Task Core 2– Spike: [Sharing is Caring]

https://github.com/SoftDevMobDev-2023-Classrooms/core2-MarcoGiacoppo

Goals:

The goal of this task is to make an application using Kotlin in Android Studio. We are tasked to create a mobile app which consists of functions to allow users to have the capabilities to rent a certain item. In this case, I chose motorbikes. Users will be able to slide through different kinds of bikes. In my app, there will be 3 different motorbikes to choose between. Users will be able to "rent" the bike. When the 'rent' button is clicked, users will be directed to a different page which shows the bike in more detail. Users will be presented with a 'seekBar' which allows users to pick how many days they're going to rent the motorbike for. Users will also be presented with the total price based on how many days they decide to rent them for. After saving, they'll be directed to the home page and they'll be shown when they need to bring the motorbike back

The app consists of 5 main functions:

- The 'Next' button; the next button is used to look through different images and choose the bset bike for them. Between the images, there will also be a ratingBar, a chipGroup, and a price showing how much it would cost per day.
- The 'Rent' button; the rent button is used when users have decided which motorbike is the best for them. After clicking on it, users will be directed toa another page.
- The 'Save' button; the save button is used when users have made up their mind on how many days the will be renting it for. This will direct them to the 1st page and now it'll show when the due date is based on how many days they're renting it for.
- The 'seekBar' slider; this slider is used by users to decide how many days they'll be renting the motorbike for. I've also implemented some 'toolTip' to help users show how many days they're choosing.
- When users change their mind about the motorbike they chose, after pressing the rent button, they can also go back using the in-built back button on the emulator. This action will be considered a 'cancellation' where a toast will show saying 'Hope you find your perfect bike!'

Tools and Resources Used

This section lists related software, tools, libraries, API's, and other resources used for this knowledge gap.

- Android Developers
- W3School
- YouTube Videos
- Modules on previous weeks
- GitHub
- Android Studio IDE

Knowledge Gaps and Solutions

This section presents the listed knowledge gaps and their solutions with supporting images, screenshots and captions where appropriate/required.

Gap 1: Able to pass back data using intents

Here I used intent to pass back data based on the user input in the 2nd page. When users presses the 'save' button, the data will be passed to be 1st page and will show data based on the users input. For example, if the date today is 24th of September, and the users decides to rent the motorbike for 4 days, the app will parse the data and sends it to the 1st page showing the due date, which is 28th of September (4 days from the day they rented it).

```
saveButton.setOnClickListener { it: View!

    if (progress == 0){
        Toast.makeText(applicationContext, text: "Minimum 1 day bruv!", Toast.LENGTH_SHORT).show()
    } else {
        val returnDate = currentDate.plusDays(progress.toLong())

        val resultIntent = Intent()
        resultIntent.putExtra( name: "tabIndex", tabs)
        resultIntent.putExtra( name: "isButtonDisabled", value: true)
        resultIntent.putExtra( name: "returnDate", returnDate.toString())
        setResult(RESULT_OK, resultIntent)
        finish()
        Toast.makeText(applicationContext, text: "Woohooo!! Happy Riding :)", Toast.LENGTH_SHORT).show()
```

Figure 1: Using intents

Gap 2: Implement validation for user input

Here I used an if statement to make sure that users put in more than 0 days. Because you can't technically rent items for 0 days. Hence, the minimum day is 1 and I implemented a toast showing that there's a minimum of 1 day to be able to rent the item.

```
saveButton.setOnClickListener { it: View!

    if (progress == 0){
        Toast.makeText(applicationContext, text: "Minimum 1 day bruv!", Toast.LENGTH_SHORT).show()
    } else {
        val returnDate = currentDate.plusDays(progress.toLong())
```

Figure 2: Validation for user input

Gap 3: Toasts

Here I used the inbuilt command which is onBackPressed and also wrote a code to show a toast whenever the function is triggered. I don't know why it shows ~~onBackPressed~~. I tried searching on the internet why this happens and it says that it indicates that the method has been deprecated or overridden from a parent class but is no longer needed. But when I remove this, the code doesn't work. Therefore, I decided to keep it there perchance it is still used even though it has a strike through.

```
override fun onBackPressed() {
    Toast.makeText( context: this, text: "Hope you find your perfect bike!", Toast.LENGTH_SHORT).show()

    super.onBackPressed()
}
```

Figure 3: Using toasts

Gap 4: Unit and UI Tests

Here's one of the examples of the testing I did. I tested the progress of the seekBar to see if it works. All the test thankfully passed.

```
@Test
fun testSeekBar() {
    val nextButton : ViewInteraction! = onView(withId(nextButton))
    val rentButton : ViewInteraction! = onView(withId(rentButton))

    nextButton.perform(click())
    rentButton.perform(click())

    val textView : ViewInteraction! = onView(withId(total))
    textView.check(matches(withText( text: "$149")))

    val totalRate : Double = 7 * 149.0
    onView(withId(bar)).perform(swipeRight())
    textView.check(matches(withText( text: "$$totalRate")))
}
```

Figure 4: Unit testing

Gap 5: Use a wider range of UI elements

I used different kinds of UI elements including ratingBar, chipGroup, and also seekBar. I decided to use seekBar instead of slider because I've already used seekBar.

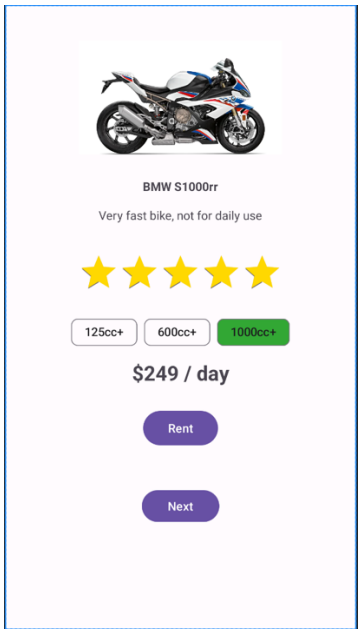

Figure 5: Using different UI elements

Gap 6: Work with resources

Here I created a data class which is used in the itemList to parse through different images.

```
data class Item(
    val imageResId: Int,
    val title: String,
    val description: String,
    val rating: Float,
    val chips: List<String>,
    val price: String,
    var returnDate: String)

// Create a list of items
private val itemList = listOf(
    Item(R.drawable.bmw,  title: "BMW S1000rr",  desc
    Item(R.drawable.cb125r,  title: "Honda CB125r",
    Item(R.drawable.r6,  title: "Yamaha R6",  descriptio
)
```

Figure 6: Using itemList to parse images

Gap 7: Use styles in app

I decided to use different types of colors for this part. I used the color gold for the ratingBar and a green coloured button for my saveButton. To use the green button, I added a new resource file in the values folder, and to use the gold color I only added a new color in the colors.xml file.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="purple">#911e9c</color>
</resources>
```

```
<RatingBar
    android:id="@+id/ratingBar"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginStart="85dp"
    android:layout_marginTop="14dp"
    android:layout_marginEnd="85dp"
    android:layout_marginBottom="11dp"
    android:rating="5"
    android:progressTint="@color/gold"
    app:layout_constraintBottom_toTopOf="@+id/chipGroup"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/desc" />
```

```
<Button
    android:id="@+id/save"
    style="@style/GreenButton"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginStart="157dp"
    android:layout_marginTop="2dp"
    android:layout_marginEnd="166dp"
    android:layout_marginBottom="202dp"
    android:text="Save"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/total" />
```

Figure 7: Using different colors for styling

Gap 8: Sketch and implement simple screens

Here I used Figma to draw up a design of how the end result would turn out to be. I actually drew a design when I first started doing the project but somehow I lost the drawing, So I remake the sketch on Figma.

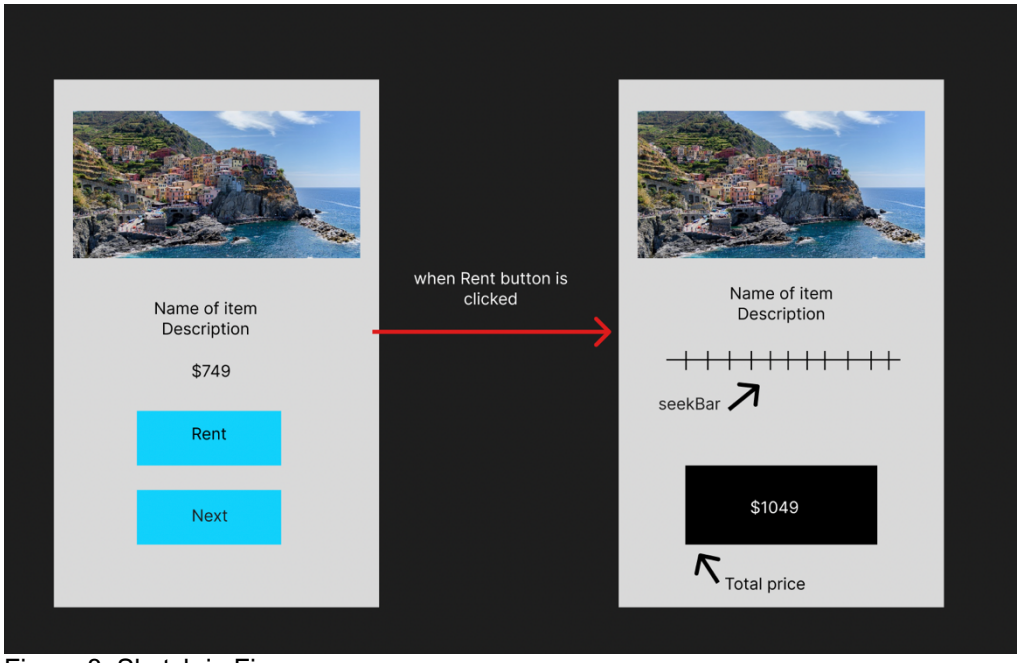

Figure 8: Sketch in Figma

Gap 9: Understanding intents

- Navigating between screens: I used intent to move between different screens or activities in our app. For example, when the user tapped the “Rent” button on the main screen, I created an intent to open the second screen which is “MainActivity2” where users could choose rental options.
- Performing actions: Intents helped me trigger actions within the app. For instance, when the user selected the number of days they wanted to rent a bike and tapped the “Save” button on the second screen, I used an Intent to process their request and update the app’s state accordingly.

Gap 10: Command of IDE

Here I used the log to help me check if my seekBar progress and the selected days are correct. At first, it gave me 2 different inputs because of the position of where my ‘progress’ was, after debugging, I figured that the progress should be inside of my onProgressChanged function.

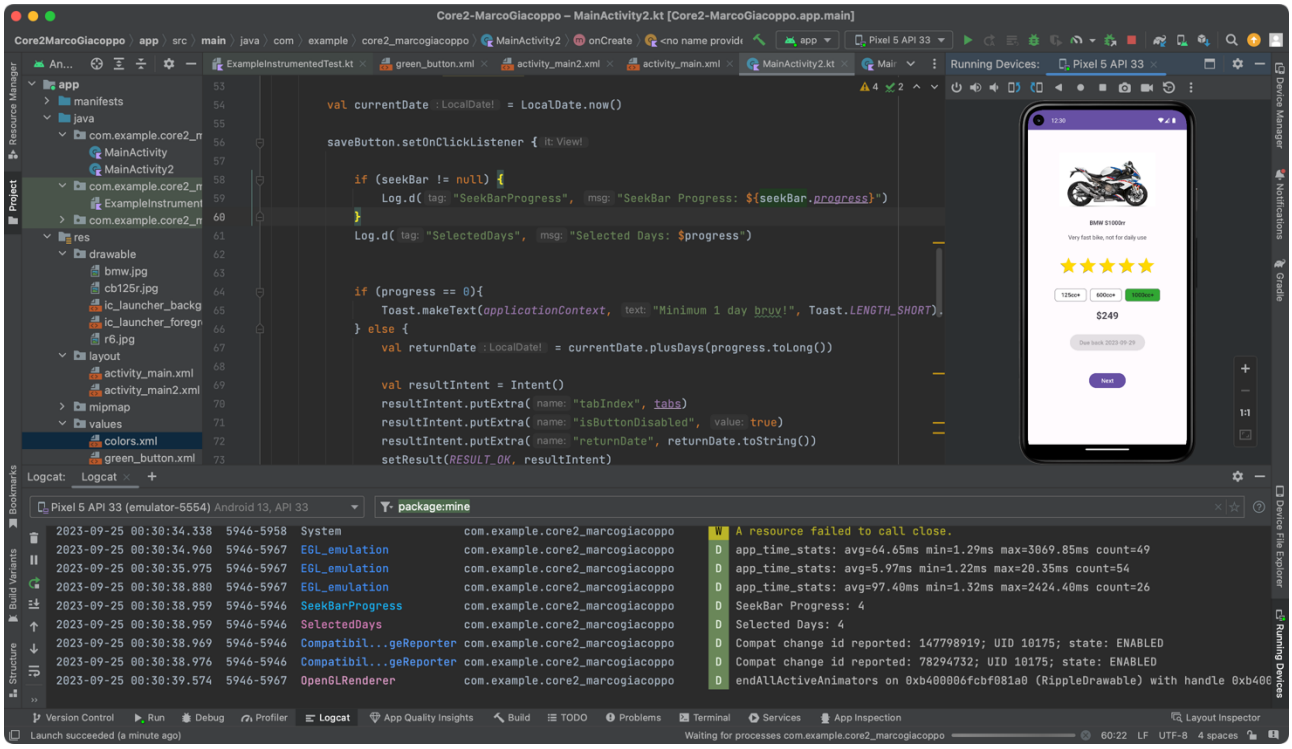

Figure 9: Using Logcat to debug

Open Issues and Recommendations

I had a problem parsing the data for when users set their days in the rent screen. I used the log to check if the progress and the days selected captures the same input, turns out the days selected always captured the input of 1, this was due to the fact that I put the progress outside of my “onProgressChanged” function. I was able to notice this issue by using the debugging tool, without this it would take me way longer to check where the code went wrong.