

CMPS 312

Read Chapters
12, 13 & 15



Navigation

Dr. Abdelkarim Erradi
CSE@QU

Navigation

The act of **moving between screens** of an app to **complete tasks**

Designing effective navigation =
Simplify the user journey

Outline

1. Communicating Between Activities
2. Dialogs
3. Navigation UI
4. Navigation Component

Communicating Between Activities



Using Multiple Activities

- How do we **navigate** to a new screen?

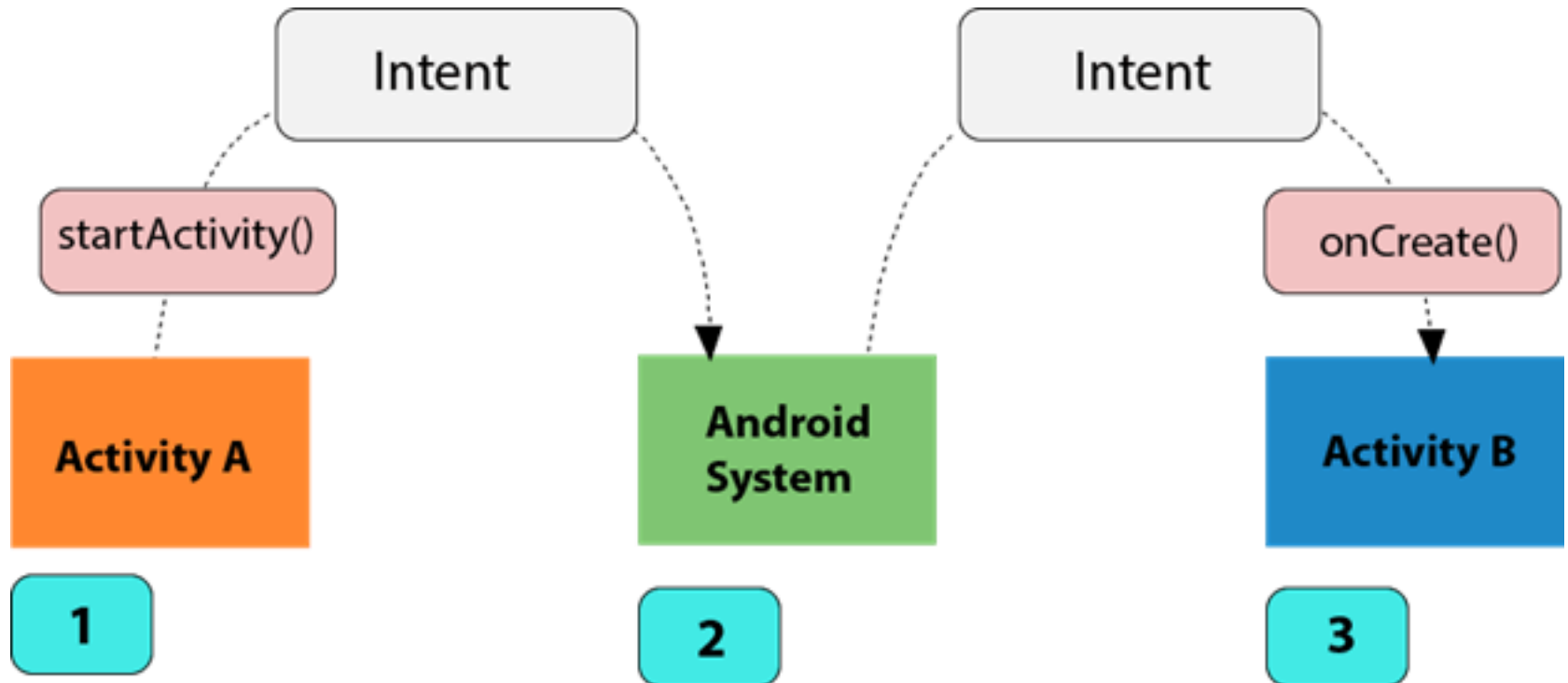
➤ Start a new Activity using an **Intent**

```
val intent = Intent(this, RegisterActivity::class.java)  
startActivity(intent)
```

- What is an **Intent**?
 - Enables communication between Activities
 - It is a **messaging object** to communicate to the system that some action should be carried out
 - **Implicit** vs **Explicit** Intents: choosing a generic action vs starting a specific app component

Explicit Intent

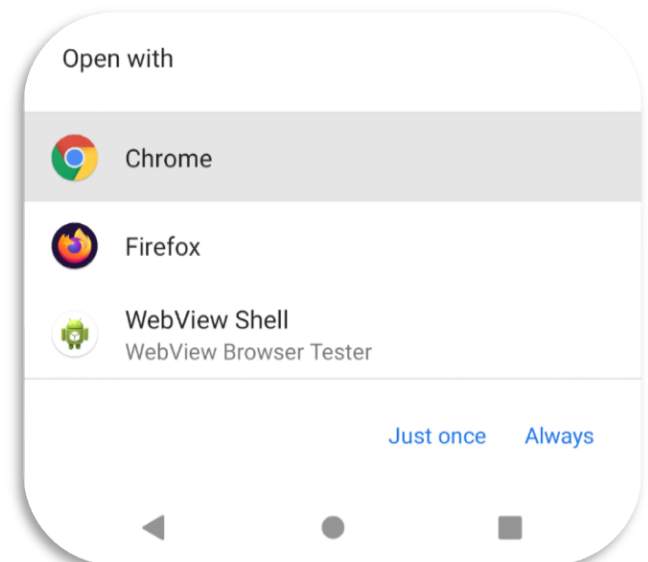
- **Explicit** intents can be used to start a specific Activity
`intent = Intent(this, RegisterActivity::class.java)`
`startActivity(intent)`



Implicit Intent

- Implicit intents describe a general action (without specifying a component to handle it) such as display contacts, broadcast a message, dial a phone call etc.
 - **Display contact:** ACTION_VIEW -> content://contacts/people/1
 - **Dial a number:** ACTION_DIAL -> content://contacts/people/1
 - **Send an email:** ACTION_SEND -> EXTRA_EMAIL, EXTRA_SUBJECT
 - Specifies an **ACTION** and **DATA** (parameters expected by the action)
 - Implicit intents can be handled by **a component in the system** registered to handle that intent type

```
val intent = Intent(Intent.ACTION_VIEW,  
Uri.parse("https://www.qu.edu.qa"))  
startActivity(intent)
```



Passing Data with Intents

- Pass data

```
val intent = Intent(this, RegisterActivity::class.java)
// Pass student ID and student name with Intent so it can be
// used by RegisterActivity when it's started
intent.putExtra("id", 235789)
intent.putExtra("name", "Peter Pan")
startActivity(intent)
```

- Get passed data

```
override fun onCreate(savedInstanceState: Bundle?) {
    ...
    // Read data sent by the caller
    val id = intent.getIntExtra("id", 0)
    val name = intent.getStringExtra("name")
}
```


Dialogs



Dialog Box

- Dialogs are displayed in front of app content
 - Inform users about a task that may contain **critical information** and/or **require a decision**
 - Interrupt the current flow and remain on screen until dismissed or action taken. Hence, they should be used sparingly
- 3 Types:
 - **Alert dialog:** request user action/confirmation. Has a title, optional supporting text and action buttons
 - **Simple dialog:** Used to present the user with a list of actions that, when tapped, take immediate effect.
 - **Confirmation dialog:** Used to present a list of single- or multi-select choices to a user. Action buttons serve to confirm the choice(s).

Alert Dialog

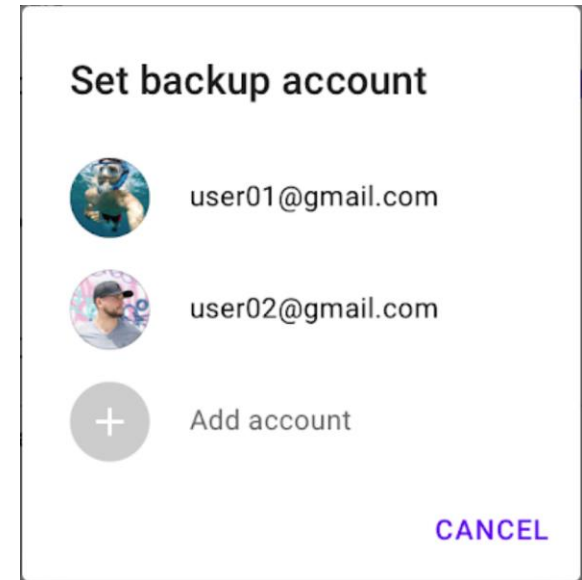
Use location service?

Let us help apps determine location. This means sending anonymous location data to us, even when no apps are running.

DISAGREE AGREE

```
MaterialAlertDialogBuilder(requireActivity())  
    .setTitle("Discard draft?")  
    .setMessage("This will permanently delete the current e-mail draft.")  
    .setPositiveButton("Discard") { dialog, which ->  
        Toast.makeText(activity, "Clicked discard", Toast.LENGTH_SHORT).show()  
    }  
    .setNegativeButton("Cancel") { dialog, which ->  
        Toast.makeText(activity, "Clicked cancel", Toast.LENGTH_SHORT).show()  
    }  
    .show()
```

Simple dialog



```
val items = arrayOf("user01@gmail.com", "user02@gmail.com", "Add account")
```

```
MaterialAlertDialogBuilder(requireActivity())
```

```
.setTitle("Set backup account")
```

```
.setItems(items) { dialog, which ->
```

```
    Toast.makeText(activity, "Clicked ${items[which]}",
```

```
        Toast.LENGTH_SHORT).show()
```

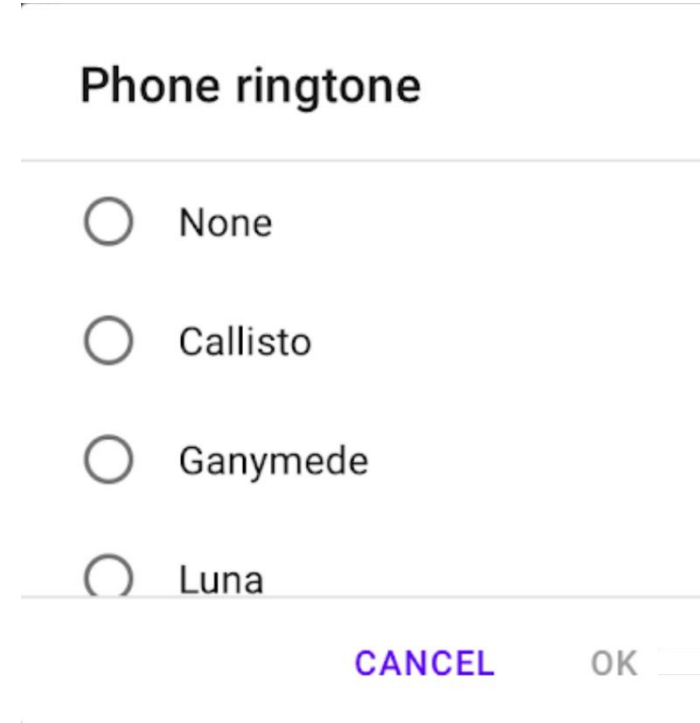
```
}
```

```
.show()
```

Confirmation dialog (single choice)

```
val items = arrayOf("None", "Callisto", "Ganymede", "Luna")  
val checkedItem = 0
```

```
MaterialAlertDialogBuilder(requireActivity())  
    .setTitle("Phone ringtone")  
    .setSingleChoiceItems(items, checkedItem) { dialog, which ->  
        Toast.makeText(activity, "Chose ${items[which]}", Toast.LENGTH_SHORT).show()  
    }  
    .setPositiveButton("Ok") { dialog, which ->  
        Toast.makeText(activity, "Clicked ok - Chose ${items[which]}", Toast.LENGTH_SHORT).show()  
    }  
    .setNegativeButton("Cancel") { dialog, which ->  
        Toast.makeText(activity, "Clicked cancel - Chose ${items[which]}", Toast.LENGTH_SHORT).show()  
    }  
    .show()
```



Confirmation dialog (multi choice)

```
val items = arrayOf("None", "Forums", "Social", "Updates")
val checkedItems = booleanArrayOf(true, false, false, false)
```

```
MaterialAlertDialogBuilder(requireActivity())
```

```
.setTitle("Label as:")
```

```
.setMultiChoiceItems(items, checkedItems) { dialog, which, checked ->
```

```
    Toast.makeText(activity, "Chose ${items[which]} - $checked",
```

```
        Toast.LENGTH_SHORT).show()
```

```
}
```

```
.setPositiveButton("Ok") { dialog, which ->
```

```
    Toast.makeText(activity, "Clicked ok", Toast.LENGTH_SHORT).show()
```

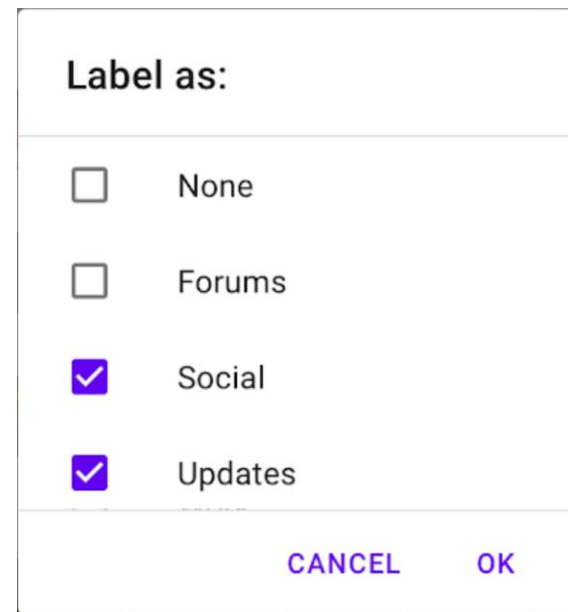
```
}
```

```
.setNegativeButton("Cancel") { dialog, which ->
```

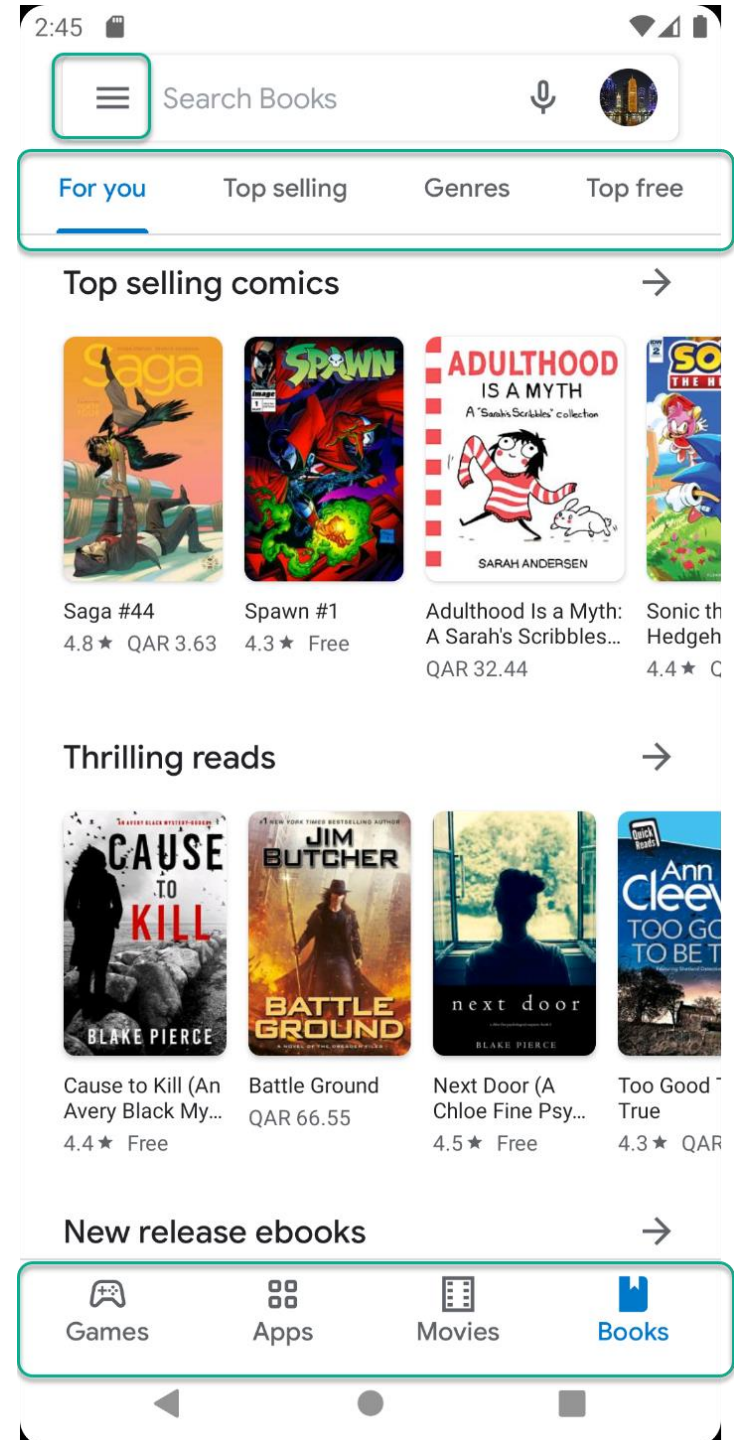
```
    Toast.makeText(activity, "Clicked cancel", Toast.LENGTH_SHORT).show()
```

```
}
```

```
show()
```



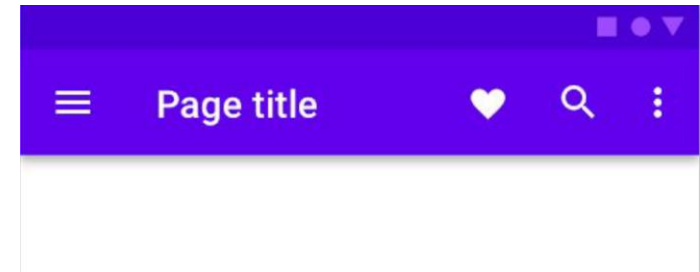
Navigation UI



Top App Bar & Bottom Navigation

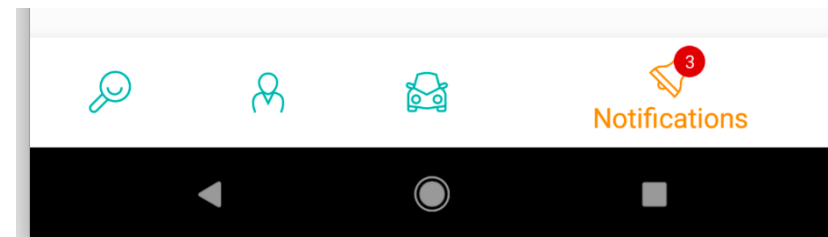
- **Top App Bar**

- Info and actions **related to the current screen**
- Typically has Title, Menu items, Drawer button / Back button



- **Bottom Navigation**

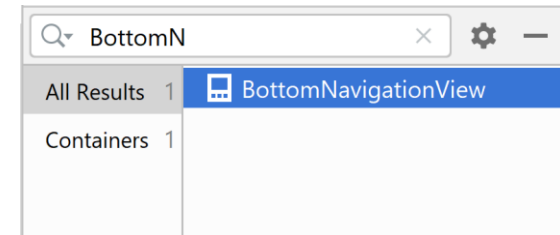
- Allow movement between the app's primary **top-level destinations** (3 to 5 options)
- Each destination is represented by an icon and an optional text label. May have notification badges



Adding Bottom Navigation

- Add **BottomNavigationView** to the main activity layout and connect it with the bottom nav menu

```
<com.google.android.material.bottomnavigation.BottomNavigationView  
    android:id="@+id/bottomNavBar"  
    ...  
    app:menu="@menu/menu_bottom_nav"/>
```

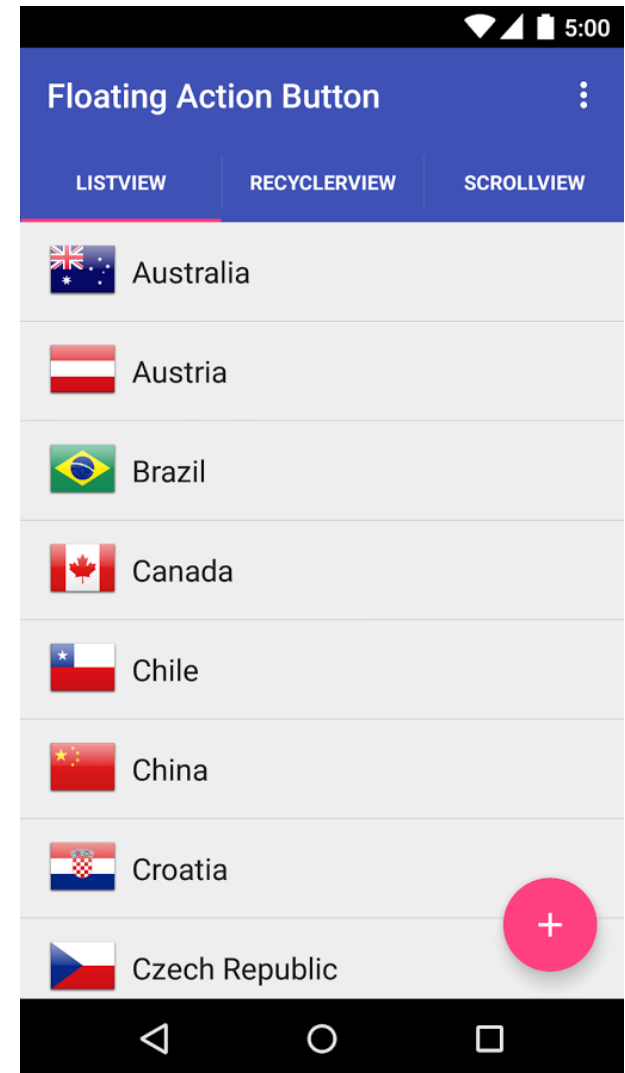


- Handle **NavigationItemSelected** event

```
bottomNavBar.setOnNavigationItemSelectedListener {  
    when (it.itemId) {  
        R.id.registerMi -> {  
            ...  
            true  
        }  
    }  
}
```

Floating Action Button (FAB)

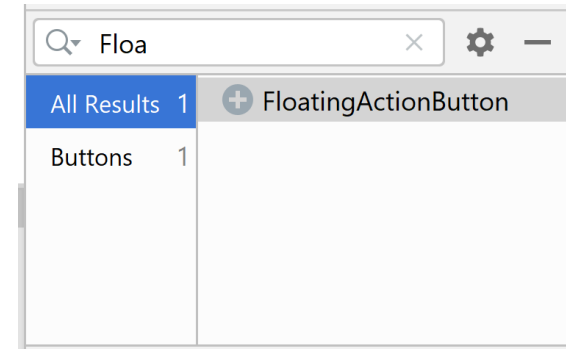
- A FAB performs the primary, or most common, action on a screen, such as drafting a new email
 - It appears in front of all screen content, typically as a circular shape with an icon in its center.
 - FAB is typically placed at the bottom right



Adding FAB

- Add **FloatingActionButton** to the fragment layout and set its layout constraints


```
<com.google.android.material.floatingactionbutton.FloatingActionButton  
    android:id="@+id/addBlogPostFab"  
    ...  
    app:layout_constraintBottom_toTopOf="parent"  
    app:layout_constraintEnd_toEndOf="parent"  
    app:srcCompat="@drawable/ic_add" />
```

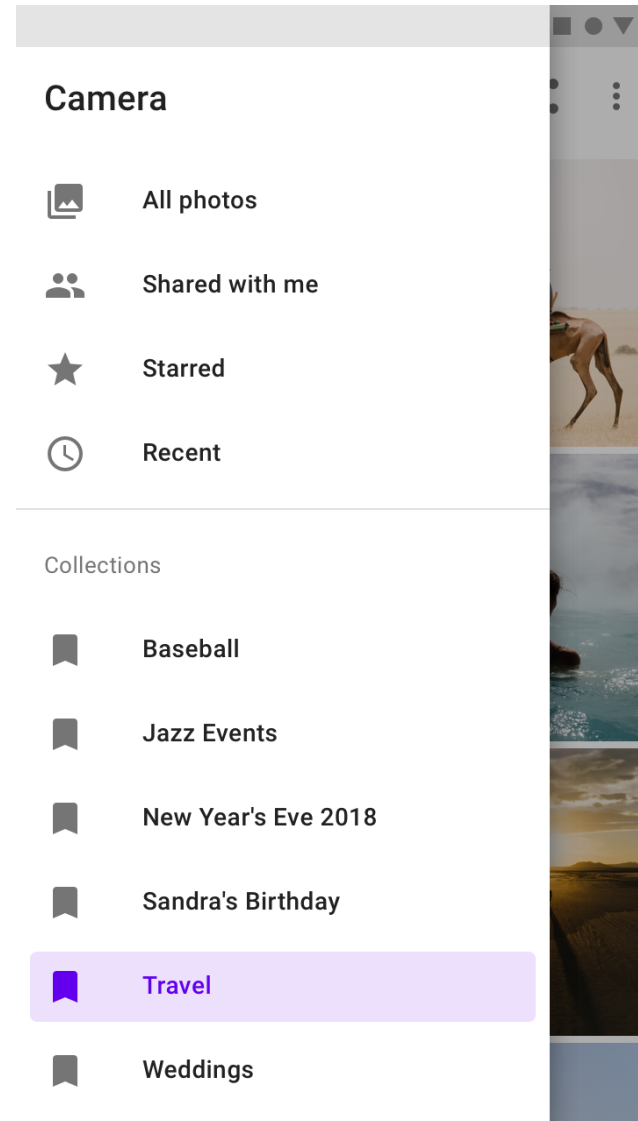


- Handle **OnClick** event

```
view.addBlogPostFab.setOnClickListener {  
    findNavController().navigate(R.id.toAddBlogPost)  
}
```

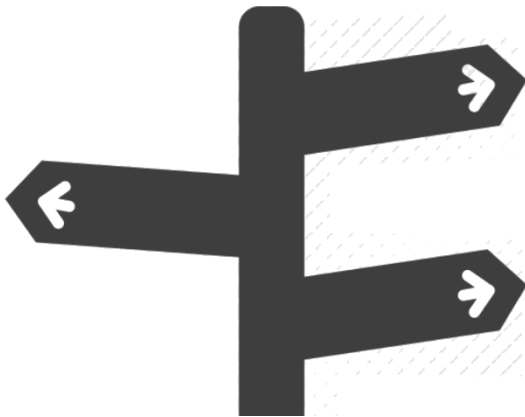
Navigation Drawer

- Navigation Drawer provides access to **primary destinations**, such as switching accounts
 - Recommended for five or more top-level destinations
 - Quick navigation between unrelated destinations
- The drawer appears when the user touches the drawer icon  in the app bar or when the user swipes a finger from the left edge of the screen
- See the example done in the Lab



Navigation Component

A framework for navigating between
'destinations' within an app

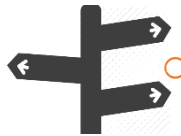


Single Activity with Multi-Fragments

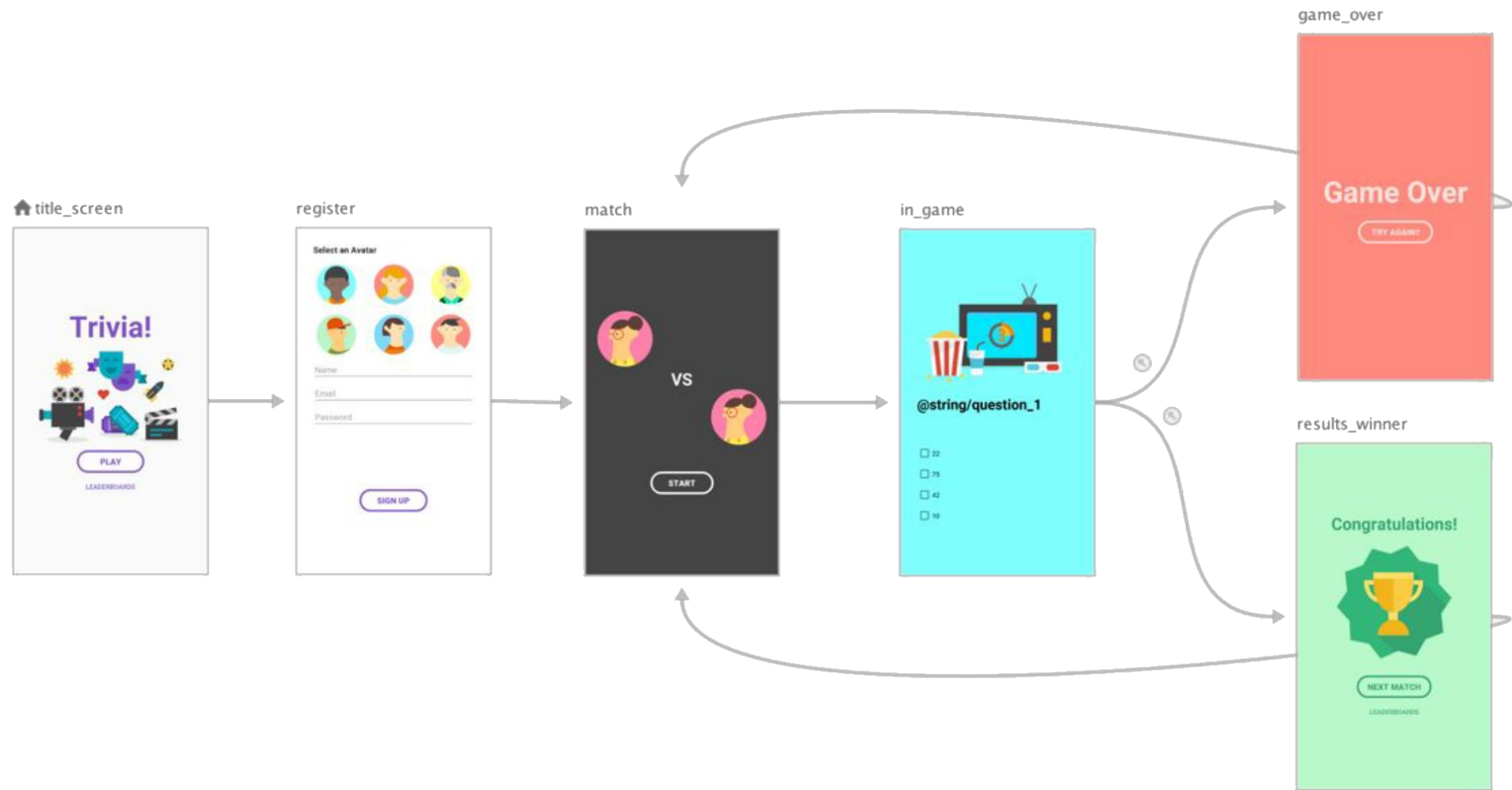
- App UI = { 1 Activity + Multi-fragments }
- A Fragment represents **a portion of the UI** in an activity
 - You can add or remove a Fragment at runtime
 - You can reuse a Fragment in multiple activities
- Like an activity:
 - A Fragment is defined in a Kotlin class
 - A Fragment's UI is defined in an XML layout file
 - A Fragment has its own lifecycle and receives its own input events

Navigation Component

- Ease implementing **Single Activity App** with Multi-Fragments
- GUI-based Editor of **Navigation Graph** to define a visual representation of app navigation flow (how users can move between screens of the app). Graph defines **Destinations & Actions**:
 - A **destination** is any place inside the app to which a user can navigate
 - **Actions** are connections between destinations and define the possible **paths** that a user can take through the app
- Integration with Navigation UI (e.g., auto show the label of current fragment on the Action Bar)



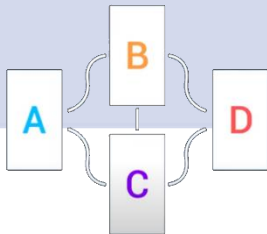
Example Navigation Graph



Key Components

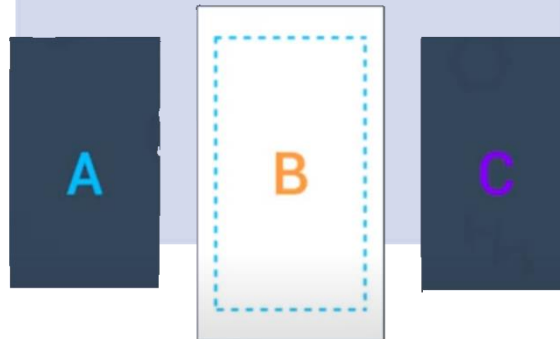
Navigation Graph

- XML representation of app navigation (**possible paths** a user can take through an app)
- Shows visually all the destinations that can be reached from a given destination



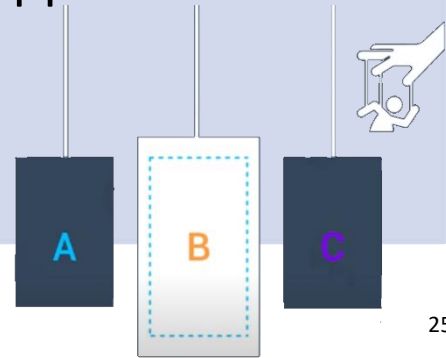
NavHost

- A container where fragments will be displayed
- **NavHostFragment** is typically used to display destination fragments



NavController

- Manages the transitions between graph destinations
- Orchestrates the **swapping** of destination fragments in the NavHost as the user navigates through the app



Implementing Navigation

Create a Nav Graph

- Create an XML file to define the app's navigation graph

Add NavHostFragment to the main activity layout

- Add **NavHostFragment** to the main activity layout. This will be the container that will display fragments as the user navigate through the app
- Associate it with the app nav graph

Navigate to destinations using the NavController

- From any view **findNavController** to navigate to a particular action
- The requested destination fragment will be loaded in the NavHostFragment

Dependencies

```
// Project/build.gradle  
def nav_version = "2.3.0"  
classpath "androidx.navigation:navigation-safe-args-gradle-plugin:$nav_version"
```

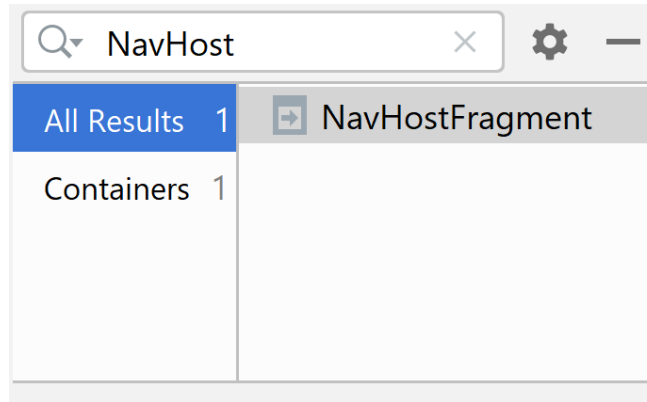
```
// Module:app/build.gradle  
def nav_version = "2.3.0"  
implementation "androidx.navigation:navigation-fragment-ktx:$nav_version"  
implementation "androidx.navigation:navigation-ui-ktx:$nav_version"
```

```
// Module:app/build.gradle  
apply plugin: "androidx.navigation.safeargs.kotlin"
```

```
// Configure using Java 8 - add Module:app/build.gradle under android { ...  
compileOptions {  
    sourceCompatibility JavaVersion.VERSION_1_8  
    targetCompatibility JavaVersion.VERSION_1_8  
}  
kotlinOptions {  
    jvmTarget = "1.8"  
}
```

Add NavHostFragment to main the activity layout

- Add **NavHostFragment** to the main activity layout and associate it with the app nav graph



<fragment

android:id="@+id/**navHostFragment**"

android:name="androidx.navigation.fragment.**NavHostFragment**"

android:layout_width="0dp"

android:layout_height="0dp"

android:layout_marginEnd="1dp"

app:defaultNavHost="true"

app:**navGraph="@navigation/nav_graph"**

... />

Navigate to destinations using NavController

- From any activity or fragment use **findNavController()** to navigate to:
 - a particular action (i.e., a specific path in the navigation graph) or
 - directly to a specific destination
- The requested destination fragment will be loaded in the NavHostFragment

// In fragment:

```
findNavController().navigate(R.id.toSecondFragment)
```

// In main activity:

```
findNavController(R.id.navHostFragment).navigate(R.id.toSecondFragment)
```

Navigate Up

- Call **setupActionBarWithNavController** in the MainActivity onCreate to show the **Navigate Up** button and the **label** of the current fragment on the Action Bar
 - This method is in `androidx.navigation.ui.NavigationUI` package

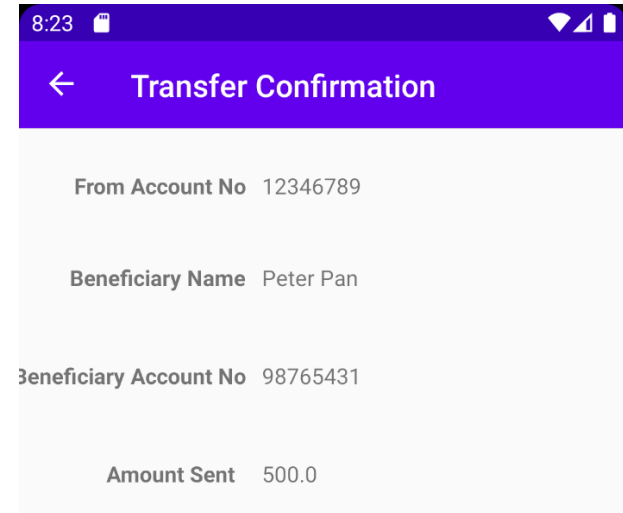
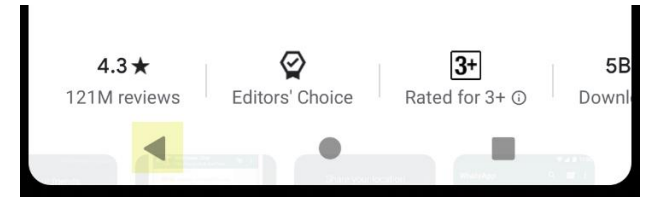
```
navController = findNavController(R.id.navHostFragment)  
setupActionBarWithNavController(this, navController)
```

- Handle *Navigate Up* event

```
override fun onSupportNavigateUp() = navController.navigateUp()
```

Back vs. Up Button

- The Back button is a system button available on all screens to allow users to navigate recently viewed screens in reverse chronological order
- Navigate Up button on the top app bar of child screens allows **upward navigation** one level upwards within the nav graph until the app's home
 - E.g., Navigate Up on Funds Transfer confirmation screen navigates back to the app's home



popUpTo and popUpToInclusive

- When navigating using an action, you can optionally pop off previously visited destinations of the back stack
- For example, after a login flow, you should **pop off all the login-related destinations** of the back stack so that the Back button doesn't take users back into the login flow.
 - Go back to the home fragment while removing all visited destinations from the back stack
 - If popUpToInclusive="true" the destination specified in popUpTo should also be removed from the back stack

→ navigateToHome		action
id	<input type="text" value="navigateToHome"/>	
destination	<input type="text" value="homeFragment"/>	
▶ Animations		
▶ Argument Default Values		
▼ Pop Behavior		
popUpTo	<input type="text" value="homeFragment"/>	
popUpToInclusive	<input checked="" type="checkbox"/> true	
▶ Launch Options		

popUpTo Example

```
<action  
  android:id="@+id/action_c_to_a"  
  app:destination="@id/a"  
  app:popUpTo="@+id/a"  
  app:popUpToInclusive="true"/>
```



- After reaching C, the back stack contains (A, B, C). When navigating back to A, we also **popUpTo A**, which means that we remove B and C from the stack as part of the call to **navigate(action_c_to_a)**
 - With `popUpToInclusive="true"`, we also pop off that first A of the stack to avoid having two instances of A

<https://developer.android.com/guide/navigation/navigation-navigate#pop>

Connect Bottom Nav Bar to NavController

- Add Bottom Nav Bar to the main layout
- Make the id of menu items the same as the id of associated destination in the nav graph
- Connect the bottomNavBar with the NavController to auto-handle OnNavigationItemSelectedListener

```
bottomNavBar.setupWithNavController(NavController)
```

Passing Data between Destinations

- To pass data between destinations, first add the argument to the destination that receives it
 - For example, a user profile destination might take a user ID argument to determine which user to display

The screenshot shows the Android Studio interface. At the top, the 'welcomeFragment' fragment is selected, showing its properties: id (welcomeFragment), label (Welcome), and name (WelcomeFragment (qa.ec)). Below this, the 'Arguments' section is expanded, showing a list of arguments. The 'Add Argument' dialog is open, allowing the user to add a new argument. The dialog has fields for Name (userName), Type (String), Array (unchecked), Nullable (unchecked), and Default Value (empty). The 'Add' button is highlighted in blue.

Property	Value
id	welcomeFragment
label	Welcome
name	WelcomeFragment (qa.ec)

Arguments + -

Add Argument X

Name: userName

Type: String

Array: ☐

Nullable: ☐

Default Value:

Add **Cancel**

Passing Data between Destinations

- Pass data to a destination

```
loginBtn.setOnClickListener {  
    val bundle = bundleOf("userName" to userNameEt.text.toString())  
    findNavController().navigate(R.id.toWelcome, bundle)  
}
```

- Read passed data

```
class WelcomeFragment : Fragment(R.layout.fragment_welcome) {  
    override fun onCreateView(view: View, savedInstanceState: Bundle?) {  
        // Read data passed from the login fragment  
        val userName = arguments?.getString("userName")  
        welcomeTv.text = "Welcome $userName"  
    }  
}
```

Use Safe Args to pass data with type safety

- Safe Args plug-in generates classes for **type-safe navigation** and access to any associated arguments
- **Pass data to a destination**

```
loginBtn.setOnClickListener {  
    val userName = userNameEt.text.toString()  
    val action = LoginFragmentDirections.toWelcome(userName)  
    findNavController().navigate(action)  
}
```

- **Read passed data**

```
private val args: WelcomeFragmentArgs by navArgs()  
  
override fun onCreateView(view: View, savedInstanceState: Bundle?) {  
    // Read data passed from the Login fragment  
  
    val userName = args.userName  
  
    welcomeTv.text = "Welcome $userName"  
}
```

Resources

- Navigation UI
 - <https://developer.android.com/guide/navigation/navigation-ui>
- Get started with the Navigation component
 - <https://developer.android.com/guide/navigation/navigation-getting-started>
- Navigation Component codelab
 - <https://codelabs.developers.google.com/codelabs/kotlin-android-training-add-navigation/>