

twitch.tv/dancojocar

youtube.com/dancojocar

Lecture #3

Navigation and Rest Resources

Mobile Applications
Fall 2023

REST using Retrofit

Retrofit

A type-safe HTTP client for Android and Java

Introduction

Retrofit turns your HTTP API into a Java interface.

```
public interface GitHubService {  
    @GET("users/{user}/repos")  
    Call<List<Repo>> listRepos(@Path("user") String user);  
}
```

The `Retrofit` class generates an implementation of the `GitHubService` interface.

```
Retrofit retrofit = new Retrofit.Builder()  
    .baseUrl("https://api.github.com/")  
    .build();  
  
GitHubService service = retrofit.create(GitHubService.class);
```

Each `Call` from the created `GitHubService` can make a synchronous or asynchronous HTTP request to the remote webserver.

```
Call<List<Repo>> repos = service.listRepos("octocat");
```

Use annotations to describe the HTTP request:

- URL parameter replacement and query parameter support
- Object conversion to request body (e.g., JSON, protocol buffers)
- Multipart request body and file upload

REST using Retrofit

```
implementation "com.squareup.retrofit2:retrofit:2.9.0"  
implementation "com.squareup.retrofit2:adapter-rxjava2:2.9.0"  
implementation "com.squareup.retrofit2:converter-gson:2.9.0"  
  
implementation "io.reactivex.rxjava2:rxandroid:2.0.1"
```

REST using Retrofit

DEMO

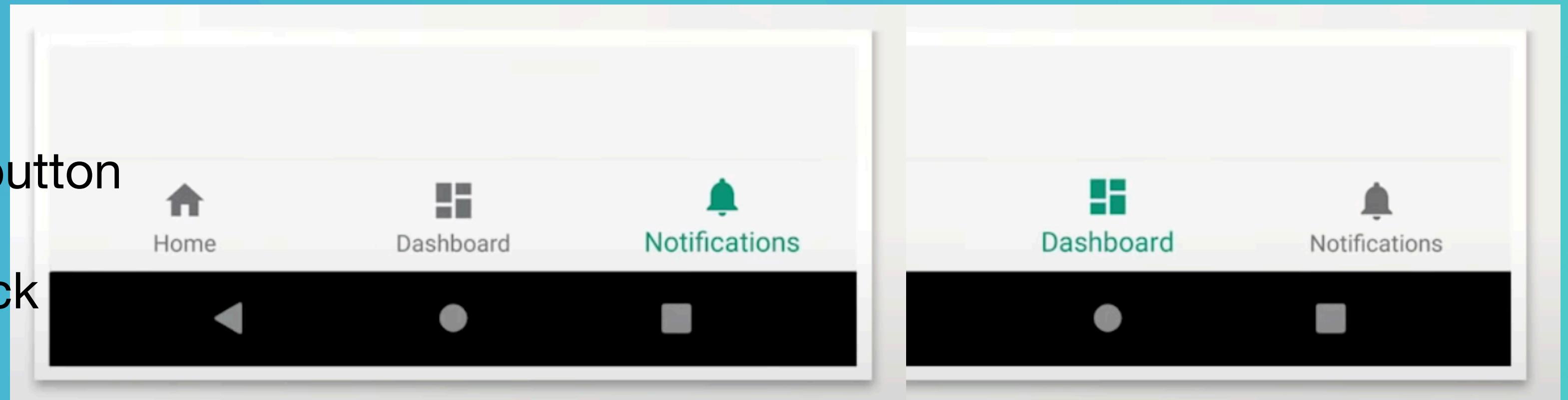
```
interface MovieService {  
  
    @GET("movies")  
    val movies: Observable<List<Movie>>  
  
    @GET("genres")  
    val genres: Observable<List<String>>  
  
    @GET("moviesByGenre/{genre}")  
    fun moviesByGenre(@Path("genre") genre: String)  
        : Observable<List<Movie>>  
  
    @GET("details/{id}")  
    fun details(@Path("id") id: Int): Observable<Movie>  
  
    @POST("updateDescription")  
    fun updateDescription(@Body movie: Movie): Observable<Movie>  
  
    @POST("updateRating")  
    fun updateRating(@Body movie: Movie): Observable<Movie>  
  
    @POST("update")  
    fun update(@Body movie: Movie): Observable<Movie>  
  
    @DELETE("delete/{id}")  
    fun delete(@Path("id") id: Int): Observable<ResponseBody>  
  
    @POST("add")  
    fun add(@Body movie: Movie): Observable<Movie>  
  
    companion object {  
        const val SERVICE_ENDPOINT = "http://SERVER_IP:2022"  
    }  
}
```

Navigation



Navigation

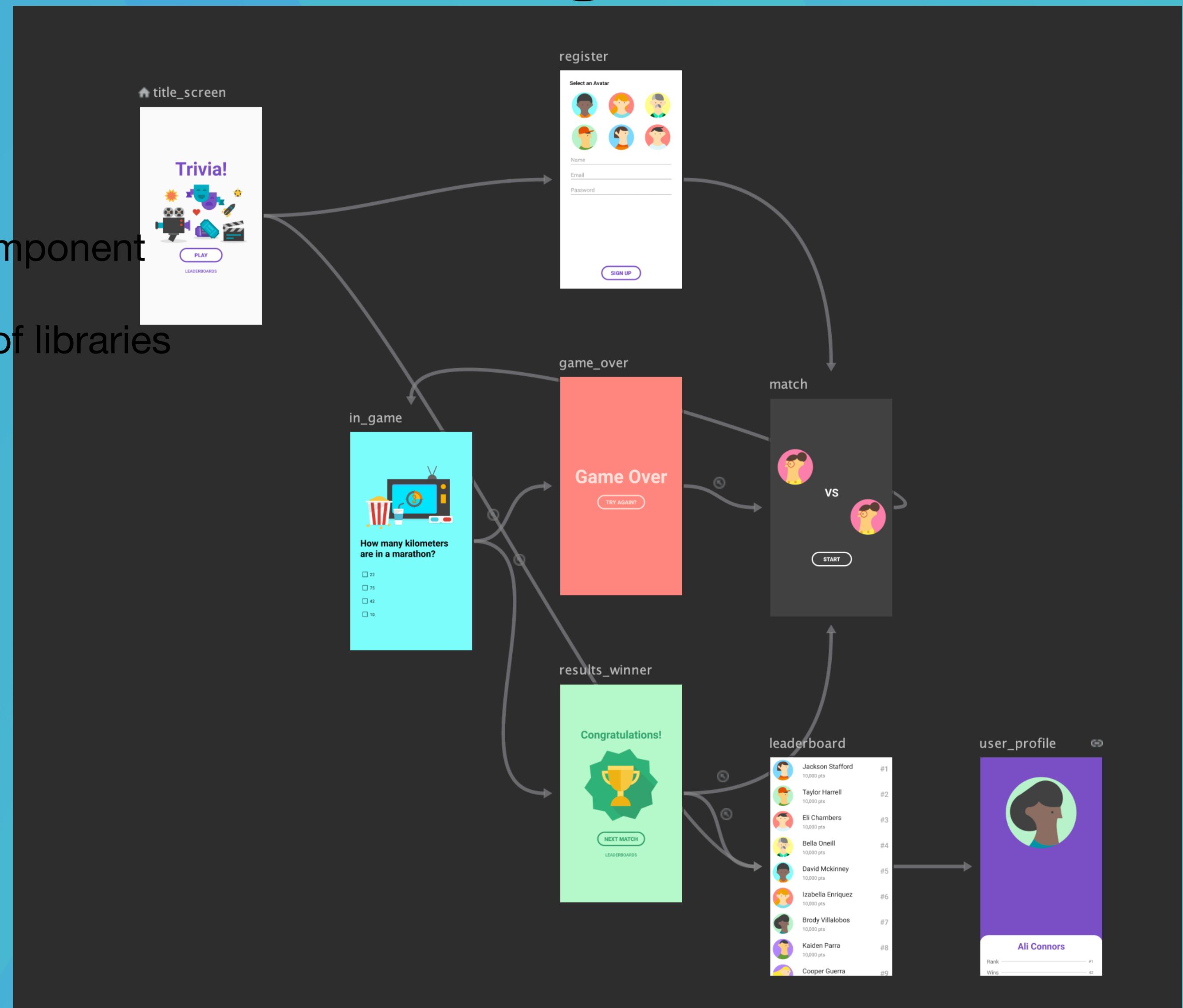
- Correctly
- Highlight the correct button
- Handles the back-stack



Navigation

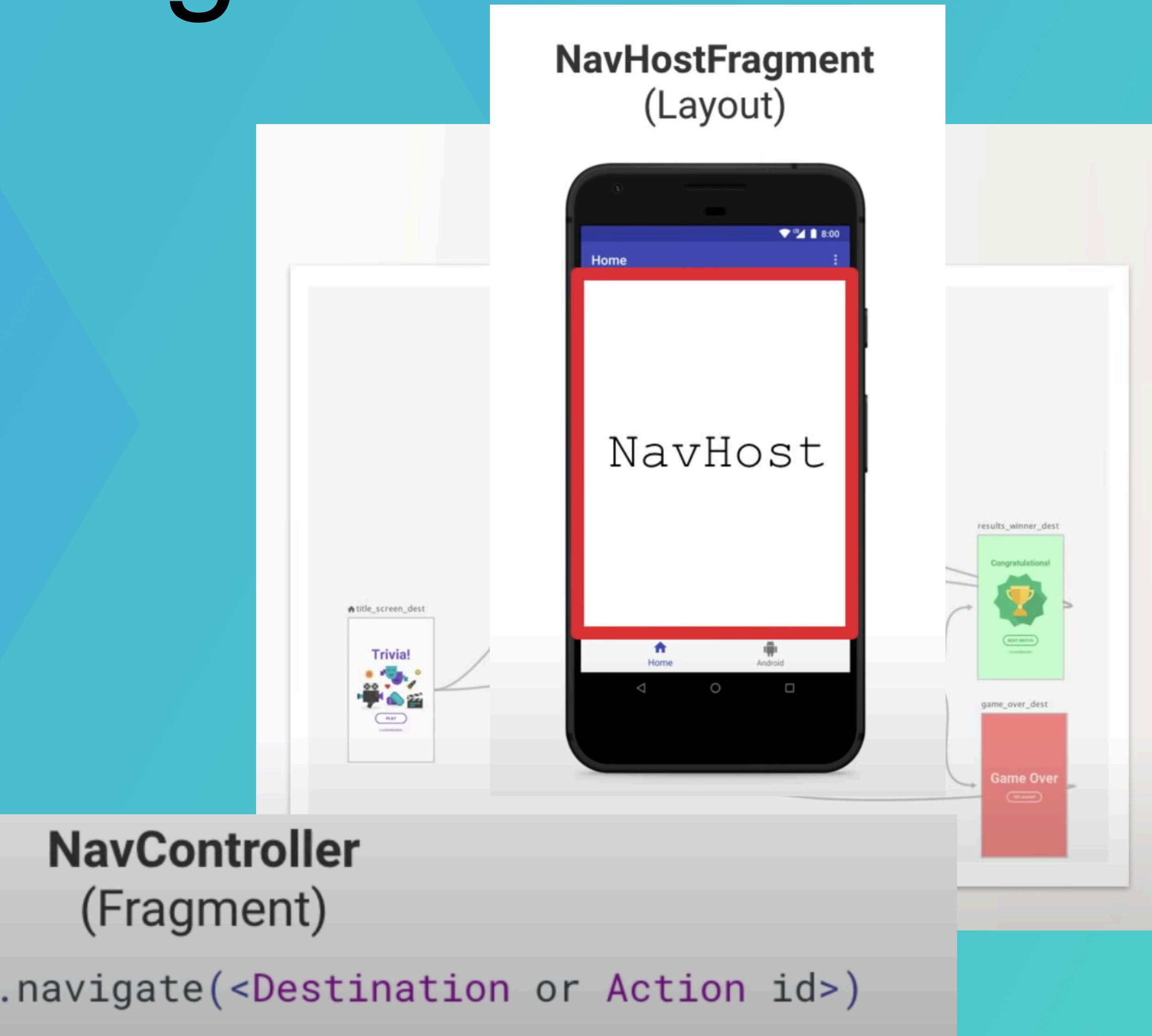
Navigation Component

- A collection of libraries
- A plugin
- Tooling



Navigation

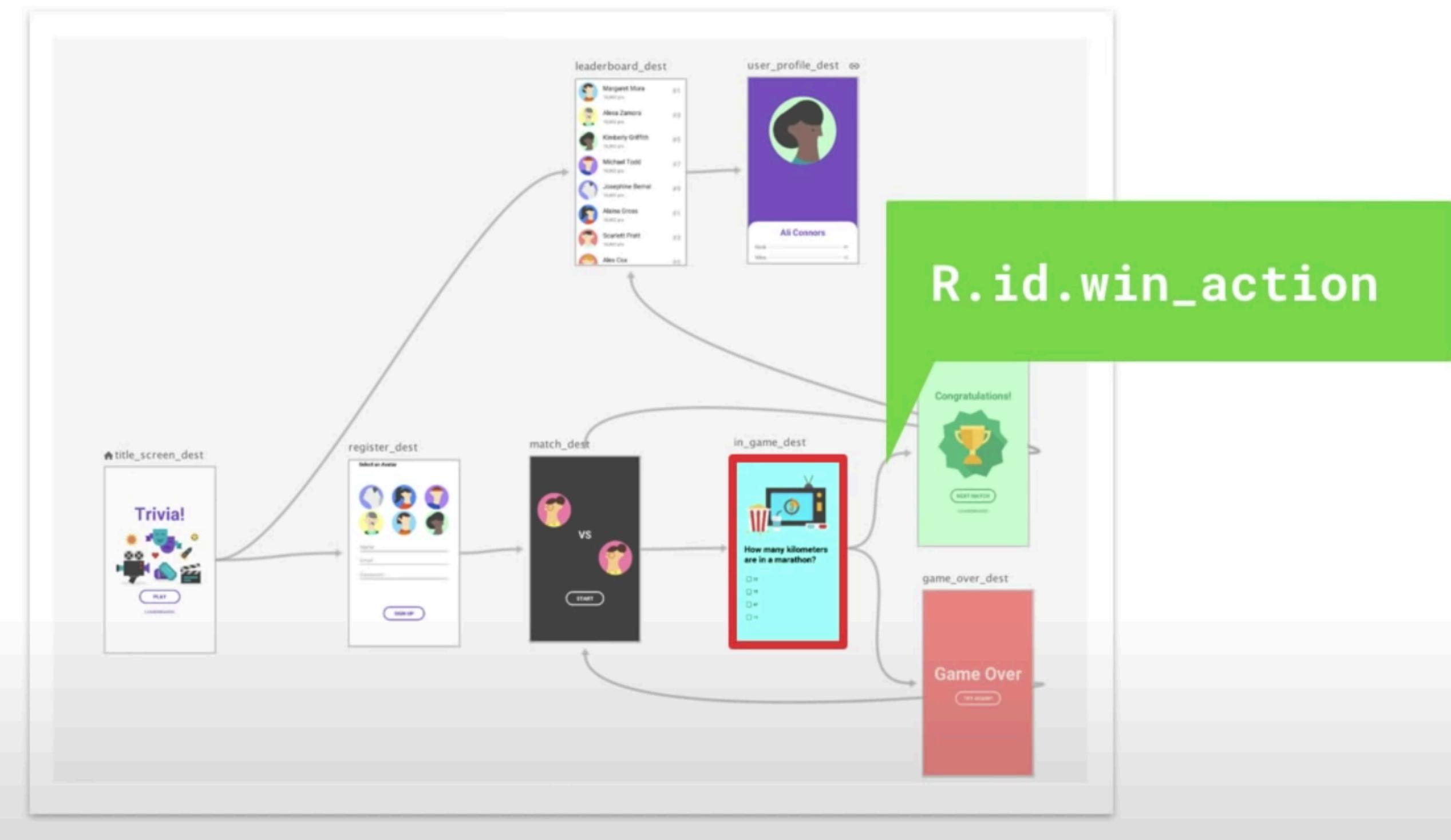
- Navigation Graph
- NavHostFragment
- NavController



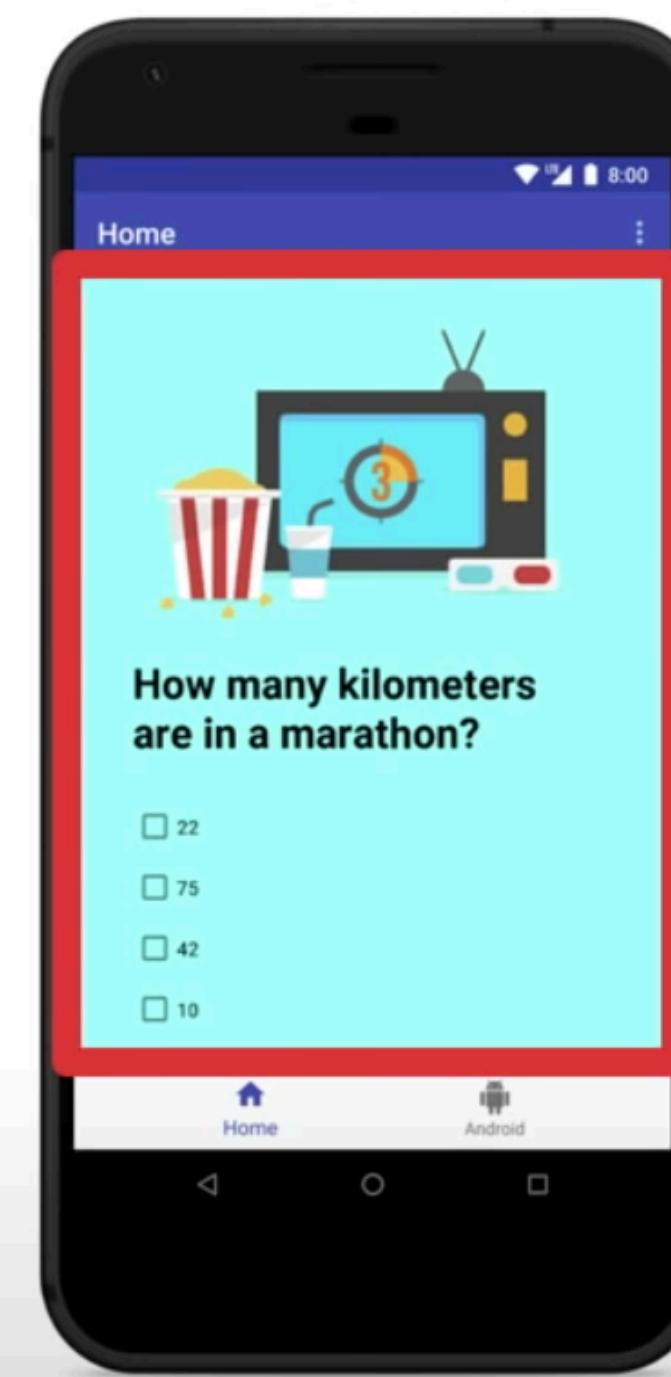
Navigation

DEMO

Navigation Graph
(New Resource)



NavHostFragment
(Layout)



NavController
(Fragment)

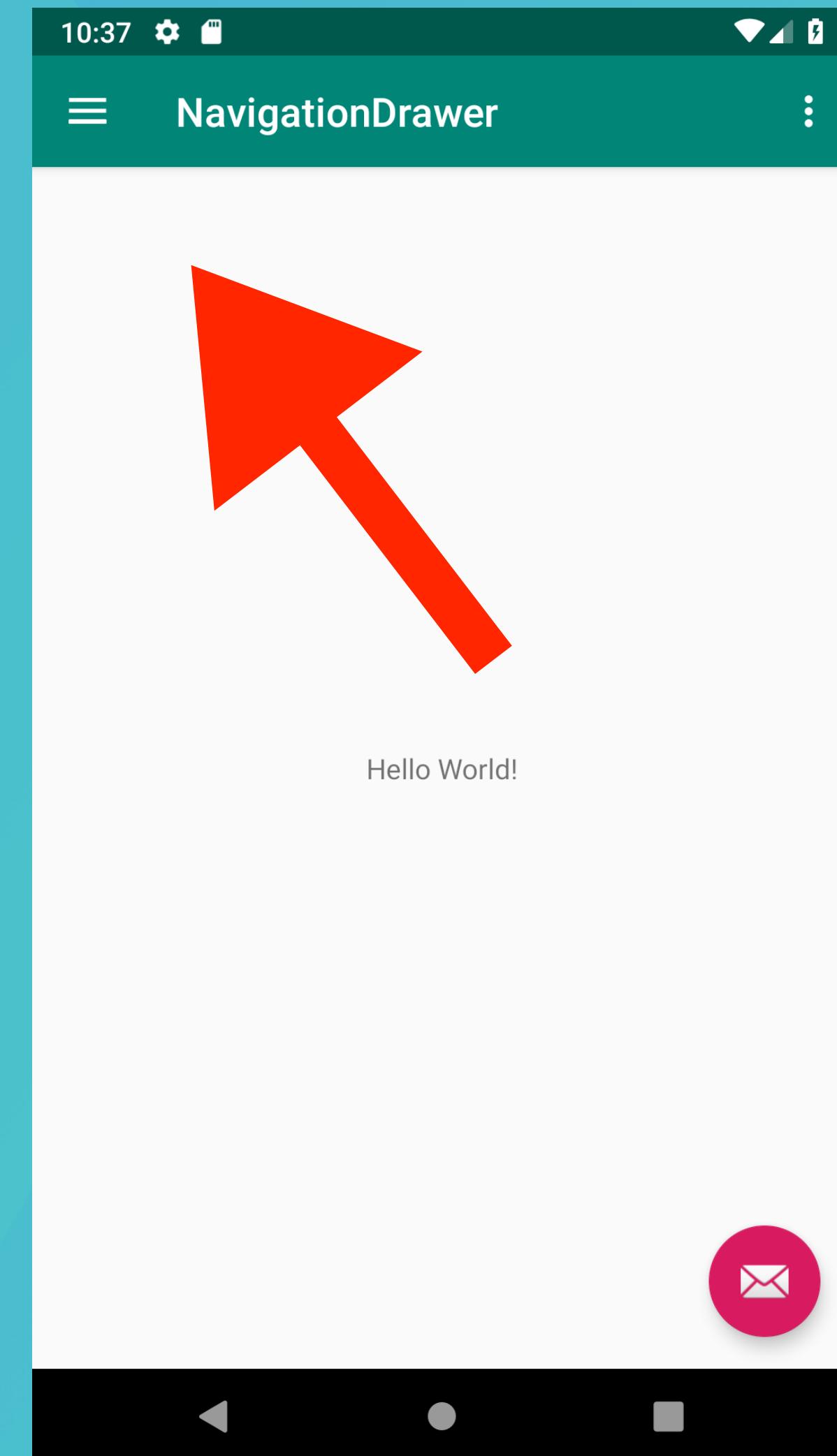
`findNavController().navigate(R.id.win_action)`

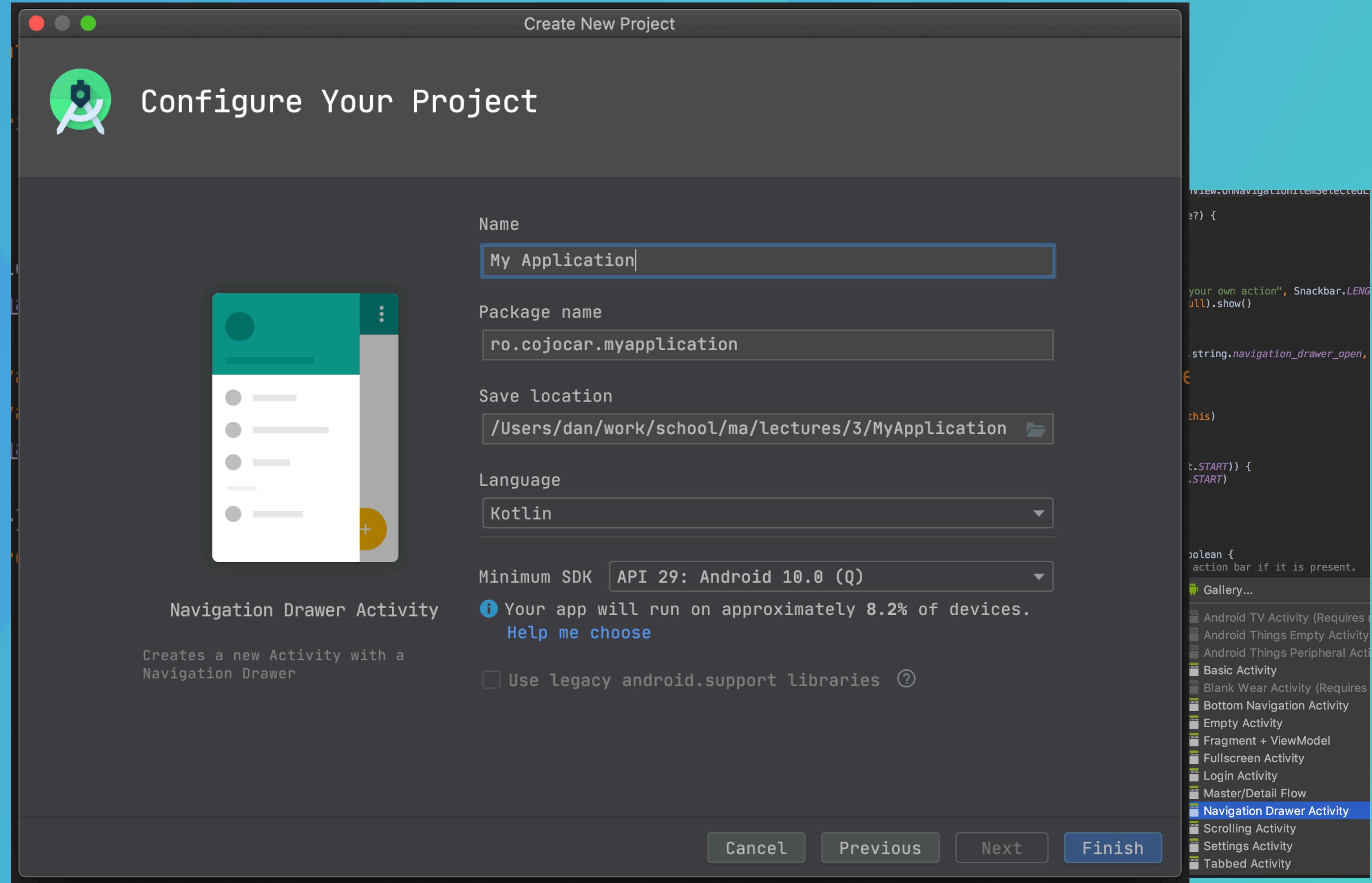
<https://developer.android.com/guide/navigation>

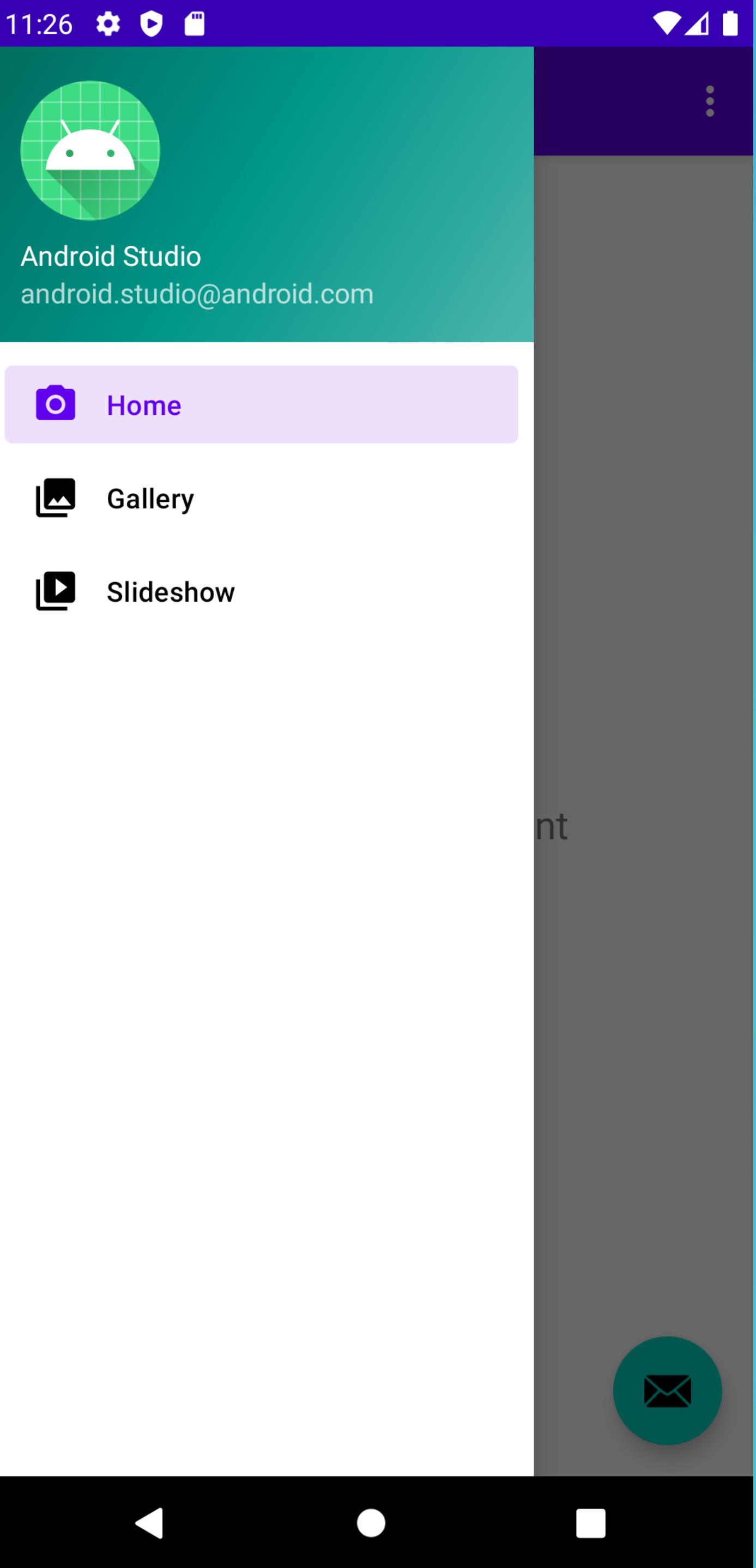


Navigation Drawer

- App main navigation menu.
- Hidden when not in use.
- Appears:
 - with a left swipe from the screen edge
 - when the user touches the drawer icon in the app bar

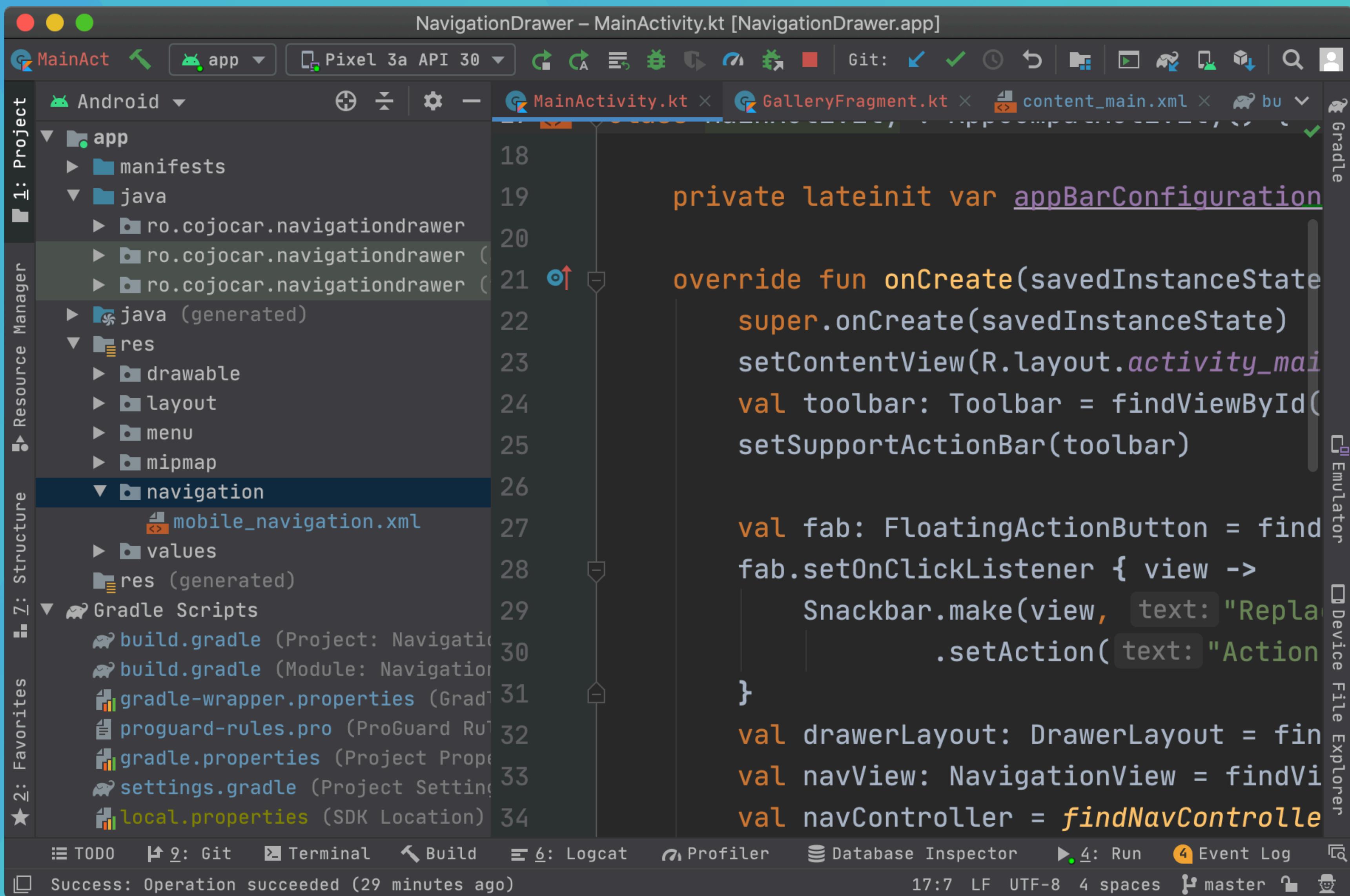






Generated Artifacts

- Sources
- Layouts
- Menus
- Navigation



The screenshot shows the Android Studio interface with the following details:

- Project Bar:** Shows "MainActivity.kt [NavigationDrawer.app]" as the active file.
- Toolbar:** Includes icons for MainAct, app, Pixel 3a API 30, and various developer tools.
- Project Structure:** The "app" module is selected. The "navigation" folder under "res" is highlighted in blue.
- MainActivity.kt:** The code is displayed in the main editor:

```
private lateinit var appBarConfiguration  
override fun onCreate(savedInstanceState)  
    super.onCreate(savedInstanceState)  
    setContentView(R.layout.activity_main)  
    val toolbar: Toolbar = findViewById(R.id.toolbar)  
    setSupportActionBar(toolbar)  
  
    val fab: FloatingActionButton = findViewById(R.id.fab)  
    fab.setOnClickListener { view ->  
        Snackbar.make(view, "Replace with your own action",  
            Snackbar.LENGTH_LONG).setAction("Action", null)  
    }  
  
    val drawerLayout: DrawerLayout = findViewById(R.id.drawer_layout)  
    val navView: NavigationView = findViewById(R.id.nav_view)  
    val navController = findNavController(R.id.nav_host_fragment)
```

- Bottom Navigation:** Includes tabs for TODO, Git, Terminal, Build, Logcat, Profiler, Database Inspector, Run, Event Log, and a success message: "Success: Operation succeeded (29 minutes ago)".
- Status Bar:** Shows the time as 17:7, LF, UTF-8, 4 spaces, master branch, and a user icon.

Dependencies

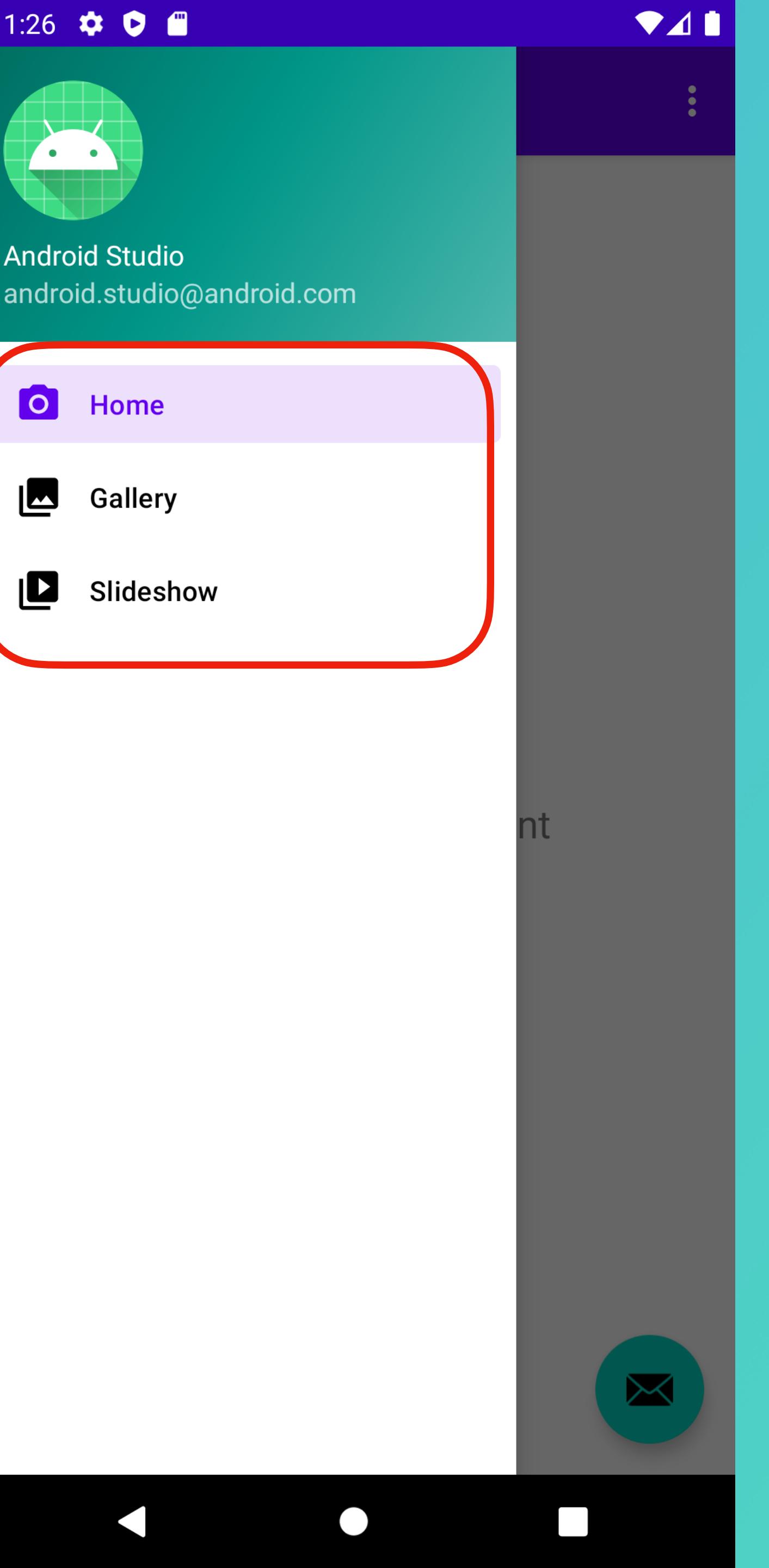
```
buildscript {  
    ext.nav_version = "2.5.3"  
}  
...  
dependencies {  
    implementation "androidx.navigation:navigation-fragment-ktx:$nav_version"  
    implementation "androidx.navigation:navigation-ui-ktx:$nav_version"  
}
```

Add a drawer to a layout

```
<?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" />
    <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/activity_main_drawer" />
</androidx.drawerlayout.widget.DrawerLayout>
```

Declare the menu items

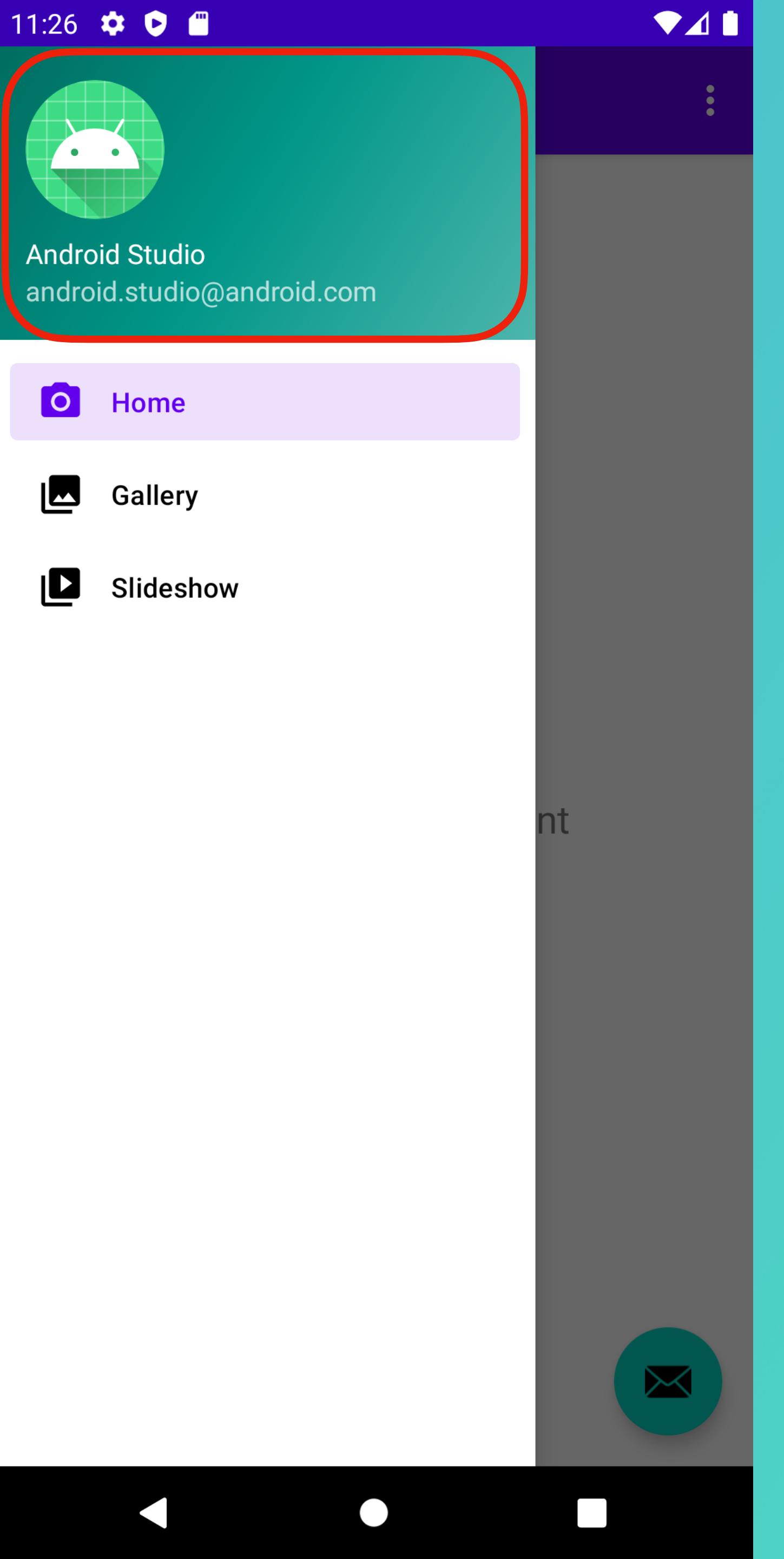
```
<?xml version="1.0" encoding="utf-8"?>
<android.support.design.widget.NavigationView
<menu xmlns:android="http://schemas.android.com/apk/res/android" >
    <group android:checkableBehavior="single">
        <item
            android:id="@+id/nav_view"
            android:layout_width="wrap_content"
            android:layout_height="match_parent"
            android:layout_gravity="start"
            android:icon="@drawable/ic_menu_camera"
            android:fitsSystemWindows="true"/>
        <item
            android:id="@+id/nav_import"
            android:layout_width="wrap_content"
            android:layout_height="match_parent"
            android:layout_gravity="start"
            android:icon="@drawable/ic_menu_gallery"
            android:label="@string/import_label"
            android:title="@string/import" />
        <item
            android:id="@+id/nav_main_drawer"
            android:layout_width="wrap_content"
            android:layout_height="match_parent"
            android:layout_gravity="start"
            android:icon="@drawable/ic_menu_slideshow"
            android:label="@string/slideshow_label"
            android:title="@string/slideshow" />
    ...
</group>
</menu>
```



Add a header to the nav drawer

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.design.widget.NavigationView
    android.support.design.widget.NavigationView
    android:id="@+id/nav_view"
    android:layout_width="wrap_content" parent"
    android:layout_width="wrap_content"
    android:layout_height="match_parent"
    android:layout_height="match_parent"
    android:layout_gravity="start" attr/colorPrimaryDark"
    android:layout_gravity="start"
    android:fitsSystemWindows="true"
    app:menu="@menu/activity_main_drawer" />
    android:orientation="vertical" header_main" />
        android:gravity="bottom">

    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="My header title"
        android:textAppearance=
            "@style/TextAppearance.AppCompat.Body1"/>
</LinearLayout>
```



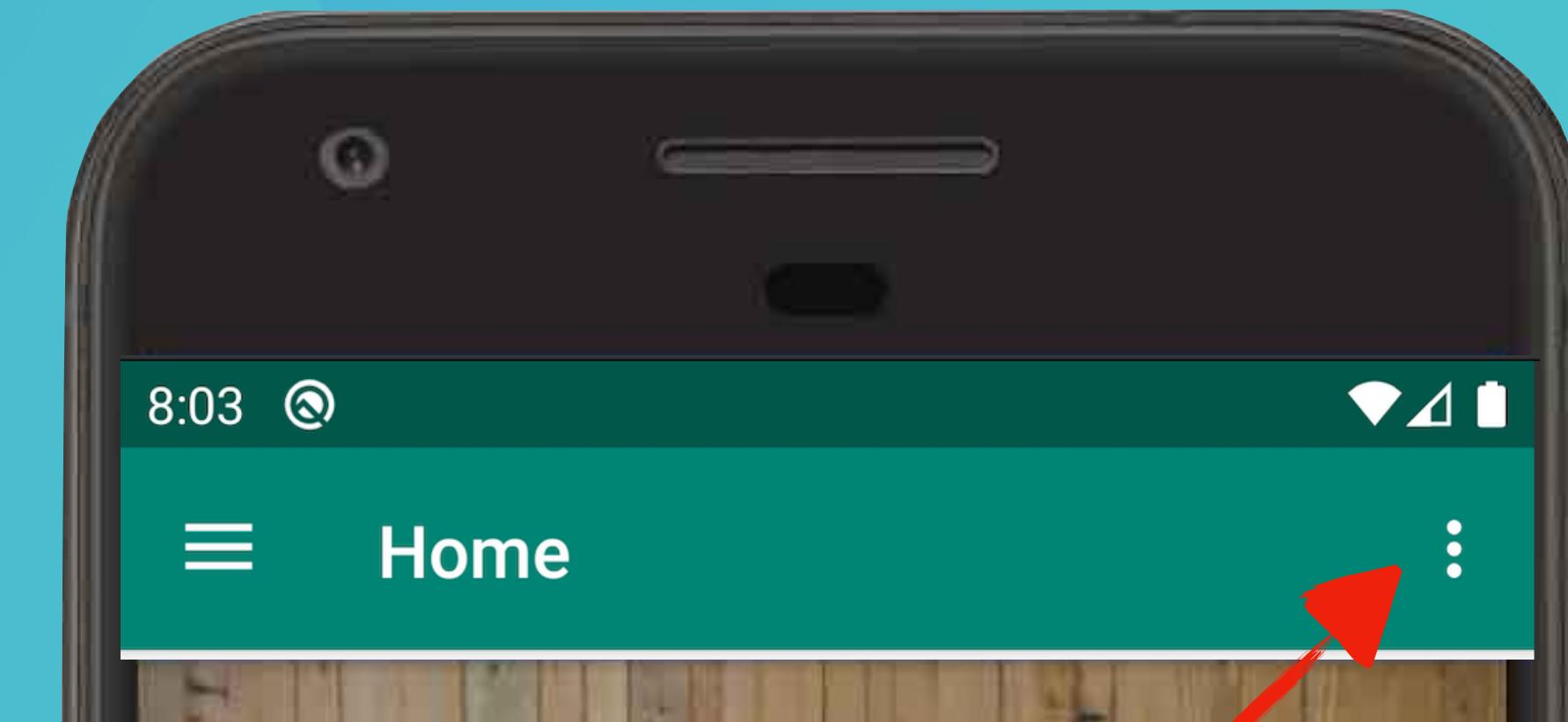
Handle navigation events

```
class MainActivity : AppCompatActivity() {  
  
    private lateinit var appBarConfiguration: AppBarConfiguration  
  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        setContentView(R.layout.activity_main)  
        setSupportActionBar(toolbar)  
  
        val navController = findNavController(R.id.nav_host_fragment)  
  
        appBarConfiguration = AppBarConfiguration(  
            setOf(  
                R.id.nav_home,  
                R.id.nav_gallery  
            ), drawer_layout  
        )  
        setupActionBarWithNavController(navController, appBarConfiguration)  
        nav_view.setupWithNavController(navController)  
    }  
}
```

Add a toolbar

OLD

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.coordinatorlayout.widget.CoordinatorLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    class="MainActivity" AppCompatActivity()
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    val actionBar: ActionBar? = supportActionBar
    android:layout_height="match_parent"
    override fun onCreate(savedInstanceState: Bundle?) {
        tools:context="MainActivity" savedInstanceState)
        <manifest> <application> <activity> <action> <parentActivityName>
        <com.google.android.material.appbar.AppBarLayout>: Boolean {
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:background="@color/colorPrimary"
            android:theme="@style/AppTheme.PopupOverlay" />
        </com.google.android.material.appbar.AppBarLayout>
```



NEW

```
class MainActivity : AppCompatActivity() {
    override fun onCreate(
        savedInstanceState: Bundle?) {
        ...
        setSupportActionBar(toolbar)
        ...
    }
}
```

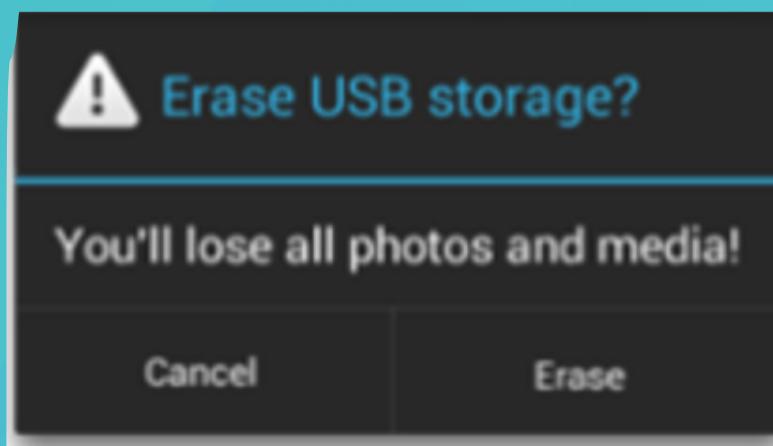
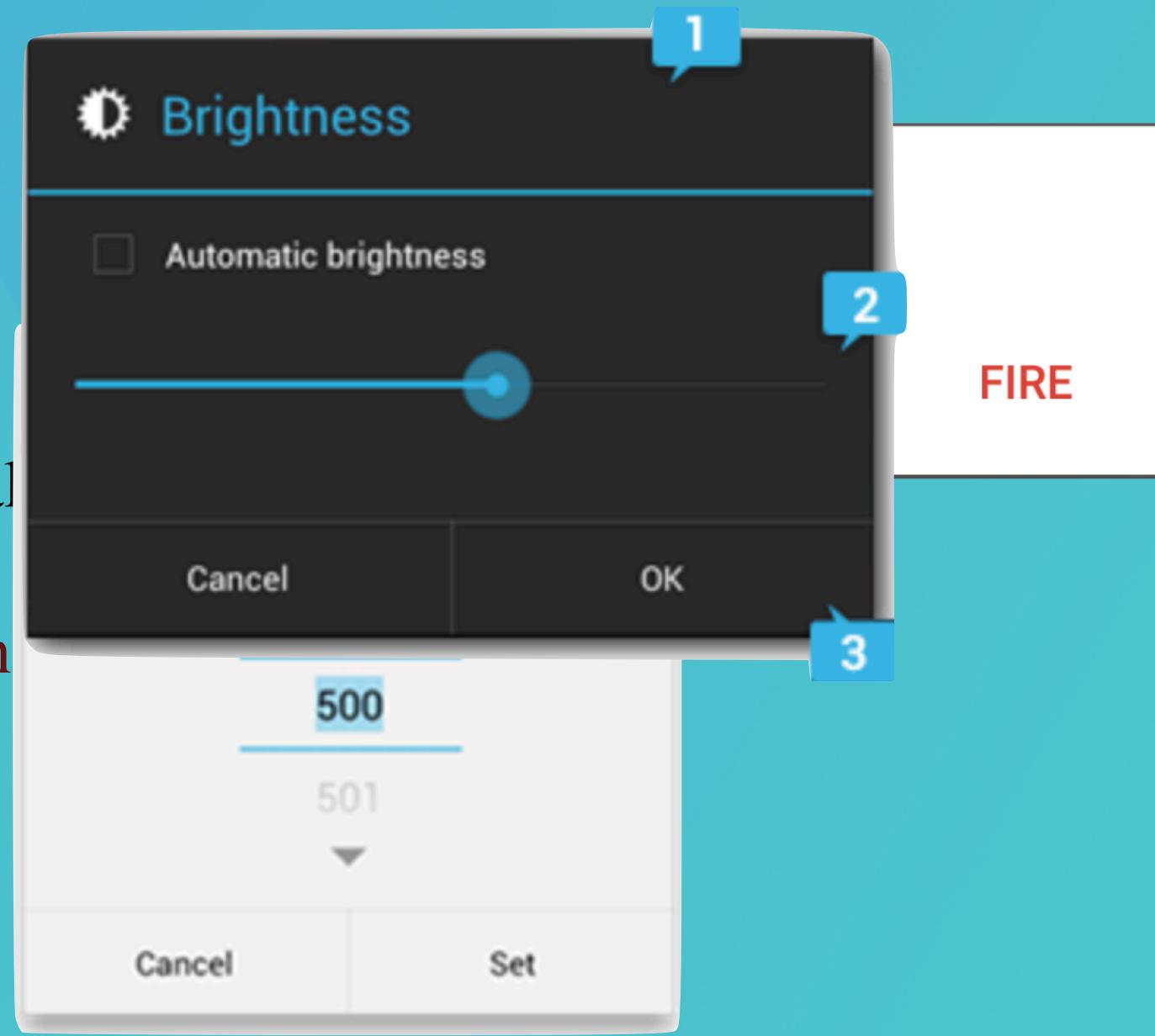
Other State Changes

```
drawer_layout.addDrawerListener(  
    object : DrawerLayout.DrawerListener {  
        override fun onDrawerSlide(drawerView: View, slideOffset: Float) {  
            // Respond when the drawer's position changes  
        }  
  
        override fun onDrawerOpened(drawerView: View) {  
            // Respond when the drawer is opened  
        }  
  
        override fun onDrawerClosed(drawerView: View) {  
            // Respond when the drawer is closed  
        }  
  
        override fun onDrawerStateChanged(newState: Int) {  
            // Respond when the drawer motion state changes  
        }  
    }  
)
```

DEMO

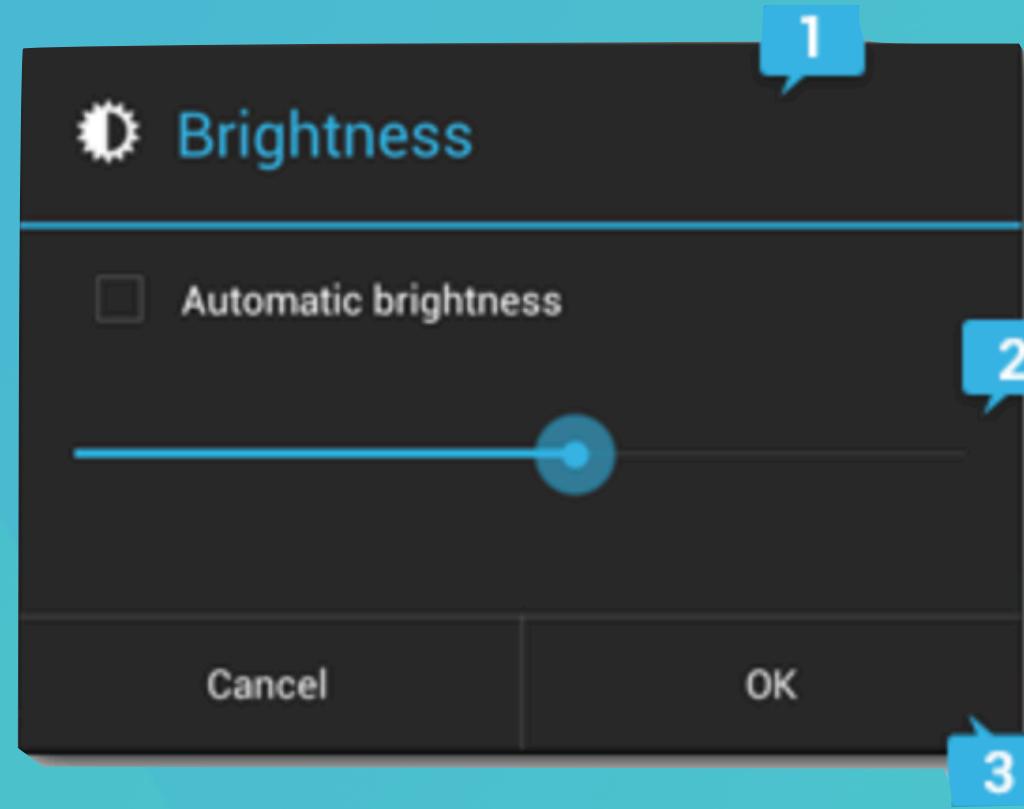
Dialogs

```
class FireMissilesDialogFragment : DialogFragment() {  
  
    override fun onCreateDialog(savedInstanceState: Bundle): Dialog {  
        return activity?.let {  
            // 1. Instantiate an AlertDialog.Builder with its constructor  
            val builder: AlertDialog.Builder? = activity?.let {  
                SmallWindowBuilder(it)  
                    .setMessage(R.string.dialog_fire_missiles)  
                    .setPositiveButton(R.string.fire,  
                        DialogInterface.OnClickListener { dialog, id ->  
                            • SmallWindowBuilder(it)  
                                .setPositiveButton(R.string.fire,  
                                    DialogInterface.OnClickListener { dialog, id ->  
                                        • Prompts the user to take a decision  
                                            // FIRE MISSILES!  
                                            // 2. Chain together various setter methods to set the dialog characteristics  
                                            builder?.setMessage(R.string.dialog_message)  
                                                .setNegativeButton(R.string.cancel,  
                                                    .setTitle(R.string.dialog_title)  
                                                    DialogInterface.OnClickListener { dialog, id ->  
                                                        • Modal, by default  
                                                        // User cancelled the dialog  
                                                        // 3. Get the AlertDialog from create()  
                                                        val dialog: AlertDialog? = builder?.create()  
                                                        // Create the AlertDialog object and return it  
                                                        builder.create()  
                                                        } ?: throw IllegalStateException("Activity cannot be null")  
                                                    }  
                                                }  
                                            }  
                                        }  
                                    }  
                                }  
                            }  
                        }  
                    }  
                }  
            }  
        }  
    }  
}
```



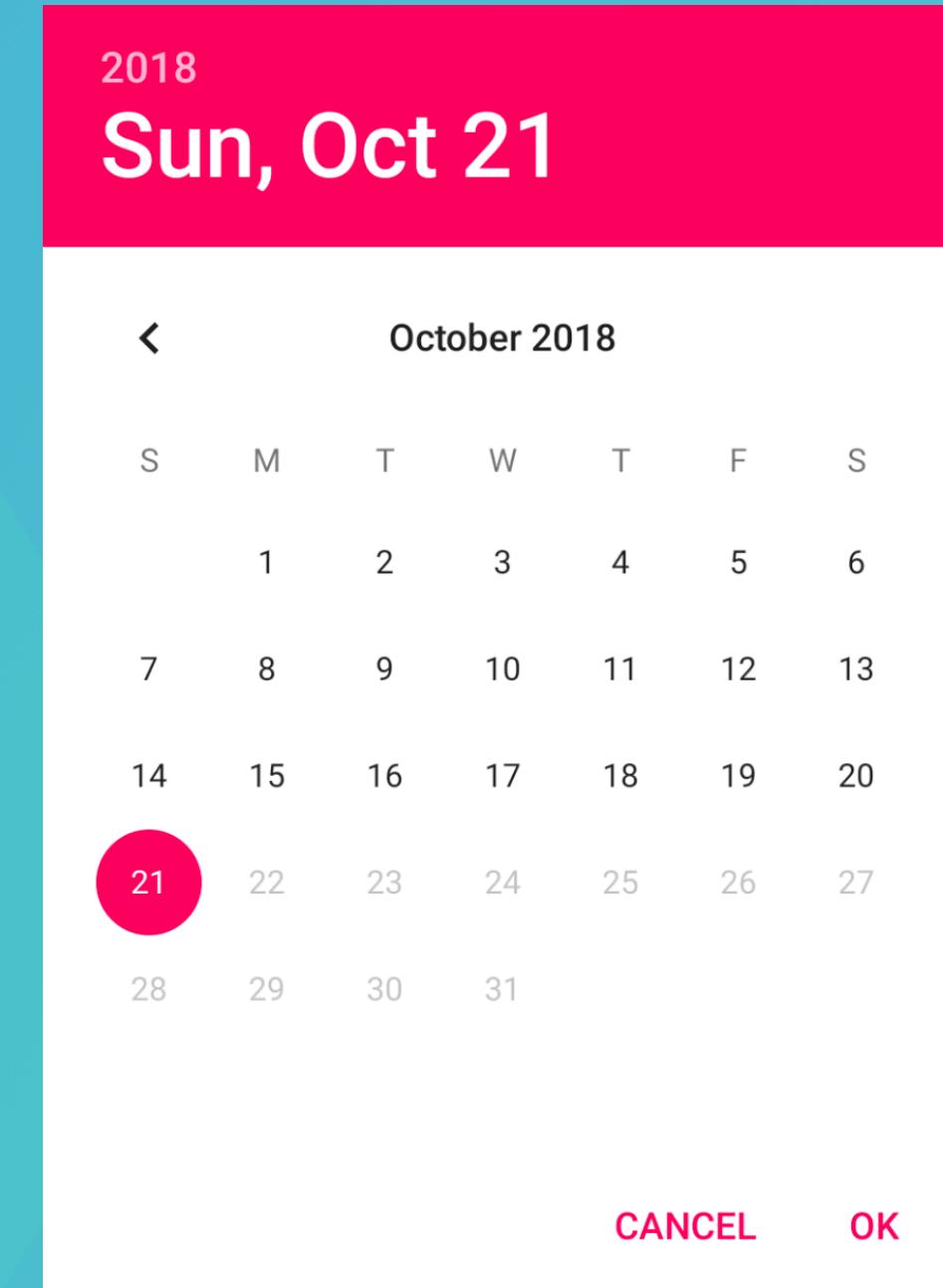
Adding actions

```
val alertDialog: AlertDialog? = activity?.let {  
    val builder = AlertDialog.Builder(it)  
    builder.apply {  
        setPositiveButton(R.string.ok,  
            DialogInterface.OnClickListener { dialog, id ->  
                // User clicked OK button  
            })  
        setNegativeButton(R.string.cancel,  
            DialogInterface.OnClickListener { dialog, id ->  
                // User cancelled the dialog  
            })  
    }  
    // Set other dialog properties  
    ...  
  
    // Create the AlertDialog  
    builder.create()  
}
```



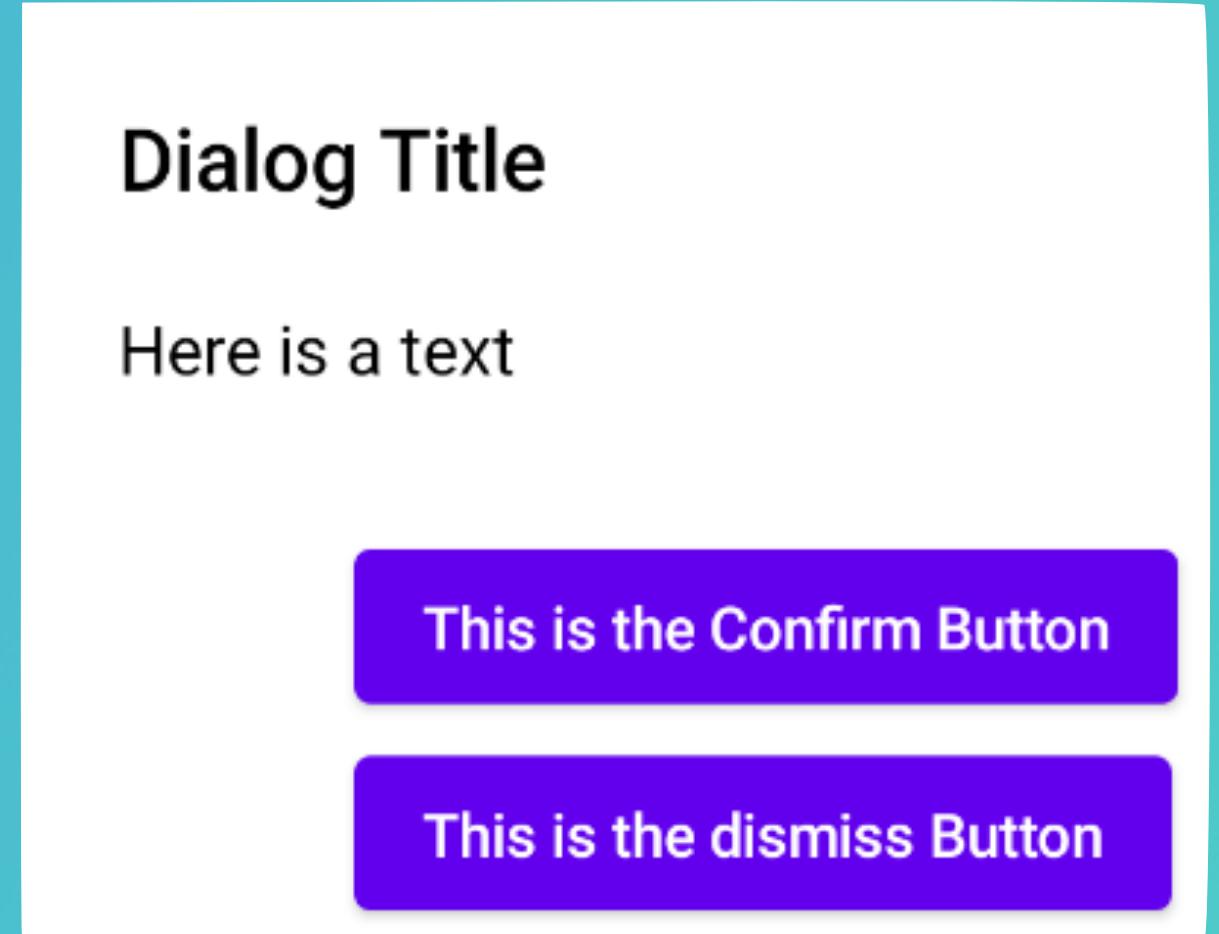
Using Anko

```
alert {  
    isCancelable = false  
    lateinit var datePicker: DatePicker  
    customView {  
        verticalLayout {  
            datePicker = datePicker {  
                maxDate = System.currentTimeMillis()  
            }  
        }  
    }  
    yesButton {  
        val parsedDate =  
            "${datePicker.dayOfMonth}/${datePicker.month + 1}/${datePicker.year}"  
        toast("Selected date: $parsedDate")  
    }  
    noButton { }  
}.show()
```

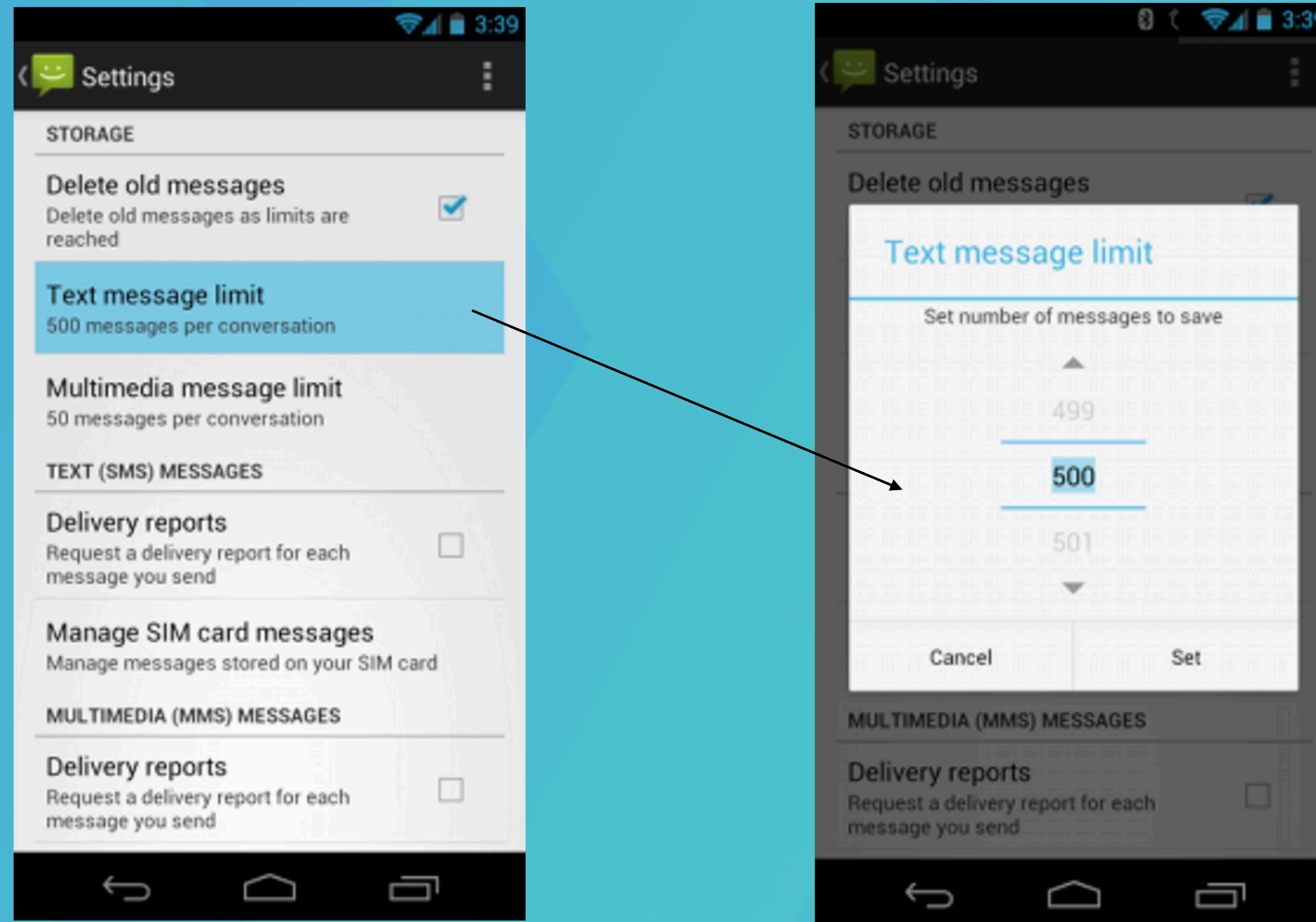


Jetpack Compose

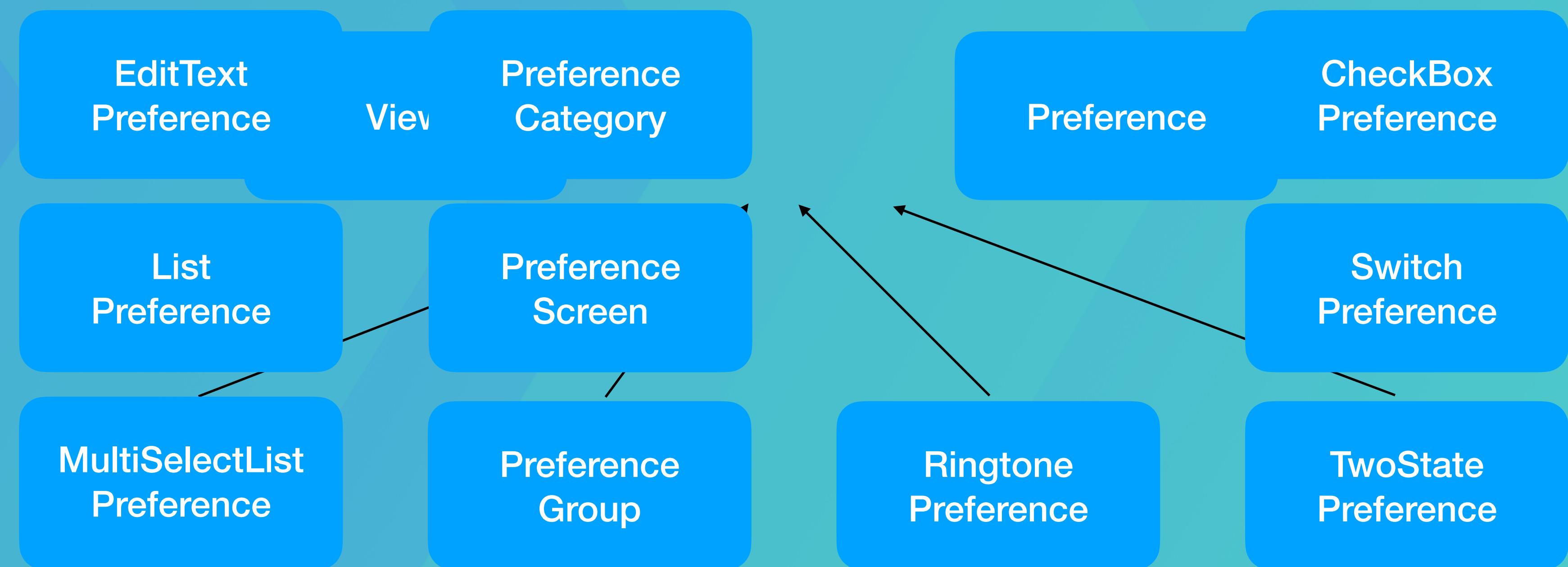
```
AlertDialog(  
    onDismissRequest = {  
        // Dismiss the dialog when the user clicks outside the dialog or on the back  
        // button. If you want to disable that functionality, simply use an empty  
        // onCloseRequest.  
        openDialog.value = false  
    },  
    title = { Text(text = "Dialog Title")},  
    text = { Text("Here is a text ")},  
    confirmButton = {  
        Button(  
            onClick = { openDialog.value = false }  
        ) { Text("This is the Confirm Button") }  
    },  
    dismissButton = {  
        Button(  
            onClick = { openDialog.value = false }  
        ) { Text("This is the dismiss Button") }  
    }  
)
```



Preferences



Preferences



Preferences

Preference

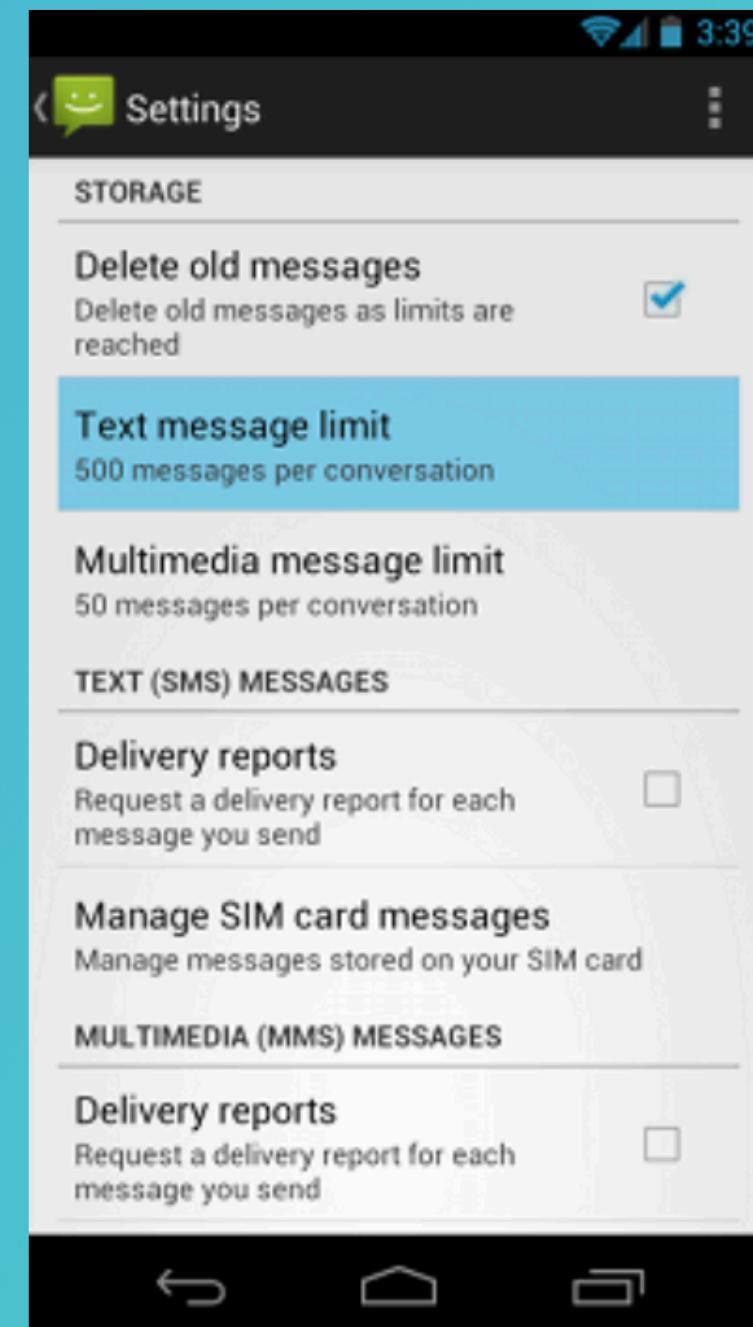
String

Set<String>

DEMO

PreferenceScreen

```
<?xml version="1.0" encoding="utf-8"?>
<PreferenceScreen
    xmlns:android="http://schemas.android.com/apk/res/android">
    <CheckBoxPreference
        android:key="pref_sync"
        android:title="@string/pref_sync"
        android:summary="@string/pref_sync_summ"
        android:defaultValue="true" />
    <ListPreference
        android:dependency="pref_sync"
        android:key="pref_syncConnectionType"
        android:title="@string/pref_syncConnectionType"
        android:dialogTitle="@string/pref_syncConnectionType"
        android:entries="@array/pref_syncConnectionTypes_entries"
        android:entryValues="@array/pref_syncConnectionTypes_values"
        class SettingsActivity> PreferenceActivity() {->
        android.defaultValue="@string/pref_syncConnectionTypes_default" />
        override fun onCreate(savedInstanceState: Bundle) {
            </PreferenceScreen>
            super.onCreate(savedInstanceState)
            addPreferencesFromResource(R.xml.preferences)
        }
    }
```



Jetpack DataStore

| Feature | SharedPreferences | Preferences DataStore | Proto DataStore |
|--|--|--|--|
| Async API | <input checked="" type="checkbox"/> (only for reading changed values, via listener) | <input checked="" type="checkbox"/> (via Flow) | <input checked="" type="checkbox"/> (via Flow) |
| Synchronous API | <input checked="" type="checkbox"/> (but not safe to call on UI thread) | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> |
| Safe to call on UI thread | <input checked="" type="checkbox"/> * | <input checked="" type="checkbox"/> (work is moved to Dispatchers.IO under the hood) | <input checked="" type="checkbox"/> (work is moved to Dispatchers.IO under the hood) |
| Can signal errors | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> |
| Safe from runtime exceptions | <input checked="" type="checkbox"/> ** | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> |
| Has a transactional API with strong consistency guarantees | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> |
| Handles data migration | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> (from SharedPreferences) | <input checked="" type="checkbox"/> (from SharedPreferences) |
| Type safety | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> with Protocol Buffers |

Using DataStore

```
// Preferences DataStore  
implementation "androidx.datastore:datastore-preferences:1.0.0"
```

Create the DataStore

```
// with Preferences DataStore  
val dataStore: DataStore<Preferences> = context.createDataStore(  
    name = "settings"  
)
```

Read Data

```
val MY_COUNTER = preferencesKey<Int>("my_counter")  
val myCounterFlow: Flow<Int> = dataStore.data  
    .map { currentPreferences ->  
        currentPreferences[MY_COUNTER] ?: 0  
    }
```

Using DataStore

Write Data

```
suspend fun incrementCounter() {  
    dataStore.edit { settings ->  
        // We can safely increment our counter without losing data due to races!  
        val currentCounterValue = settings[MY_COUNTER] ?: 0  
        settings[MY_COUNTER] = currentCounterValue + 1  
    }  
}
```

Using DataStore

DEMO

Write Data

```
suspend fun incrementCounter() {  
    dataStore.edit { settings ->  
        // We can safely increment our counter without losing data due to races!  
        val currentCounterValue = settings[MY_COUNTER] ?: 0  
        settings[MY_COUNTER] = currentCounterValue + 1  
    }  
}
```

Migrate from SharedPreferences

```
val dataStore: DataStore<Preferences> = context.createDataStore(  
    name = "settings",  
    migrations = listOf(SharedPreferencesMigration(context, "settings_preferences"))  
)
```

Saving & Reading Local Files

- Internal storage
 - Internal cache files
- External storage
- Shared preferences
- Databases



<https://developer.android.com/guide/topics/data>

Internal Storage

- It's always available.
- Available only to your app.
- On uninstall everything is removed.



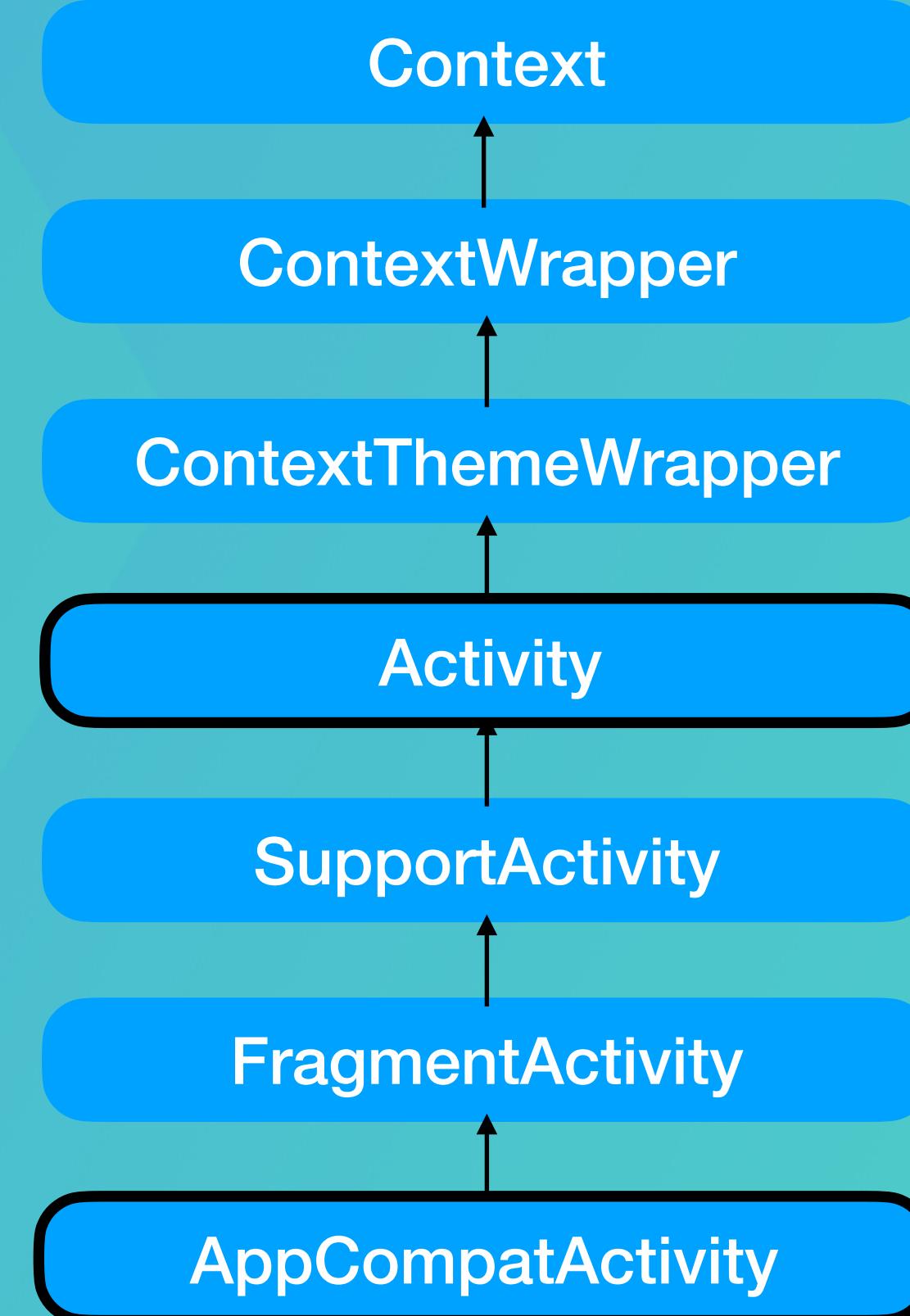
Neither the user nor other apps can access your files!

<https://developer.android.com/guide/topics/data/data-storage#filesInternal>

Internal Storage



```
public abstract class Context {  
    val file = File(context.filesDir, filename)  
    ...  
    public abstract File getFilesDir();  
    val filename = "myfile"  
    ...  
    val fileContents = "Hello world!" File getCacheDir();  
    context.openFileOutput(filename,  
        Context.MODE_PRIVATE).use {  
            it.write(fileContents.toByteArray())  
    }  
  
    private fun getTempFile(  
        context: Context, url: String): File? =  
        Uri.parse(url)?.lastPathSegment?.let { filename ->  
            File.createTempFile(filename, null, context.cacheDir)  
    }
```



External Storage Permissions

```
<manifest ...>
    <uses-permission android:name=
        "android.permission.WRITE_EXTERNAL_STORAGE"/>
    <uses-permission android:name=
        "android.permission.READ_EXTERNAL_STORAGE" />
</manifest>
...
</manifest>
```

<Android 4.4 (API level 19)

```
/* Checks if external storage is available for read and write */
fun isExternalStorageWritable(): Boolean {
    return Environment.getExternalStorageState() == Environment.MEDIA_MOUNTED
}
/* Checks if external storage is available to at least read */
fun isExternalStorageReadable(): Boolean {
    return Environment.getExternalStorageState() in
        setOf(Environment.MEDIA_MOUNTED, Environment.MEDIA_MOUNTED_READ_ONLY)
}
```



FileProvider

DEMO

```
content://com.example.myapp.fileprovider/myimages/default_image.jpg
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.myapp">
    <application
        ...
        <provider
            android:name="android.support.v4.content.FileProvider"
            android:authorities="com.example.myapp.fileprovider"
            android:grantUriPermissions="true"
            android:exported="false">
            <meta-data
                android:name="android.support.FILE_PROVIDER_PATHS"
                android:resource="@xml/filepaths" />
        </provider>
        ...
    </application>
</manifest>
<paths>
    <files-path path="images/" name="myimages" />
</paths>
```

<https://developer.android.com/training/secure-file-sharing/setup-sharing>

Lecture outcomes

- Navigate between screens/views.
- Use dialogs and pickers.
- Manage files & preferences.

