# Task Core 2 – Spike: Taking Chances

**Link to GitHub repository: https://github.com/SoftDevMobDevJan2024/core2-104177995**

## Goals:

- Demonstrate the ability to create a multi-activity app.
- Share data between activities using intents.
- Utilize images and resources effectively.
- Work with complex UI widgets.
- Implement advanced data sharing techniques.
- Design user-friendly interfaces with consistent styles.
- Implement error handling and user notifications

## Tools and Resources Used

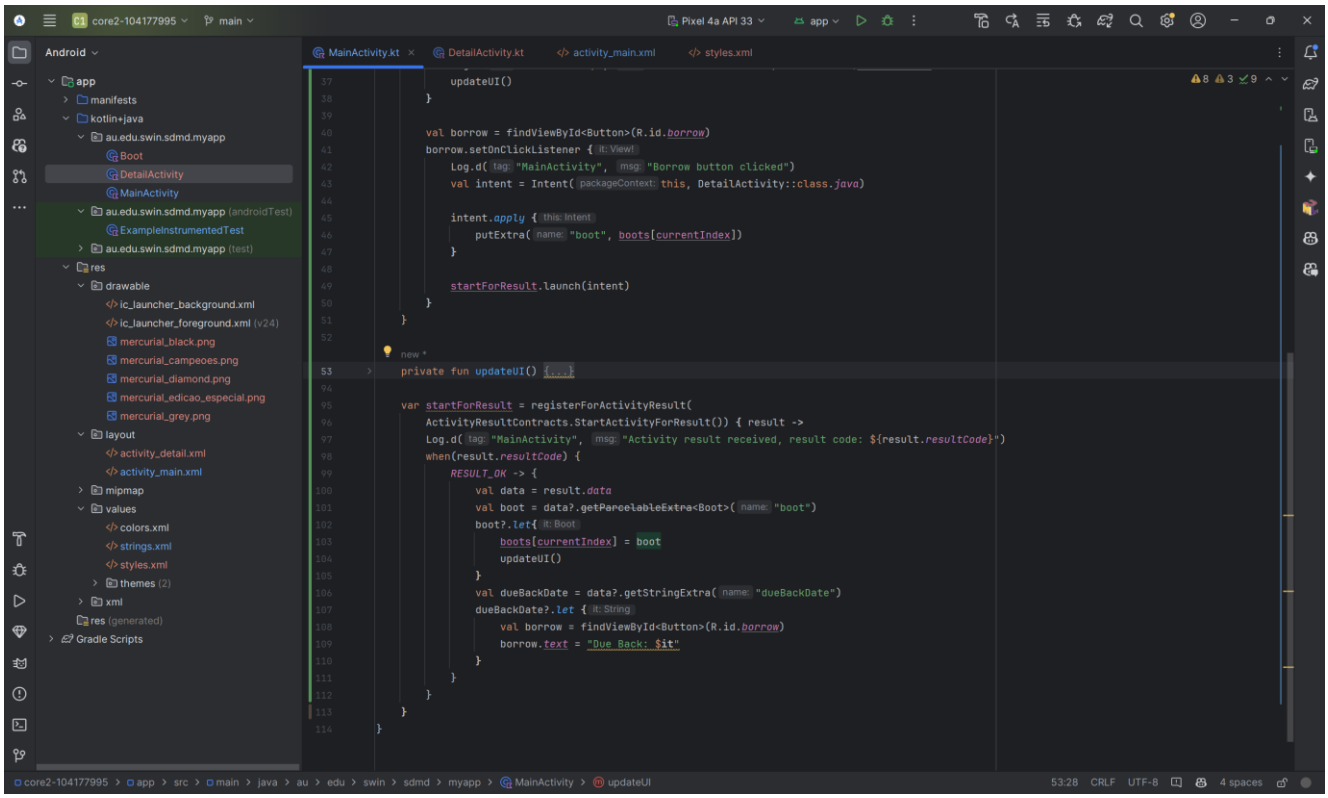
- The course's modules
- GitHub and Git
- Kotlin programming language and XML files
- Android Studio IDE
- Parcelable and Intents documentation: Parcelable and Intents
- Espresso for UI testing: Espresso
- UI components and styles: Android UI Components and Themes and Styles
- Toasts: Toasts

## Knowledge Gaps and Solutions

### Gap 1: ActivityResultContracts

In this assignment, I have utilized the ActivityResultContracts.StartActivityForResult() to initiate DetailActivity and receive the result. This is a new way of handling activity results, which is more type-safe and intuitive compared to the traditional startActivityForResult() method.

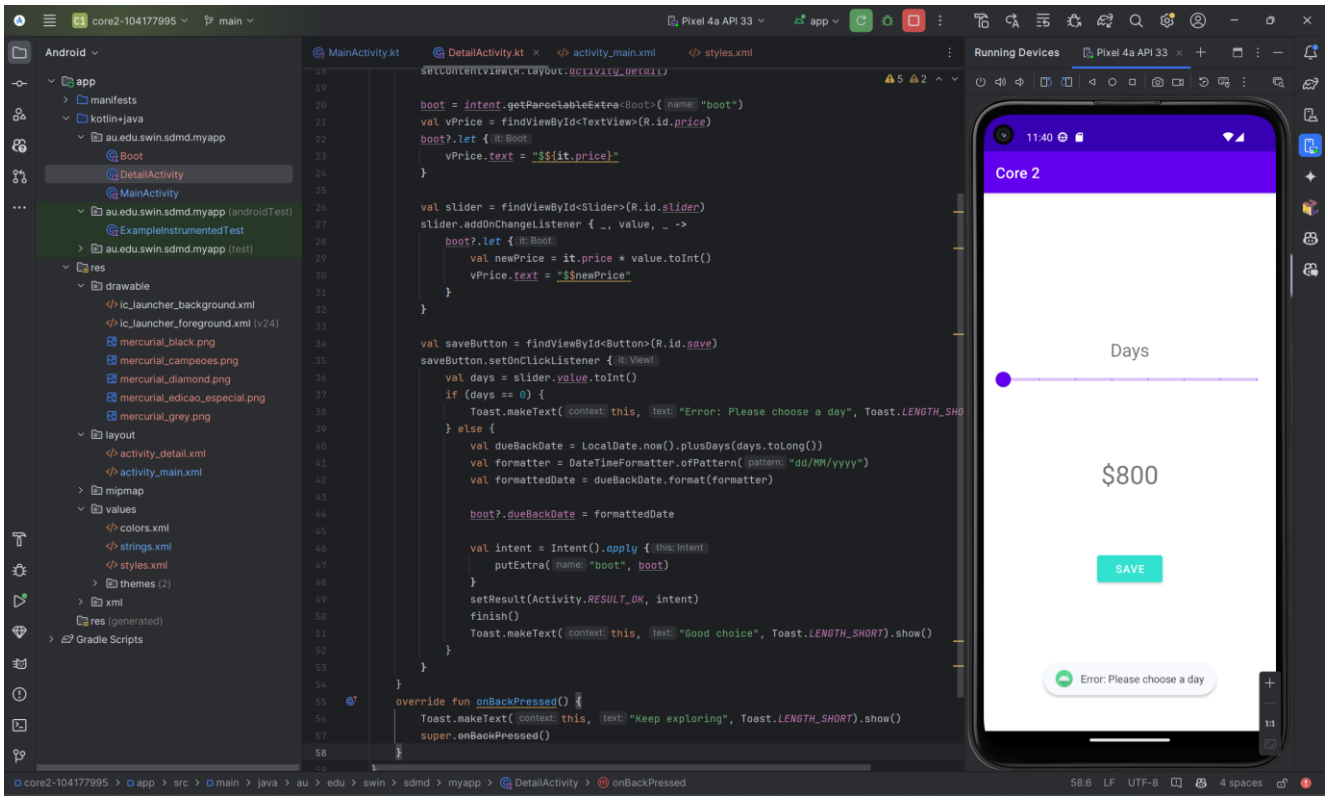
```
var startForResult = registerForActivityResult(
    ActivityResultContracts.StartActivityForResult())
        { result -> // Handle the result here }
    startForResult.launch(intent)
```




## Gap 2: Implement validation for user input

I have implemented user input validation in DetailActivity.kt. Specifically, I check if the slider value (number of days) is zero before proceeding. If it's zero, a Toast message is displayed to inform the user to choose a valid number of days.

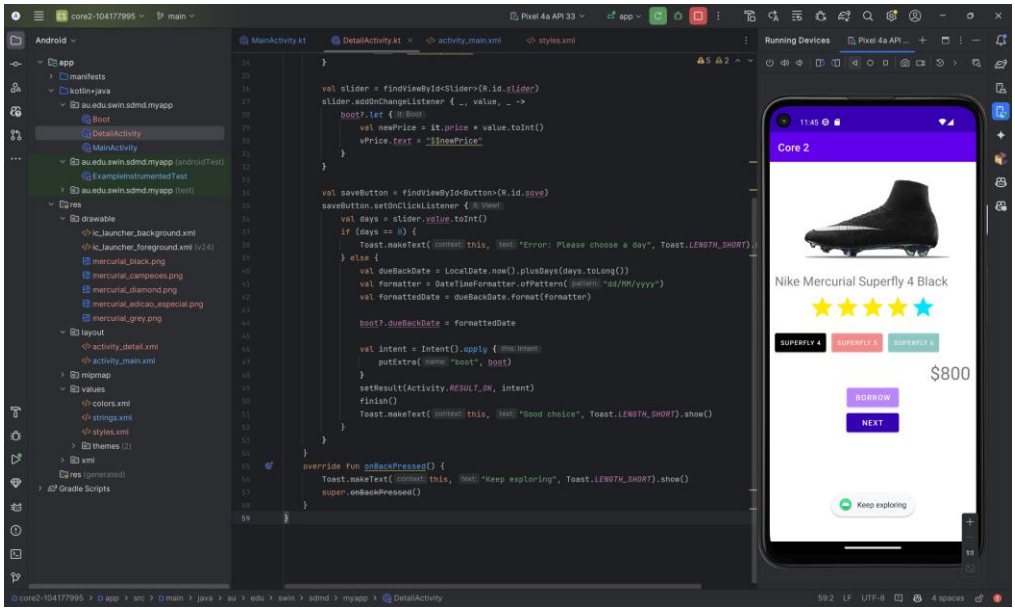
```
val days = slider.value.toInt()

if (days == 0) {
    Toast.makeText(this, "Error: Please choose a day",
    Toast.LENGTH_SHORT).show()
} else {
    // Proceed with the operation
}
```



## Gap 3: Toasts and Snackbars

I have used Toast messages in DetailActivity and MainActivity to display simple messages to the user. This is a straightforward and effective way to provide feedback or instructions to the user. Here's an example from onBackPressed function.

```
override fun onBackPressed() {
        Toast.makeText(this, "Keep exploring", Toast.LENGTH_SHORT).show()
        super.onBackPressed()
}
```

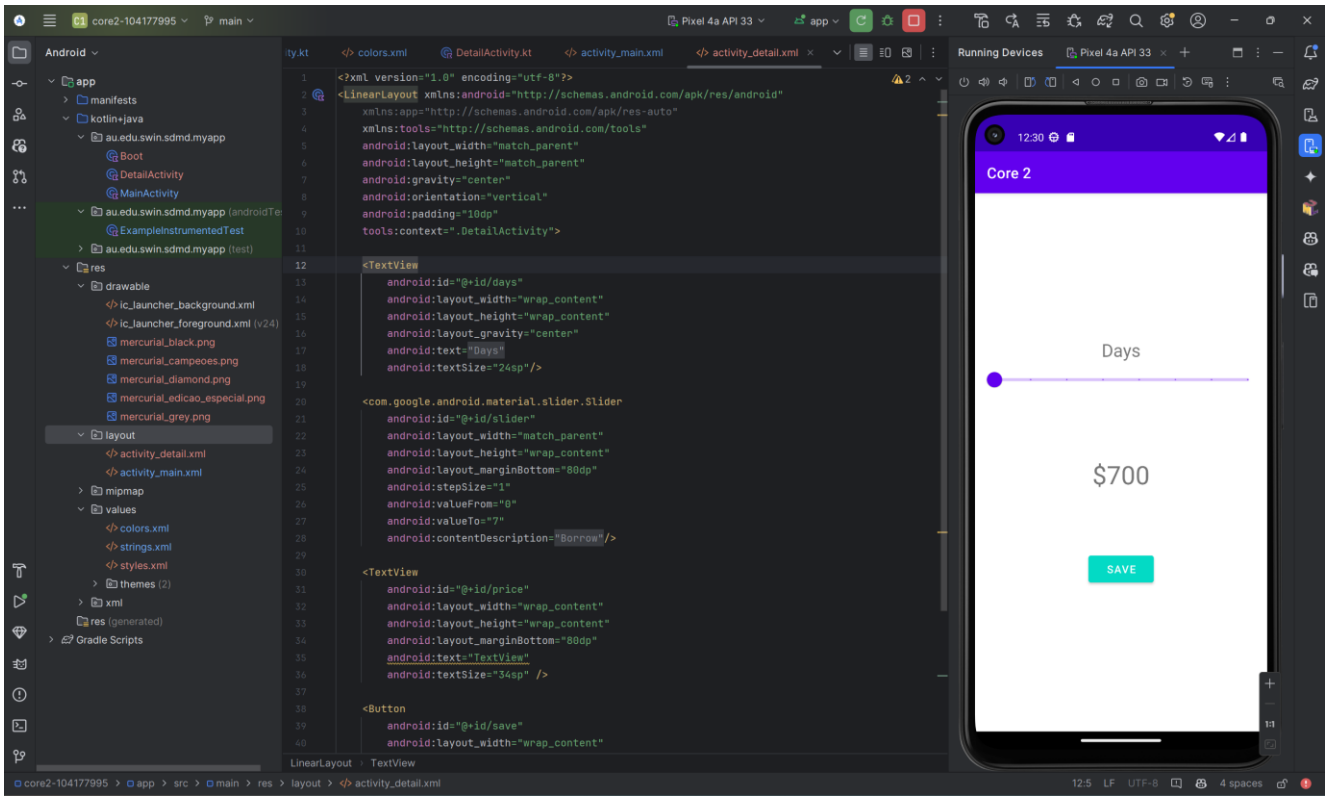


## Gap 4: Unit and UI tests

In ExampleInstrumentedTest.kt, I have written several UI tests that check the visibility of buttons and the functionality of the "Next" button. These tests help ensure the correctness of the application's UI and its interactions.

## Gap 5: Use a wider range of UI elements

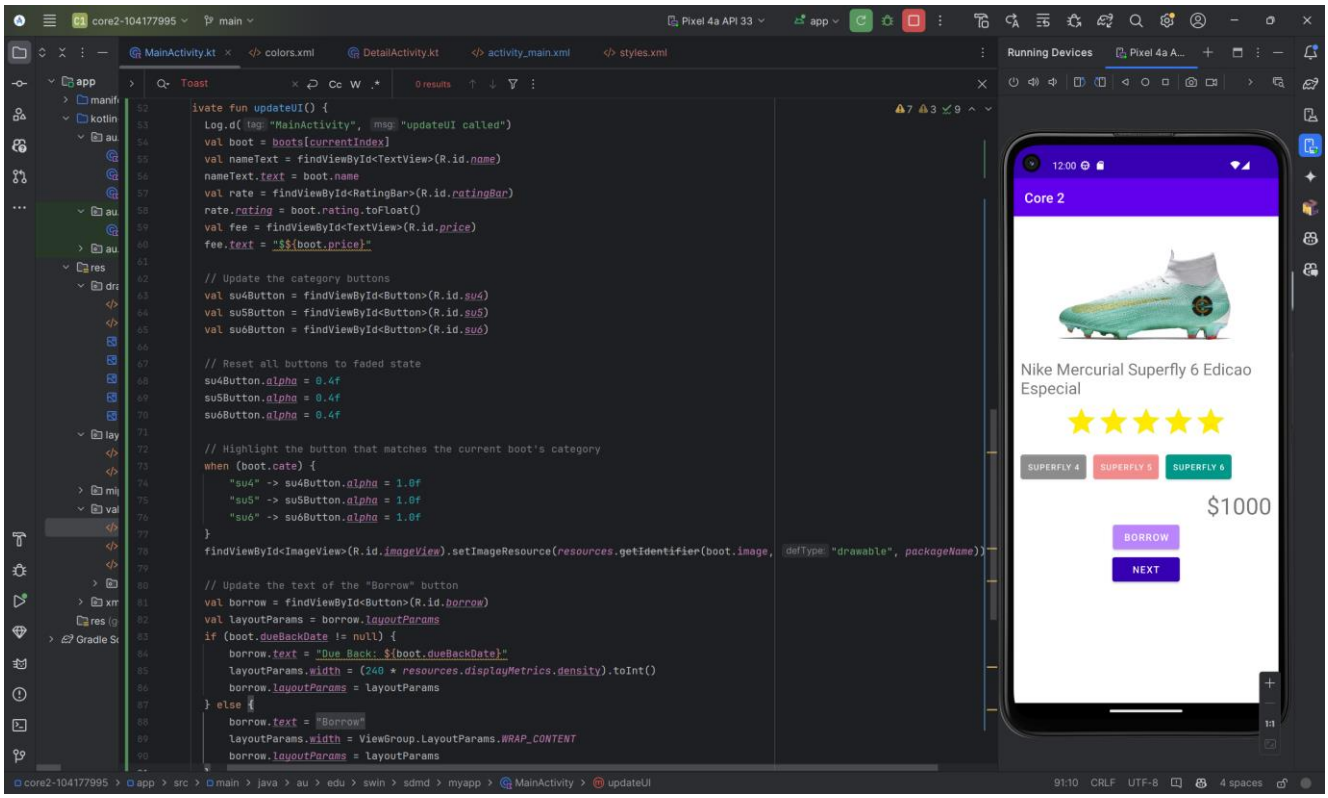
In this assignment, I have used a variety of UI elements including Button, TextView, ImageView, RatingBar, and Slider. These elements provide a rich and interactive user interface for the application.




## Gap 6: Work with resources

I have worked with image resources in MainActivity and color resources in the layout files. This demonstrates my ability to use resources effectively to enhance the look and feel of the application. Here's the example of the usage of image resources in MainActivity.kt.

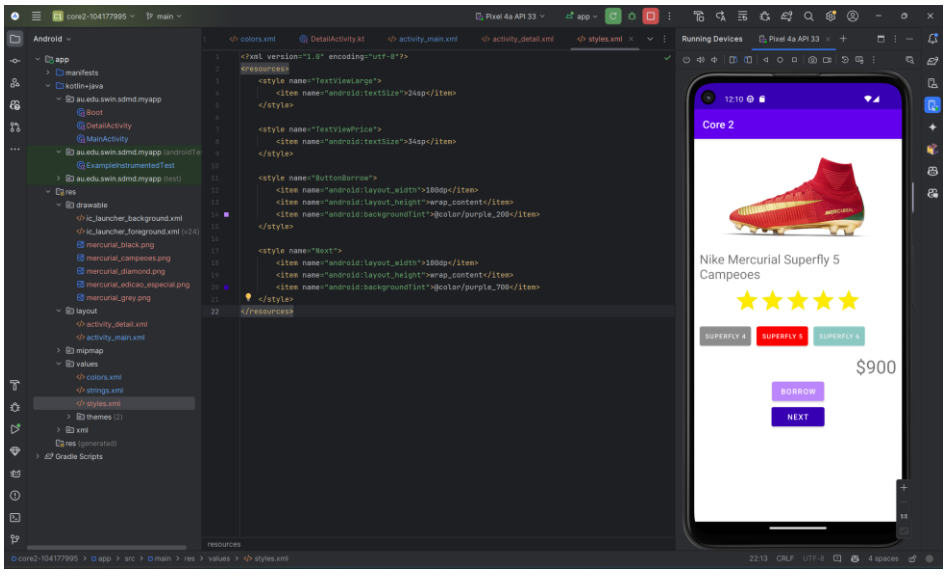
```
findViewById<ImageView>(R.id.imageView).setImageResource(resources.
getIdentifier(boot.image, "drawable", packageName))
```




## Gap 7: Use styles in an app

I have defined multiple styles in styles.xml and used them in the layout files. This not only makes the layout files cleaner and easier to manage, but also promotes consistency in the application's appearance. Here's an example of ButtonBorrow style which is inside the styles.xml.

```
<style name="ButtonBorrow">
        <item name="android:layout_width">100dp</item>
        <item name="android:layout_height">wrap_content</item>
        <item name="android:backgroundTint">@color/purple_200</item>
</style>
```



## Gap 8: Sketch and implement simple screens

I have sketched and implemented two screens (MainActivity and DetailActivity) using basic UI components. This shows my ability to design and build user interfaces for Android applications.

- **User Story 1: View and Borrow Boots**

As a user, I want to view a list of available boots and borrow one, so that I can choose the best boot for my needs.

- **Use Case for User Story 1:**

1. Open the app → Display the MainActivity screen with a list of boots.
2. Click on Next → Show various boots with images and basic information.
3. Click on Borrow → Launch DetailActivity to show detailed information about the renting price selected boot.
4. Drag the slider to choose how many days to rent the boot → The renting price will be updated based on the renting days.
5. Click on Save → Display the MainActivity screen with a due date to return the boot.

- **User Story 2: Rate Boots**

As a user, I want to rate a boot after viewing its details, so that other users can benefit from my experience.

- **Use Case for User Story 2:**

1. Open the app → Display the MainActivity screen with a list of boots.
2. Click on the RatingBar to rate the boot → Submit the rating.

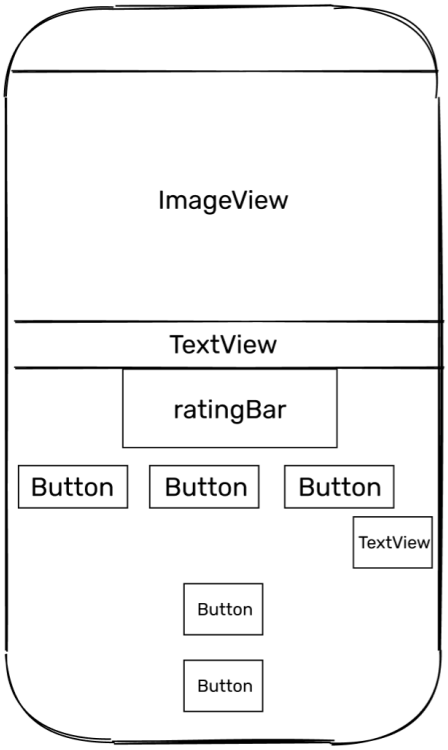
- **User Story 3: Navigate Back to Boot List**

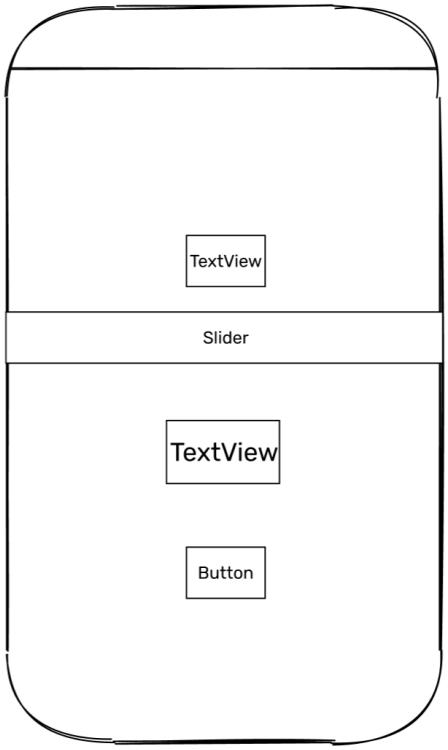
As a user, I want to easily navigate back to the list of boots after viewing the renting price of a particular boot, so that I can continue browsing other boots.

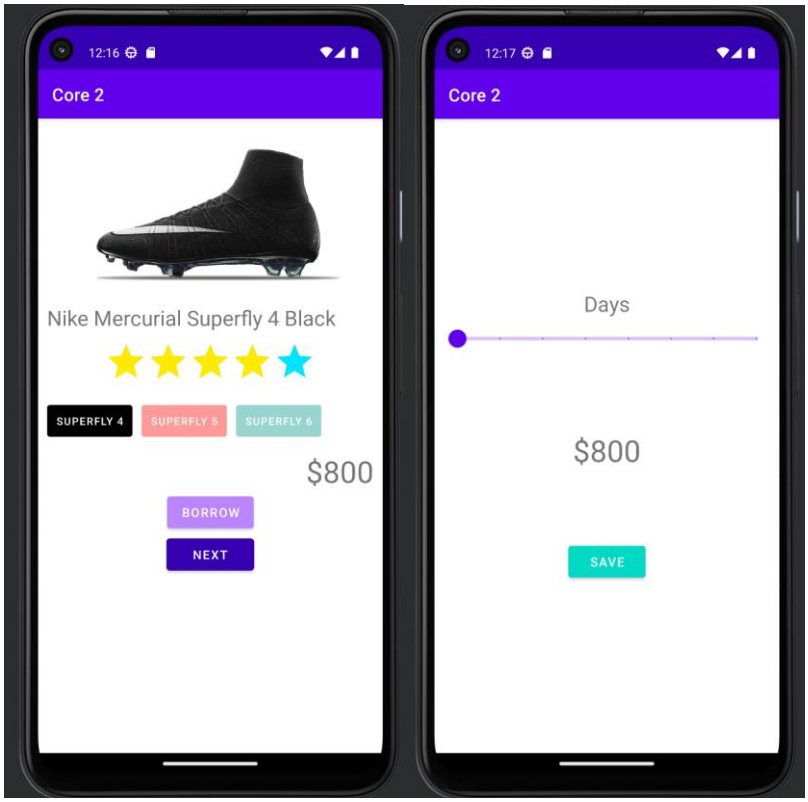
- **Use Case for User Story 3:**

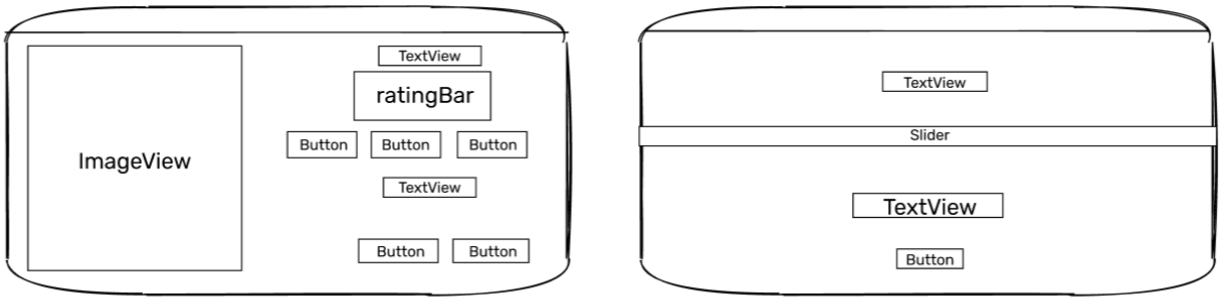
1. Open the app → Display the MainActivity screen with a list of boots.
2. Click on Borrow → Launch DetailActivity to show detailed information about the renting price selected boot.
3. Press the back button → Display a Toast message "Keep exploring" and navigate

back to MainActivity.



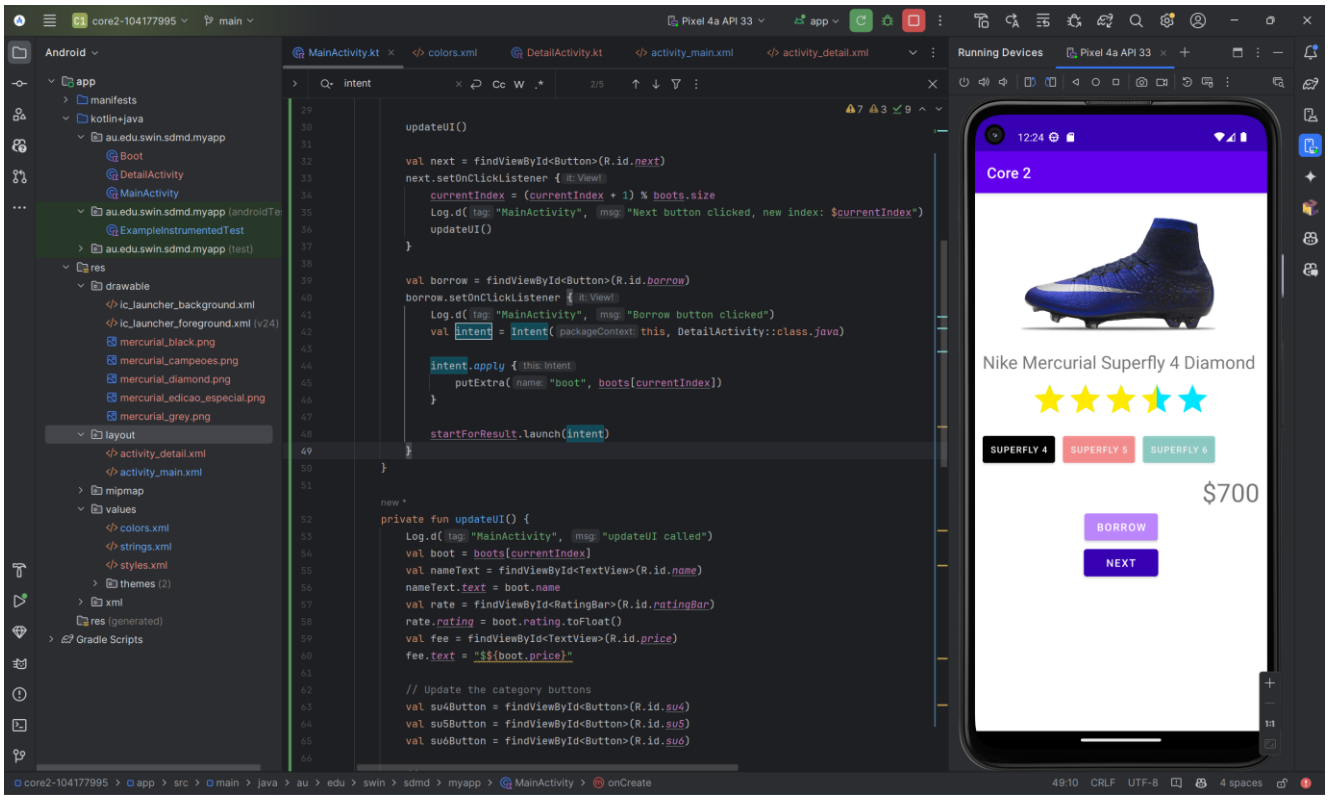




## Gap 9: Understand intents

I have used an Intent to start DetailActivity and pass data to it. This demonstrates my understanding of intents and their use in Android for starting activities and sharing data between them.

```
val intent = Intent(this, DetailActivity::class.java)
intent.apply {
    putExtra("boot", boots[currentIndex])
}
startForResult.launch(intent)
```



## Gap 10: Command of IDE

I have demonstrated command of the Android Studio IDE by using its features like Logcat for logging and ActivityScenarioRule for testing. The build process is handled by Gradle, showing my understanding of the build process in Android development.