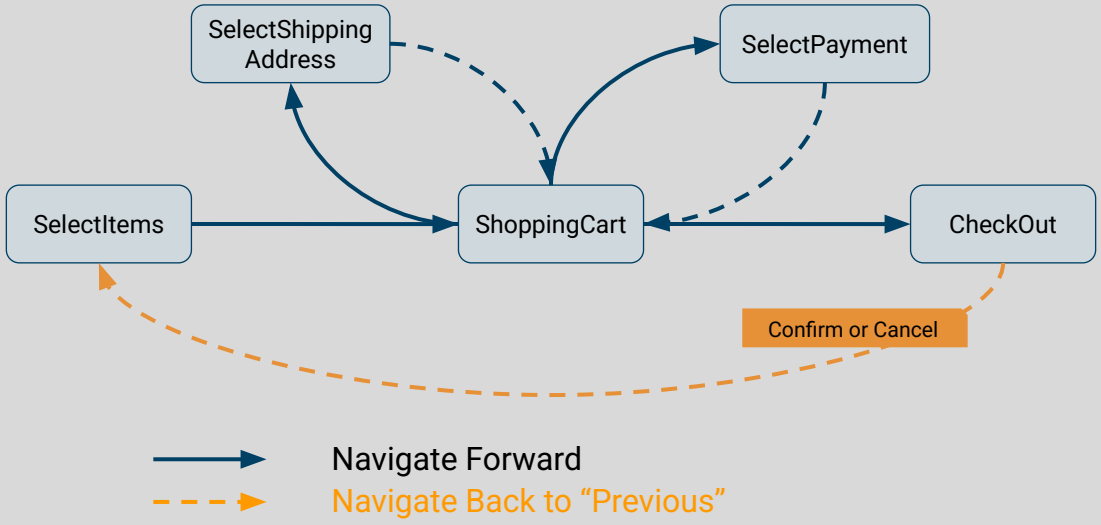
Elements of Navigation

Road Trip	UI Navigation
City of origin	@Composable Screen of origin
City of destination	@Composable Screen of destination
Map	Navigation Host
Roads	Routes
Driver	Navigation Controller

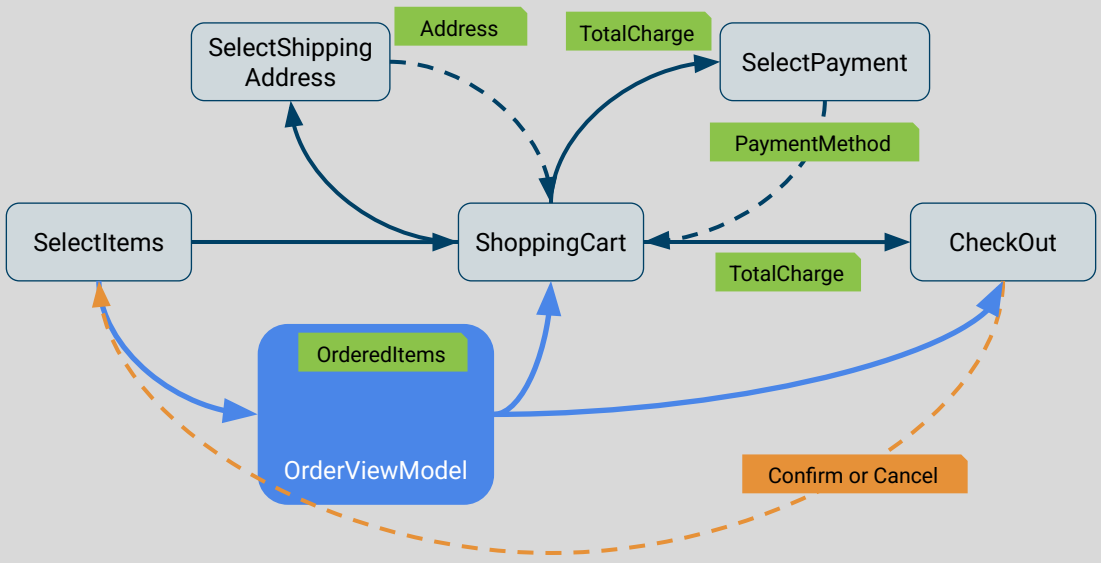
Compose Navigation Router

- Required Components
  - Source/Destination: UI @Composable functions
  - A Navigation Host (@Composable) at the “top-level” Composable
  - An instance of a NavigationController
- Similar to Routers in Web Apps
  - React Router
  - Vue Router

Step 1a: Create the Screen Graph

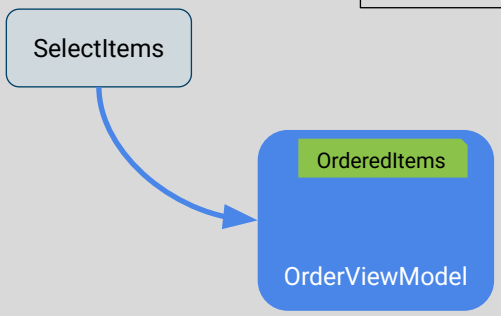


Step 1b: Create the Screen Graph & Data Flow

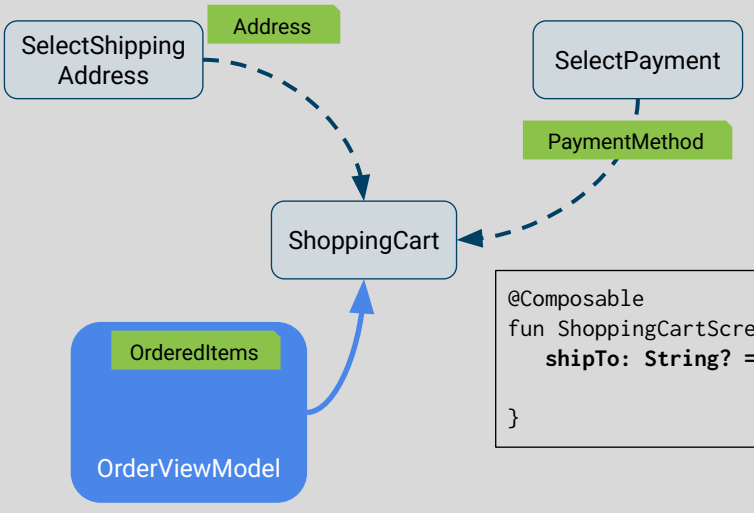


SelectItemsScreen

```
@Composable
fun SelectItemsScreen(vm: OrderViewModel) {

}
```

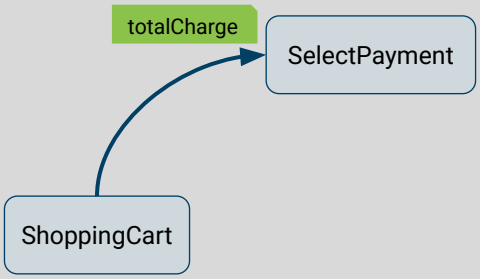


ShoppingCartScreen




```
@Composable
fun ShoppingCartScreen(vm: OrderViewModel,
   shipTo: String? = null, payWith: String? = null) {

}
```

SelectPaymentScreen



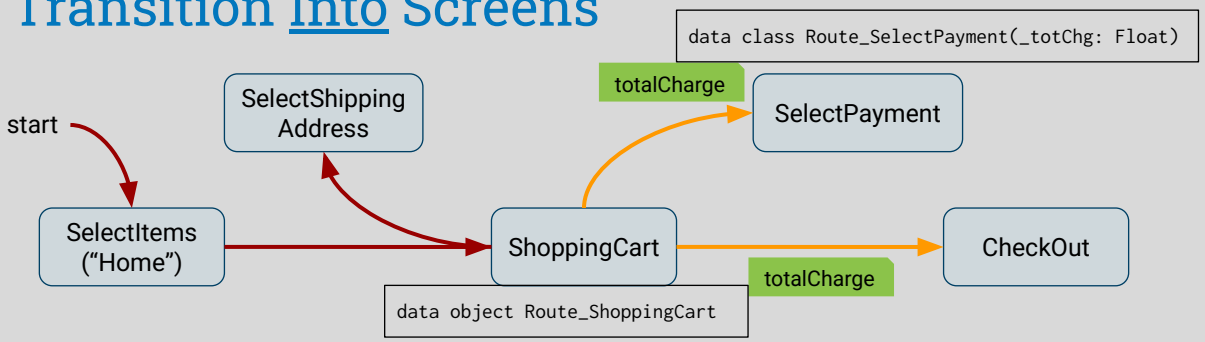

```
@Composable
fun SelectPaymentScreen(totalCharge: Float) {

}
```


Step 2: Define Each Screen (& Params)

```
@Composable
// SelectItems and ShoppingCart are linked to the OrderViewModel
fun SelectItemsScreen(vm: OrderViewModel) {   }
fun ShoppingCartScreen(vm: OrderViewModel) {   }

// Transition INTO SelectShippingAddress does not carry input data
fun SelectShippingAddressScreen(shipTo: String? = null, payWith: String? = null) {  }

// Transition INTO SelectPayment requires the current total charge
fun SelectPaymentScreen(totalCharge: Float) { }

fun CheckOutScreen(totalCharge: Float, vm: OrderViewModel) {  }
```

Compare the parameter(s) of each function with the screen graph

Transition Into Screens



- Transition into (incoming arrows) SelectItems, ShoppingCart, SelectShippingAddr does not require data
- Transition into SelectPayment and CheckOut requires data (totalCharge)

Step 3: Define Navigation Routes (One per Screen)

```
@Serializable
sealed class Route {
    @Serializable
    data object SelectItems

    @Serializable
    data object ShoppingCart

    @Serializable
    data object SelectShippingAddress

    @Serializable
    data class SelectPayment(val totalCharge: Float)

    @Serializable
    data class Checkout(val totalCharge: Float)
}
```

- *Use **data object** if the transition into the screen takes no parameter*
- *Use **data class** if it requires parameters*

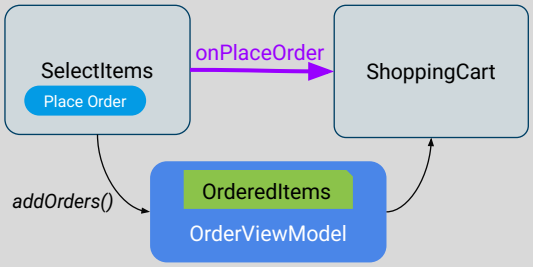
Step 4: Associate Routes with Screens in NavHost

```
Scaffold (
   modifier = Modifier.fillMaxSize(),
) { padding ->
   val nc = rememberNavController()
   NavHost(modifier = Modifier.padding(padding),
       navController = nc, startDestination = Route.SelectItems) {
       composable<Route.SelectItems> {
           SelectItemsScreen(orderViewModel)
       }
       // include ALL screen destinations
       composable<Route.SelectPayment> {
           SelectPaymentScreen(20f)  // Supply dummy argument (temporary)
       }
       composable<Route.Checkout> {
           CheckOutScreen(totalCharge = 50f, vm = orderViewModel)
       }
   }
}
```


Where To Create ViewModel Instances?

```
override fun onCreate(_____) {
  super.onCreate(_____)
  val orderViewModel by viewModels<OrderViewModel>()
  setContent {
    Scaffold { padding ->
      val nc = rememberNavController()
      NavHost(modifier = Modifier.padding(padding),
        navController = nc, startDestination = Route.SelectItems) {
        composable<Route.SelectItems> {
          SelectItemsScreen(orderViewModel)
        }
        composable<Route.Checkout> {
          CheckOutScreen(totalCharge = 50f, vm = orderViewModel)
        }
        /* more routes not shown */
     }
  }
}
```

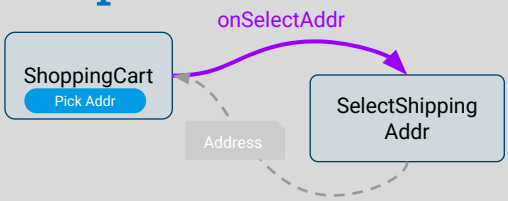
Step 5a: Add Lambda(s) To Trigger Navigation




```
@Composable
fun SelectItemsScreen(vm: OrderViewModel,
                      onPlaceOrder: () -> Unit) {
  Column {
    // Other widgets here
    Button(
      onClick = {
        vm.addOrders(listOf("Boot","Sock"))
        onPlaceOrder()
      }) {
      Text("Place Order")
    }
  }
```

```
NavHost(navController = nc, startDestination = Route.SelectItems) {
composable<Route.SelectItems> {
  SelectItemsScreen(vm = orderViewModel, onPlaceOrder = { nc.navigate(Route.ShoppingCart) })
  /* OR */ SelectItemsScreen(vm = orderViewModel) { nc.navigate(Route.ShoppingCart) }
}
composable<Route.ShoppingCart> {
  ShoppingCarScreen(orderViewModel)
}
```

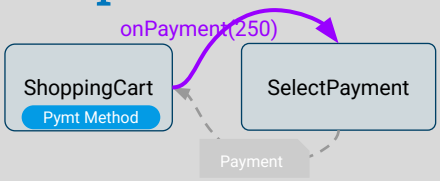

Step 5b: Add Lambdas without arguments




```
@Composable
fun ShooppingCartScreen(vm: OrderViewModel,
                        onSelectAddr: () -> Unit) {
  Button(onClick = { onSelectAddr() }) {
    Text("Pick Address")
  }
}
```

```
NavHost(navController = nc, startDestination = Route.SelectItems) {
composable<Route.ShoppingCart> {
  ShoppingCartScreen(vm = orderViewModel,
     onSelectAddr = { nc.navigate(Route.SelectShippingAddress)  })
}
```

Step 5c: Add Lambdas with argument(s)

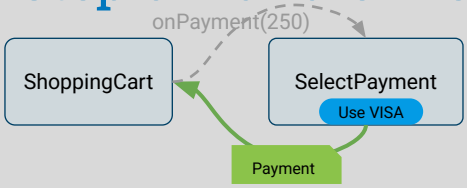



```
sealed class Route {
    data class SelectPayment(val totalCharge: Float)
}

@Composable
fun SelectPaymentScreen(___: Float) { }
```

```
@Composable
fun ShoppingCartScreen(vm: OrderViewModel,
                       onPayment: (Float) -> Unit) {
  Button(onClick = { onPayment(250f)}) {
    Text("Payment Method")
  }
}
@Composable
fun SelectPaymentScreen(amount: Float) {}
```

```
NavHost(navController = nc, startDestination = Route.SelectItems) {
  composable<Route.ShoppingCart> {
    ShoppingCartScreen(orderViewModel,
             onPayment    = { amt ->
                              val dest = Route.SelectPayment(amt)
                              nc.navigate(dest) })
  }
  composable<Route.SelectPayment> {
    val howmuch = it.toRoute<Route.SelectPayment>()
    SelectPaymentScreen(amount = howmuch.totalCharge)
  }
}
```


Step 6: Handle "Return" Objects (via Stack)



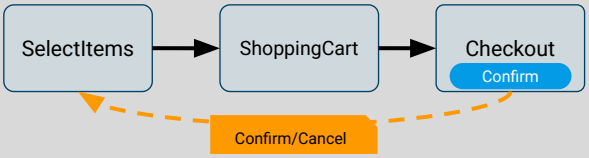

```
@Composable
fun SelectPaymentScreen(totalCharge: Float,
                        onPayBy: (String) -> Unit) {
  Button(onClick = { onPayBy("1111 2222 3333 4444") }) {
    Text("Use VISA")
  }
}
```

```
NavHost(navController = nc, startDestination = Route.SelectItems) {
composable<Route.SelectPayment> {
  SelectPaymentScreen(/* args */) /* onPayBy */ { myPayMethod ->
    nc.previousBackStackEntry?.saveStateHandle?.set("PAY_WITH", myPayMethod)
    nc.popBackStack()  // DO_NOT_USE_nc.navigate() !!!!!
  }
}
composable<Route.ShoppingCart> {
  val paymentMethod = nc.saveStateHandle.get<String>("PAY_WITH")
  ShoppingCartScreen(vm = orderViewModel, payWith = paymentMethod, /* other args */)
}
```

Save to Stack

Get from Stack

Pop Back To “Top”




```
@Composable
fun CheckOutScreen(/* other params */,
                   onCheckout: (Boolean) -> Unit) {
  Button(onClick = { onCheckout(true) }) {
    Text(“Confirm”)
  }
}
```

```
NavHost(navController = nc, startDestination = Route.SelectItems) {
composable<Route.Checkout> {
  CheckoutScreen(/* args */) /* onCheckOut */ { confirmed->
    if (confirmed) {
      // Complete transaction
    }
    // WARNING: DO NOT USE nc.navigate() !!!!!
    nc.popBackStack(route = Route.SelectItems, inclusive = false)
  }
}
```


GitHub

android-compose-navigation

android-compose-navigation

Adding Back Button (Optional)

Back Button as Navigation Icon

```
val nc = rememberNavController()
Scaffold (
   topBar = {
       TopAppBar(
           title = {Text("ABC Shopper")},
           navigationIcon = {
               IconButton(onClick = { nc.popBackStack() }) {
                   Icon(Icons.Default.ArrowBack, contentDescription = "Back")
               }
           }
       )
   }
) { padding ->
   NavHost(modifier = Modifier.padding(padding), /* args */) {
       composable<Route.____> {   }
       composable<Route.____> {   }
   }
}
```

Navigation with Intent

Android `Intent` (Objects that represent work)

This technique can be used to navigate another activity

- Inside your own app (*irrelevant for Jetpack Compose)*
- Outside your own app
  - Required: package name of the external app /activity
  - Optional: additional information needed by the external app
- Intent objects
  - Without Result: ACTION_SET_ALARM, ACTION_VIEW, ACTION_CALL, ...
  - With (return) Result: ACTION_IMAGE_CAPTURE, ACTION_VIDEO_CAPTURE, ...

General Pattern for Launching External Activity

Without Result

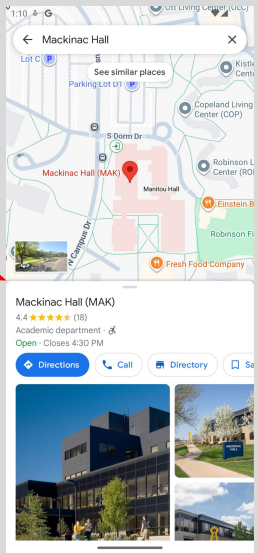
```
val context = LocalContext.current
Button(onClick = {
   val xyzIntent = Intent(Intent.ACTION_XYZ, /* args */)
   xyzIntent.setPackage("______")
   try {
       context.startActivity(xyzIntent)
   } catch (e: Exception) {
   }
}) {
   Text("Do It")
}
```

```
val contract = ActivityResultContracts.StartActivityForResult()
val xyzLauncher = rememberLauncherForActivityResult(contract) { res ->
   if (res.resultCode == RESULT_OK) {
       // Unpack the result
   }
}

Button(onClick = {
   val xyzIntent = Intent(Intent.ACTION_XYZ, /* args */)
   xyzIntent.setPackage("______")
   xyzLaunher.launch(kontakIntent)
 }) {
   Text("Do It")
}
```

With Result

Launching Intent (with no return data)

```
// WARNING: require virtual dev with Google Play Services
val context = LocalContext.current
Button(onClick = {
   val gmap = Intent(Intent.ACTION_VIEW,
          Uri.parse("geo:0,0?q=Mackinac+Hall")).also {
       it.setPackage("com.google.android.apps.maps")
   }
   try {
       context.startActivity(gmap)
   } catch (e: Exception) {
       println("Can't use maps: $e")
   }
}) {
   Text("By Map")
}
```



Launching Intent (with return data)

```
val contract = ActivityResultContracts.StartActivityForResult()
val pickKontakLauncher = rememberLauncherForActivityResult(contract) { result ->
   if (result.resultCode == RESULT_OK) {
       println("Result: ${result.resultCode}, ${result.data?.data}")
       // Unpack the result
   } else {
   }
}

Button(onClick = {
   val kontakIntent = Intent(Intent.ACTION_PICK,
       ContactsContract.CommonDataKinds.StructuredPostal.CONTENT_URI)
   pickKontakLauncher.launch(kontakIntent)
 }) {
   Text("Search Contact")
}
```