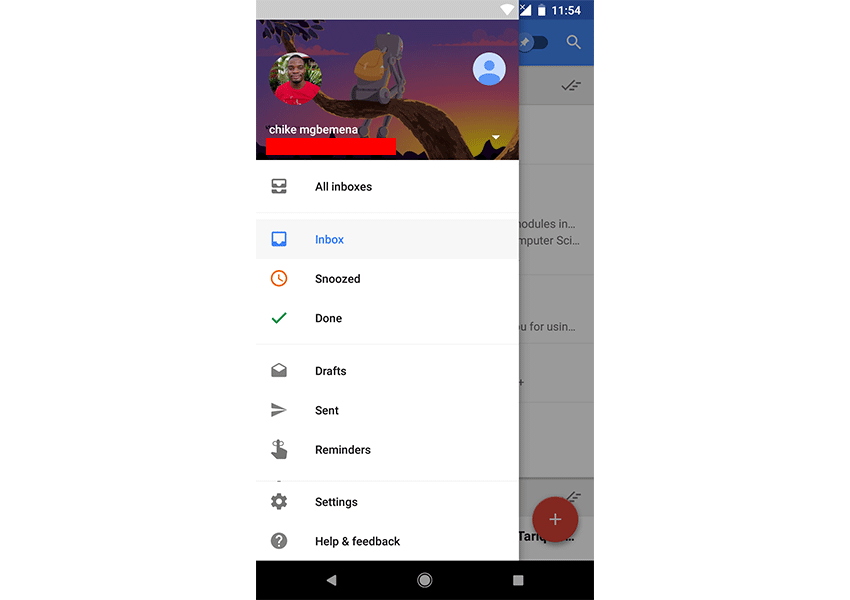
Module 4 Assignment

**How to Code a Navigation Drawer for an Android App**

What You'll Be Creating a Navigation Drawer for an Android App. The navigation drawer slides in from the left and contains the navigation destinations for your app.

An example of a popular Android app that implements the navigation drawer menu design is the Inbox app from Google, which uses a navigation drawer to navigate different application sections. You can check it yourself by downloading the Inbox app from the Google Play store if you don't already have it on your device. The screenshot below shows Inbox with the navigation drawer pulled open.



The user can view the navigation drawer when they swipe a finger from the left edge of the activity. They can also find it from the home activity (the top level of the app) by tapping the app icon (also known as the Android "hamburger" menu) in the action bar.

Note that if you have many different destinations (more than six, say) in your app, it's recommended that you use a navigation drawer menu design.

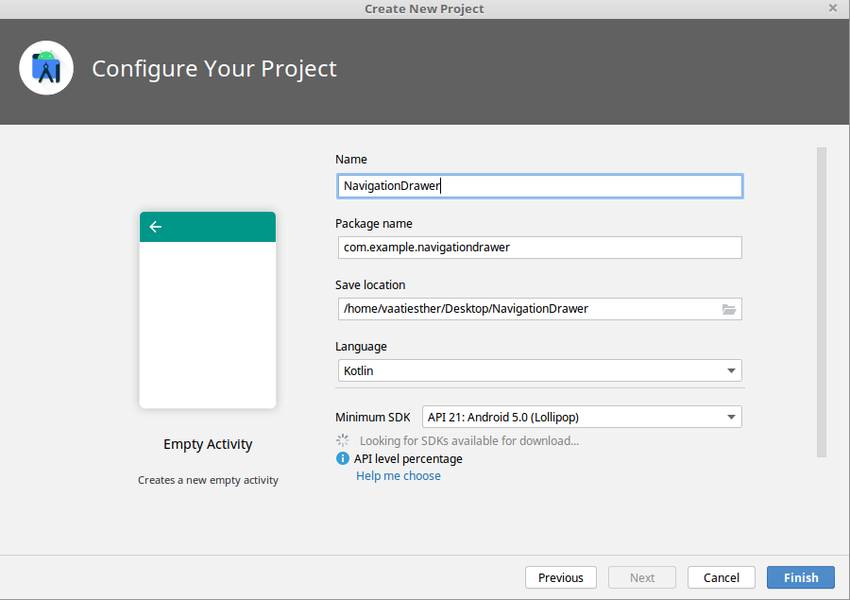
**Prerequisites**

To be able to follow this Android Studio navigation drawer tutorial, you'll need:

* [**Android Studio 3.3 or higher**](https://developer.android.com/studio)
* [**Kotlin plugin**](https://plugins.jetbrains.com/plugin/6954-kotlin) 1.1.51 or higher
* [**Java 8 language features**](https://developer.android.com/studio/write/java8-support)

**Create an Android Studio Project**

Fire up Android Studio and create a new project (you can name it NavigationDrawer) with an empty activity called MainActivity. Make sure to also choose the **Kotlin**language.



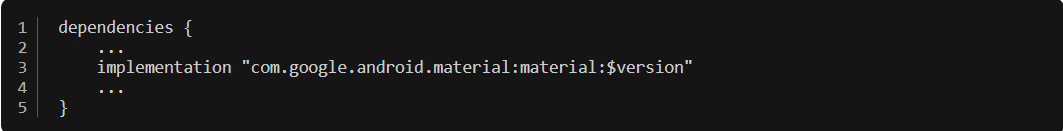
**Add Project Dependencies**

Support for navigation requires some dependencies. Open the app **build.gradle** file and add the following dependencies.

|  |
| --- |
| dependencies { |
| def lifecycle\_version = "2.2.0" |
| implementation "androidx.lifecycle:lifecycle-livedata-ktx:$lifecycle\_version" |
| implementation "androidx.lifecycle:lifecycle-viewmodel-ktx:$lifecycle\_version" |
| ... |
| } |

Also add the material library to the project.

|  |
| --- |
| dependencies { |
| ... |
| implementation "com.google.android.material:material:$version" |
| ... |
| } |

****

Sync the project files for the changes to take effect.

**Create the DrawerLayout**

To display the drawer icon on all destinations in our app, we will use the DrawerLayout component. Open **main\_acivity.xm**l and add DrawerLayout as the root view. The drawer layout will host two child views, NavHostFragment and NavigationView.

|  |
| --- |
| <?xml version="1.0" encoding="utf-8"?> |
| <androidx.drawerlayout.widget.DrawerLayout xmlns:android="https://schemas.android.com/apk/res/android" |
| xmlns:app="http://schemas.android.com/apk/res-auto" |
| xmlns:tools="http://schemas.android.com/tools" |
| android:id="@+id/drawer\_layout" |
| android:layout\_width="match\_parent" |
| android:layout\_height="match\_parent" |
| android:fitsSystemWindows="true" |
| tools:openDrawer="start"> |
|  |
| <!--TOOLBAR HERE--> |
|  |
|  |
| <!--NavHostFragment HERE--> |
|  |
| <!--NavigationView HERE--> |
|  |
| </androidx.drawerlayout.widget.DrawerLayout> |

Here we created a DrawerLayout widget with the id drawer\_layout. The tools:openDrawer property is used to display the navigation drawer toggle when the XML layout is open in Android Studio design view.

The [**official documentation**](https://developer.android.com/reference/android/support/v4/widget/DrawerLayout.html) says the following about DrawerLayout:

DrawerLayout acts as a top-level container for window content that allows for interactive "drawer" views to be pulled out from one or both vertical edges of the window.

After adding the DrawerLayout widget, we included a child layout, **app\_bar\_main.xml** which points to the toolbar layout.

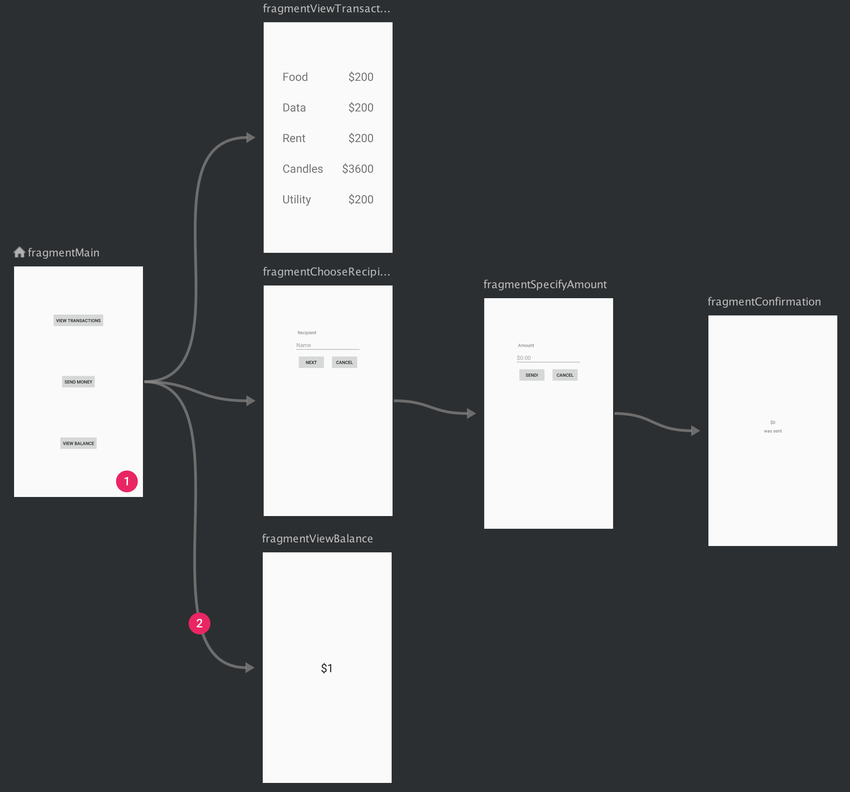
|  |
| --- |
| <!--main\_activity.xml--> |
| <?xml version="1.0" encoding="utf-8"?> |
| <androidx.drawerlayout.widget.DrawerLayout xmlns:android="http://schemas.android.com/apk/res/android" |
| xmlns:app="http://schemas.android.com/apk/res-auto" |
| xmlns:tools="http://schemas.android.com/tools" |
| android:id="@+id/drawer\_layout" |
| android:layout\_width="match\_parent" |
| android:layout\_height="match\_parent" |
| android:fitsSystemWindows="true" |
| tools:openDrawer="start"> |
|  |
| <include |
| layout="@layout/app\_bar\_main" |
| android:layout\_width="match\_parent" |
| android:layout\_height="match\_parent" /> |
|  |
|  |
| <!--NavHostFragment HERE--> |
|  |
| <!--NavigationView HERE--> |
|  |
| </androidx.drawerlayout.widget.DrawerLayout> |
|  |

Here is my **app\_bar\_main.xml** resource file. This file has a CoordinatorLayout, an AppBarLayout, and a Toolbar widget.

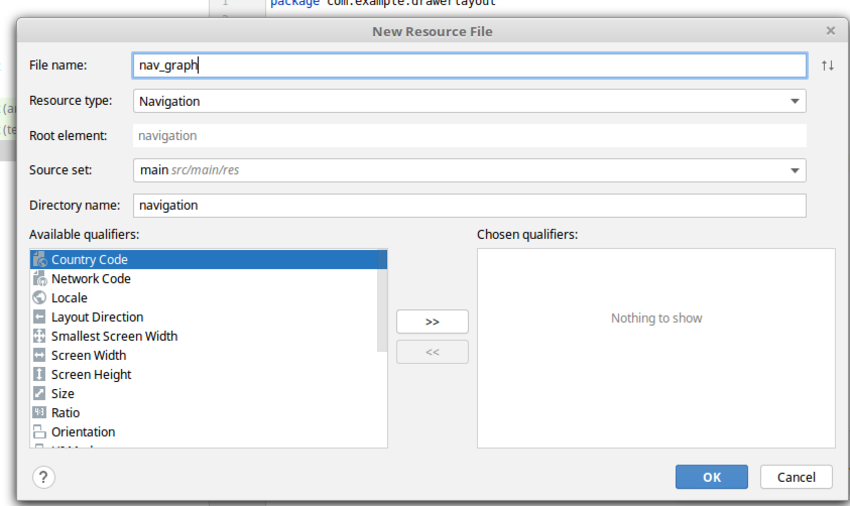
|  |
| --- |
| <!--tool\_bar\_layout.xml--> |
| <?xml version="1.0" encoding="utf-8"?> |
| <androidx.coordinatorlayout.widget.CoordinatorLayout xmlns:android="http://schemas.android.com/apk/res/android" |
| xmlns:app="http://schemas.android.com/apk/res-auto" |
| xmlns:tools="http://schemas.android.com/tools" |
| android:layout\_width="match\_parent" |
| android:layout\_height="match\_parent" |
| tools:context=".MainActivity"> |
|  |
| <com.google.android.material.appbar.AppBarLayout |
| android:layout\_width="match\_parent" |
| android:layout\_height="wrap\_content" |
| android:theme="@style/Theme.NavigationDrawer.AppBarOverlay"> |
|  |
| <androidx.appcompat.widget.Toolbar |
| android:id="@+id/toolbar" |
| android:layout\_width="match\_parent" |
| android:layout\_height="?attr/actionBarSize" |
| android:background="?attr/colorPrimary" |
| app:popupTheme="@style/Theme.NavigationDrawer.PopupOverlay" /> |
|  |
| </com.google.android.material.appbar.AppBarLayout> |
|  |
| </androidx.coordinatorlayout.widget.CoordinatorLayout> |

**Create a Navigation Graph**

A *navigation graph* is an XML resource file that contains all of your app's destinations and actions, and these destinations are connected via actions. Below is an example of a navigation graph showing five fragments.



To add a navigation graph, **right-click** on the res directory and select **New > Android Resource File**. In the next dialog, select **Navigation** as the **Resource Type**, and click **OK**. A new XML file, **nav\_graph.xml**, will be created in the **Navigation** folder, as shown below.



**Add NavHostFragment**

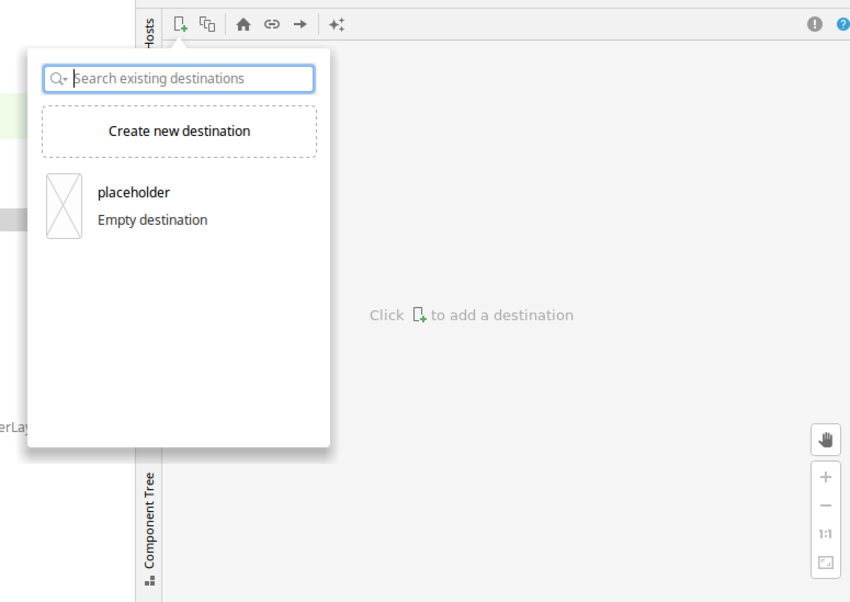
A navigation host fragment acts as a host for the app's fragments and swaps fragments in and out as necessary when the user moves from one destination to the other. These destinations have to be defined in the navigation graph.

Add NavHostFragment to the **main\_activity.xml** file and reference the navGraph.

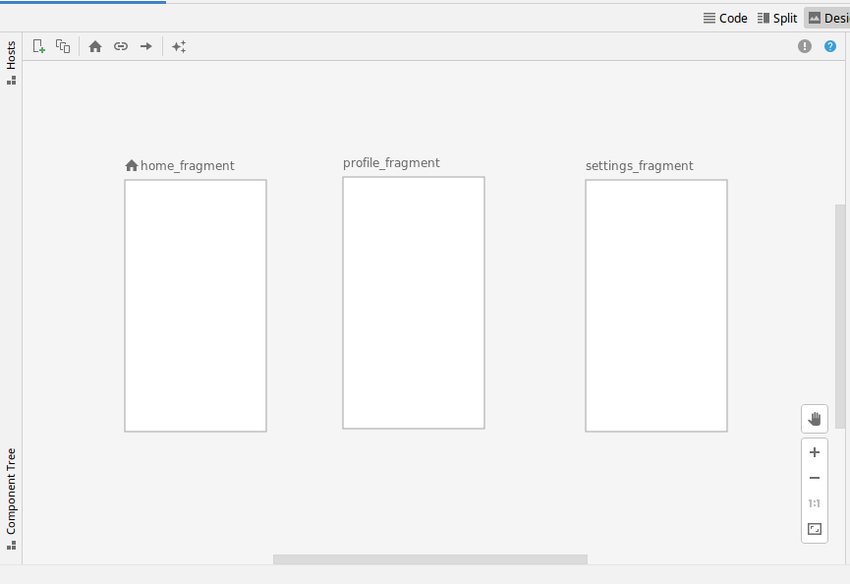
|  |
| --- |
| <!--main\_activity.xml--> |
| <?xml version="1.0" encoding="utf-8"?> |
| <androidx.drawerlayout.widget.DrawerLayout xmlns:android="http://schemas.android.com/apk/res/android" |
| xmlns:app="http://schemas.android.com/apk/res-auto" |
| xmlns:tools="http://schemas.android.com/tools" |
| android:id="@+id/drawer\_layout" |
| android:layout\_width="match\_parent" |
| android:layout\_height="match\_parent" |
| android:fitsSystemWindows="true" |
| tools:openDrawer="start"> |
|  |
| <include |
| layout="@layout/app\_bar\_main" |
| android:layout\_width="match\_parent" |
| android:layout\_height="match\_parent" /> |
|  |
|  |
| <androidx.constraintlayout.widget.ConstraintLayout |
| android:layout\_width="match\_parent" |
| android:layout\_height="match\_parent"> |
|  |
| <fragment |
| android:id="@+id/nav\_host\_fragment" |
| android:name="androidx.navigation.fragment.NavHostFragment" |
| android:layout\_width="match\_parent" |
| android:layout\_height="match\_parent" |
| app:defaultNavHost="true" |
| app:layout\_constraintLeft\_toLeftOf="parent" |
| app:layout\_constraintRight\_toRightOf="parent" |
| app:layout\_constraintTop\_toTopOf="parent" |
| app:navGraph="@navigation/nav\_graph" /> |
| </androidx.constraintlayout.widget.ConstraintLayout> |
|  |
| <!--NavigationView HERE--> |
|  |
| </androidx.drawerlayout.widget.DrawerLayout> |

**Add Fragments to the Destination Graph**

Fragments represent all the destinations of your app. In our case, we will add three fragments to the navigation graph. **Right-click** the navigation folder and open **nav\_graph.xml.** To add a fragment, click on **Create New Destination** and fill out the rest of the details.



Repeat the same steps and create two additional fragments, the profile fragment and the settings fragment. Your navigation graph should now look like this.



**Add a NavigationView Component**

Finally, let's create  a NavigationView widget. The [**official documentation**](https://developer.android.com/reference/android/support/design/widget/NavigationView.html) says the following about NavigationView:

NavigationView represents a standard navigation menu for application. The menu contents can be populated by a menu resource file.

Open **main\_activity.xml** and add the NavigationView.

|  |
| --- |
| <!--main\_activity.xml--> |
| <?xml version="1.0" encoding="utf-8"?> |
| <androidx.drawerlayout.widget.DrawerLayout xmlns:android="http://schemas.android.com/apk/res/android" |
| xmlns:app="http://schemas.android.com/apk/res-auto" |
| xmlns:tools="http://schemas.android.com/tools" |
| android:id="@+id/drawer\_layout" |
| android:layout\_width="match\_parent" |
| android:layout\_height="match\_parent" |
| android:fitsSystemWindows="true" |
| tools:openDrawer="start"> |
|  |
| <include |
| layout="@layout/app\_bar\_main" |
| android:layout\_width="match\_parent" |
| android:layout\_height="match\_parent" /> |
|  |
|  |
| <androidx.constraintlayout.widget.ConstraintLayout |
| android:layout\_width="match\_parent" |
| android:layout\_height="match\_parent"> |
|  |
| <fragment |
| android:id="@+id/nav\_host\_fragment" |
| android:name="androidx.navigation.fragment.NavHostFragment" |
| android:layout\_width="match\_parent" |
| android:layout\_height="match\_parent" |
| app:defaultNavHost="true" |
| app:layout\_constraintLeft\_toLeftOf="parent" |
| app:layout\_constraintRight\_toRightOf="parent" |
| app:layout\_constraintTop\_toTopOf="parent" |
| app:navGraph="@navigation/nav\_graph" /> |
| </androidx.constraintlayout.widget.ConstraintLayout> |
|  |
| <com.google.android.material.navigation.NavigationView |
| android:id="@+id/nav\_view" |
| android:layout\_width="wrap\_content" |
| android:layout\_height="match\_parent" |
| android:layout\_gravity="start" |
| android:fitsSystemWindows="true" |
| app:headerLayout="@layout/nav\_header\_main" |
| app:menu="@menu/drawer\_menu" /> |
|  |
| </androidx.drawerlayout.widget.DrawerLayout> |
|  |

In the NavigationView XML widget, you can see that we added an android:layout\_gravity attribute with the value start. This is used to position the drawer—you want the navigation drawer menu design to come out from the left or right (the start or end on platform versions that support layout direction). In our own case, the drawer will come out from the left.

We also included an app:headerLayout attribute, which points to @layout/nav\_header\_main. This will add a View as a header of the navigation menu.

Here is my **nav\_header\_main.xml** layout resource file:

|  |
| --- |
| <?xml version="1.0" encoding="utf-8"?> |
| <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android" |
| xmlns:app="http://schemas.android.com/apk/res-auto" |
| android:layout\_width="match\_parent" |
| android:layout\_height="@dimen/nav\_header\_height" |
| android:gravity="bottom" |
| android:orientation="vertical" |
| android:theme="@style/ThemeOverlay.AppCompat.Dark"> |
|  |
| <TextView |
| android:layout\_width="match\_parent" |
| android:layout\_height="wrap\_content" |
| android:paddingTop="@dimen/nav\_header\_vertical\_spacing" |
| android:text="@string/nav\_header\_title" |
| android:textAppearance="@style/TextAppearance.AppCompat.Body1" /> |
|  |
| </LinearLayout> |

To include the menu items for the navigation drawer, we can use the attribute app:menu with a value that points to a menu resource file.

app:menu="@menu/drawer\_menu" />

Here is the **res/menu/drawer\_menu.xml** menu resource file:

|  |
| --- |
| <?xml version="1.0" encoding="utf-8"?> |
| <menu xmlns:android="http://schemas.android.com/apk/res/android" |
| xmlns:tools="http://schemas.android.com/tools" |
| tools:showIn="navigation\_view"> |
|  |
| <group android:checkableBehavior="single"> |
| <item |
| android:id="@+id/nav\_home" |
| android:icon="@drawable/home" |
| android:title="@string/menu\_home" /> |
| <item |
| android:id="@+id/nav\_gallery" |
| android:icon="@drawable/person" |
| android:title="@string/menu\_gallery" /> |
| <item |
| android:id="@+id/nav\_slideshow" |
| android:icon="@drawable/settings" |
| android:title="@string/menu\_slideshow" /> |
| </group> |
| </menu> |

Here we have defined a Menu using the <menu> which serves as a container for menu items. An <item> creates a MenuItem, which represents a single item in a menu. It's also important to note that the ids of the menu items correspond to the ids of the matching fragment.

Note that when showing the navigation list items from a menu resource, we could use a ListView instead. But, by configuring the navigation drawer with a menu resource, we get the material design styling on the navigation drawer for free! If you used a ListView, you'd have to maintain the list and also style it to meet the recommended [**material design specs for the navigation drawer**](https://material.io/components/navigation-drawer).

**Initialization of Components**

Next, we are going to initialize instances of all our components. Initialization is going to happen inside onCreate() in **MainActivity.kt**.

The AppBarConfiguration object is used to manage the behavior of the navigation drawer button.

private lateinit var appBarConfiguration: AppBarConfiguration

First, we use the setSupportActionBar() method to set the toolbar as the app bar for the activity.

|  |
| --- |
| val toolbar: Toolbar = findViewById(R.id.toolbar) |
| setSupportActionBar(toolbar) |

Next, we set all fragments as top-level destinations, this means that they will remain in the back stack when navigating.

|  |
| --- |
| // Passing each menu ID as a set of Ids because each |
| // menu should be considered as top level destinations. |
| appBarConfiguration = AppBarConfiguration(setOf( |
| R.id.home\_menu, R.id.profile\_menu, R.id.settings\_menu), drawerLayout) |

The method setupActionBarWithNavController automatically updates the title in the action bar when the destination changes.

|  |
| --- |
| setupActionBarWithNavController(navController, appBarConfiguration) |

Set up the navigation drawer.

|  |
| --- |
| val navView: NavigationView = findViewById(R.id.nav\_view) |
| 2 | val navController = findNavController(R.id.nav\_host\_fragment) |
| 3 | navView.setupWithNavController(navController) |

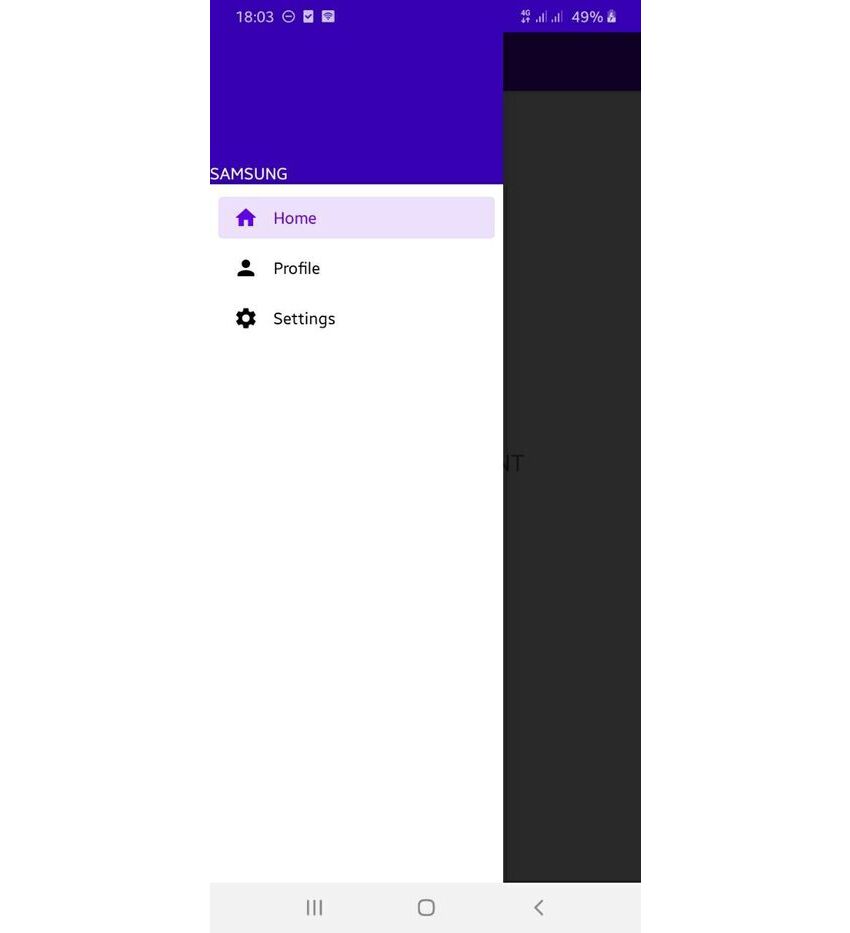
Lastly, show the up button that appears at the top left of the app bar. This is done by integrating the navigation controller withe app bar using the onSupportNavigateUp method.

The final code for **MainActivity.kt** should look like this.

|  |  |
| --- | --- |
| 1 | package com.example.navigationdrawer |
| 2 |  |
| 3 | // imports |
| 4 |  |
| 5 | class MainActivity : AppCompatActivity() { |
| 6 |  |
| 7 | private lateinit var appBarConfiguration: AppBarConfiguration |
| 8 |  |
| 9 | override fun onCreate(savedInstanceState: Bundle?) { |
| 10 | super.onCreate(savedInstanceState) |
| 11 | setContentView(R.layout.activity\_main) |
| 12 | val toolbar: Toolbar = findViewById(R.id.toolbar) |
| 13 | setSupportActionBar(toolbar) |
| 14 | /// |
| 15 | val drawerLayout: DrawerLayout = findViewById(R.id.drawer\_layout) |
| 16 | val navView: NavigationView = findViewById(R.id.nav\_view) |
| 17 | val navController = findNavController(R.id.nav\_host\_fragment) |
| 18 | // Passing each menu ID as a set of Ids because each |
| 19 | // menu should be considered as top level destinations. |
| 20 | appBarConfiguration = AppBarConfiguration(setOf( |
| 21 | R.id.nav\_home, R.id.nav\_gallery, R.id.nav\_slideshow), drawerLayout) |
| 22 | setupActionBarWithNavController(navController, appBarConfiguration) |
| 23 | navView.setupWithNavController(navController) |
| 24 | } |
| 25 |  |
| 26 |  |
| 27 | override fun onSupportNavigateUp(): Boolean { |
| 28 | val navController = findNavController(R.id.nav\_host\_fragment) |
| 29 | return navController.navigateUp(appBarConfiguration) || super.onSupportNavigateUp() |
| 30 | } |
| 31 | } |
| 32 |  |

**Testing the App**

At this point, we can run the app!



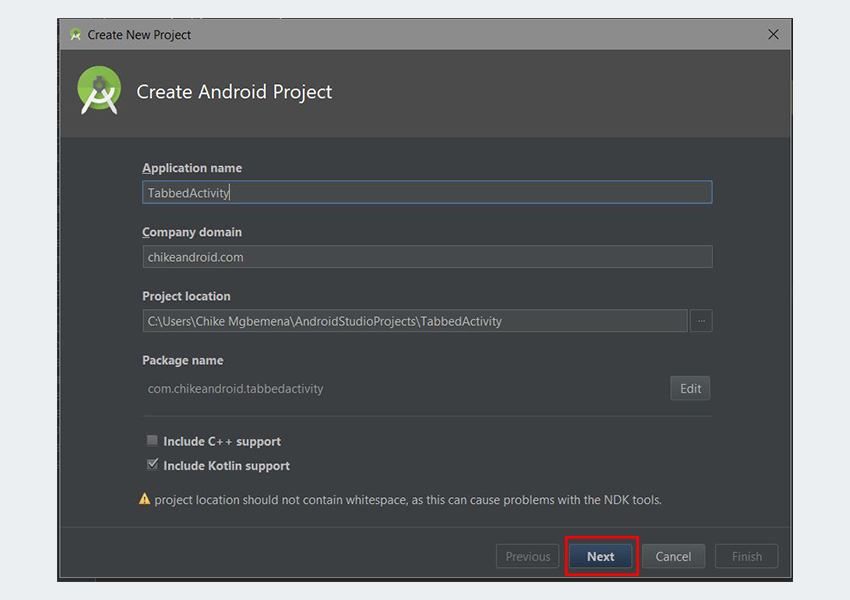
## Bonus: Using Android Studio Templates

Now that you've learnt about the APIs involved to create a navigation drawer, I'll show you a shortcut that will make it faster next time. You can simply use a template instead of coding a navigation drawer Activity from scratch.

Android Studio provides code templates that follow the Android design and development best practices. These existing code templates (available in Java and Kotlin) can help you quickly kick-start your project. One such template can be used to create a navigation drawer activity.

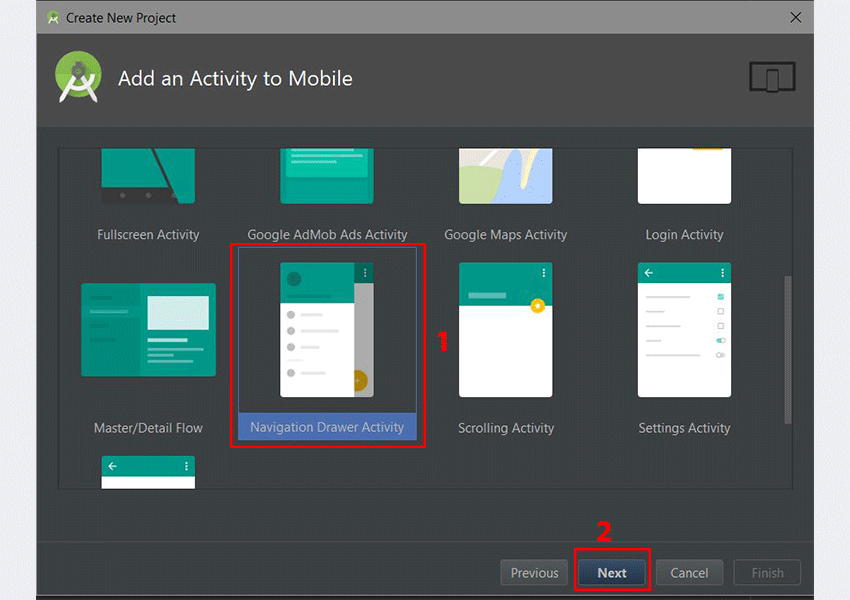
I'll show you how to use this handy feature in Android Studio.

For a new project, fire up Android Studio.

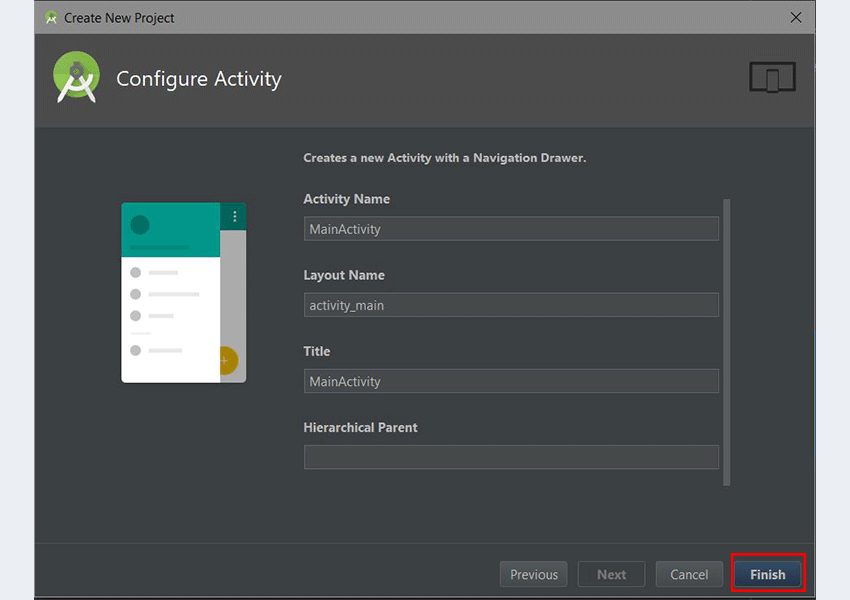


Enter the application name and click the **Next** button.

You can leave the defaults as they're in the **Target Android Devices** dialog. Click the **Next** button again.



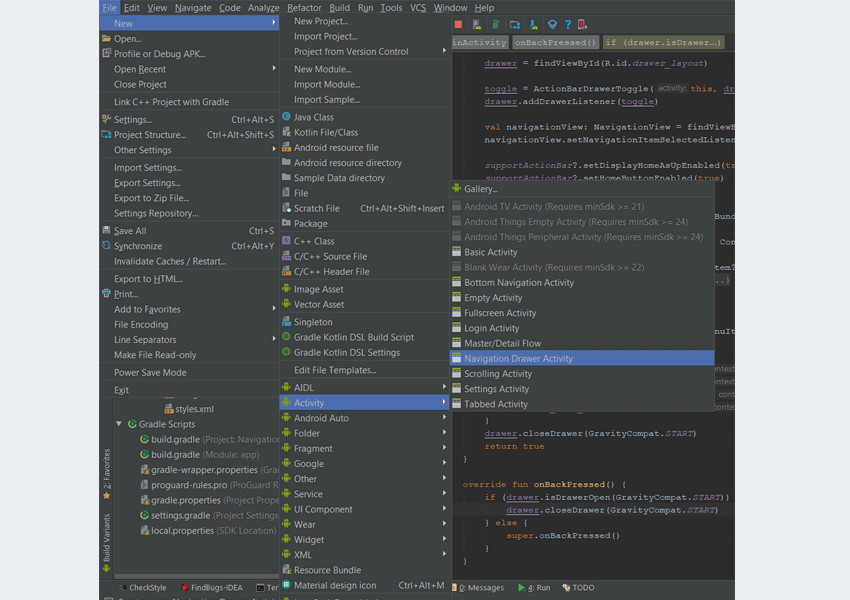
In the **Add an Activity to Mobile** dialog, scroll down and select **Navigation Drawer Activity.**Click the **Next** button after that.



In the last dialog, you can rename the Activity name, layout name, or title if you want. Finally, click the **Finish** button to accept all configurations.

Android Studio has now helped us to create a project with a navigation drawer activity. Really cool! You're strongly advised to explore the code generated.

You can use templates for an already existing Android Studio project too. Simply go to **File > New > Activity > Navigation Drawer Activity**.



**Submission requirement**: You are to provide a complete screenshot of your step-by-step application in a word document and a zip file of your code. Push your project files to a new github.com repo and submit the GitHub link. Create a 3 to 5-minute LOOM video ([*https://www.loom.com/)*](https://www.loom.com/)) in which you explain the execution of your programs. Be sure to address any challenges encountered and new information learned while completing this assignment. Be sure to test your results carefully.