Task Core 2 – Spike: Extension on data communication and testing

Link to GitHub repository: https://github.com/SoftDevMobDevJan2023/core2-103488117/tree/extension

Goals:

- All the goals presented in task 2 core spike report.
- The skills to communicate data between 2 activities by passing the data through intents back and forth.
- Properly take inputs from the user through conditional checking and input validation. Display proper error messages.
- Implement UI and unit tests. For this I used Espresso.

Tools and Resources Used

- Android Studio IDE
- Git and GitHub
- Kotlin programming language and XML files
- The course's modules
- UI and unit testing: https://developer.android.com/training/testing/espresso/basics
- Passing data:
  https://developer.android.com/reference/kotlin/androidx/activity/result/contract/ActivityResultContracts

Knowledge Gaps and Solutions

Gap 1: EditText input validation

For this task, we are required to change the TextViews in DetailActivity to EditText so that the user can change the data and get more interaction with the app. An important aspect of working with user input is that we need to validate those inputs to assure that the format is correct. Therefore I added a conditional block of code to assign the error widget according to the EditText field that is in the wrong format.

```
if (detailName.text.toString().isEmpty()) detailName?.error = "Name
cannot be empty"
        else if (detailLocation.text.toString().isEmpty())
detailLocation?.error =
            "Location cannot be empty"
        else if (!matcherObj.matches()) detailDate?.error =
            "Date have to be in the right format: dd-mm-yyyy"
```

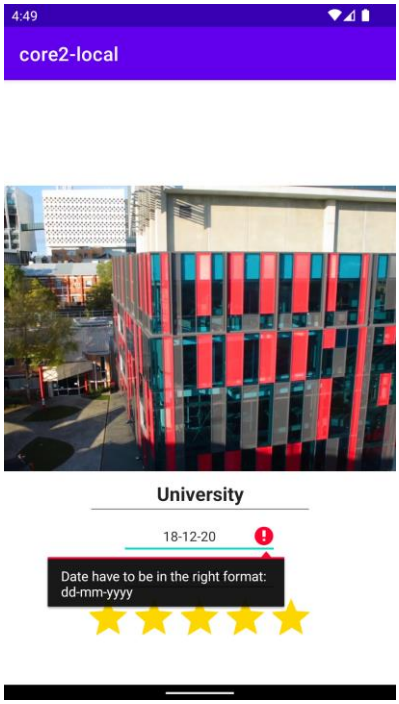

Figure 1: Error message

Gap 2: Communicate data through Intents and ActivityResultContracts

To send the data from the DetailActivity back to the MainActivity, we need to override a function in DetailActivity called onBackPressed. We also need to create a new assign the current data in the views to the attributes of the location object. After that, we apply the putExtra() method of the intent to send a key and the location object back to the MainActivity. The value RESULT_OK also needs to be sent back along the intent.

Back to the MainActivity we use ActivityResultContracts to reassign the TextViews to the changed Parcelable object that we have just received. The attributes of changed is then also applied according to which image/object was clicked on. To check which object should be changed, I added a new attribute to the Location class, which is id. This id will not be changed throughout the program, and each object has its own distinct id that will be checked when we receive the Location object send from the DetailActivity to the MainActivity through intent.

```
// DetailActivity
location?.name = detailName.text.toString()
            location?.rating =
findViewById<RatingBar>(R.id.ratingBar).rating
            location?.date = detailDate.text.toString()
            location?.location = detailLocation.text.toString()

            val i = intent.apply {
                putExtra("changed", location)
            }
            setResult(Activity.RESULT_OK, i)

// MainActivity
private val startForResult =

registerForActivityResult(ActivityResultContracts.StartActivityForResult
()) { result ->
            when (result.resultCode) {
```

```
                RESULT_OK -> {
                    val data = result.data
                    val changed =
data?.getParcelableExtra<Location>("changed")

                    val uniName =
findViewById<TextView>(R.id.university_caption)
                    val uniRating =
findViewById<TextView>(R.id.university_rating)
             …

                    changed?.let {
           when (it.id) {
                 1 -> {…}
                 2 -> {…}
                 3 -> {…}
                 4 -> {…}
                             }
                         }
                    }
                }
            }
        }
```

Gap 3: Mobile software development testing

To test the program, I used Espresso. The test function I implemented was as follows:

1. Click on an image in the MainActivity.
2. Change the name in the DetailActivity.
3. Press Back to return to the MainActivity
4. Check the name of the image in the MainActivity to see if it was changed correctly.

The goal was to check for all images available. This can lead to duplicated blocks of code, which is not recommended at this stage of the course. Therefore, I created a new class named TestLocation with 2 attributes: mainName and mainImage, which accordingly stored the ID to the image and the name for all images displayed in MainActivity. The needed objects are then created and stored in testLocationArray. I also created a testStringArray array for testing. After that, it only required a for loop to iterate throught the testLocationArray array and check for all images.

```
fun changedImageName() {
      …
         val testStringArray = arrayOf("Luong", "Trac", "Duc", "Anh")
         val testLocationArray = arrayOf(university, station, hall,
garden)

         fun miniTest(testLocation: TestLocation, testString: String) {
      …
              mainImage.perform(ViewActions.click())
              detailName.perform(ViewActions.clearText())
              detailName.perform(ViewActions.typeText(testString))
              closeSoftKeyboard()
              pressBack()
              mainName.check(matches(withText(testString)))
         }
         for (i in testLocationArray.indices)
miniTest(testLocationArray[i], testStringArray[i])
     }
```

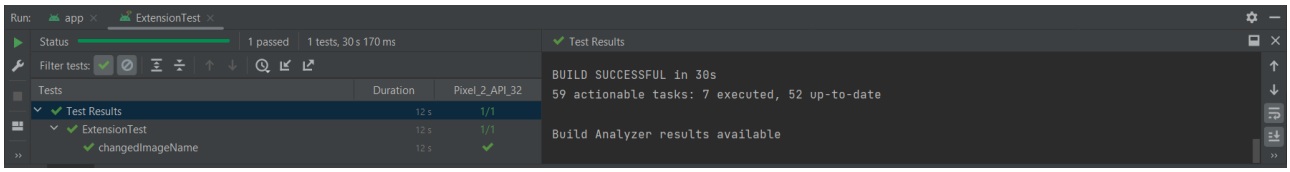

Figure 2: Test run successfully

Gap 4: Implementation of snackbar to display updated objects

When an image is clicked on and gets updated, a snackbar will appear at the bottom of the screen. This can be implemented easily by add the name of the changed object to the snackbar text string.

```
Snackbar.make(window.decorView.rootView, "${changed?.name} updated",
Snackbar.LENGTH_LONG).show()
```

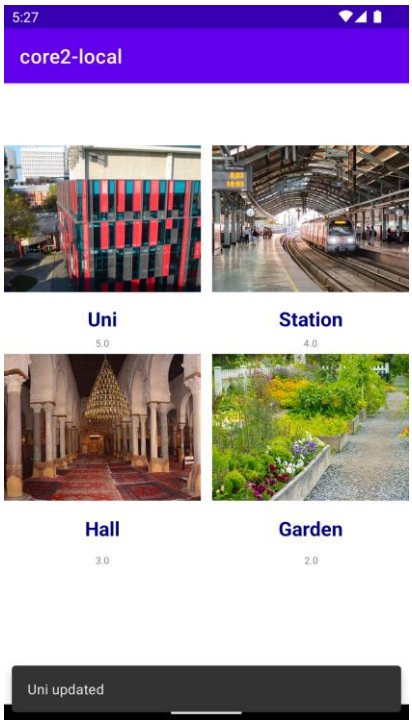

Figure 3: Snackbar appeared

Open Issues and Recommendations

1. **Error messages appear time**

At first, I was not sure if the error message should appear when the user changes the text or when they close the keyboard and press the Back button. After close inspections of the sample video I believe that it should be the latter. However due to being uncertain, I kept the piece of code for the first scenario to be commented instead of deleted.

```
val detailName = findViewById<EditText>(R.id.detail_name)
        /*detailName.doAfterTextChanged {
            detailName?.error = if
(detailName.text.toString().isNotEmpty()) null else "Name cannot be
empty"
        }*/
```

```
        val detailLocation =
findViewById<EditText>(R.id.detail_location)
        /*detailLocation.doAfterTextChanged {
            detailLocation?.error = if
(detailLocation.text.toString().isNotEmpty()) null else "Location cannot
be empty"
        }*/

        val detailDate = findViewById<EditText>(R.id.detail_date)
        /*detailDate.doAfterTextChanged {
            val regEx = "^(0[1-9]|[12]\\d|3[01])[- /.](0[1-9]|1[012])[-
/.](19|20)\\d{2}$"
            val matcherObj: Matcher =
Pattern.compile(regEx).matcher(detailDate.text)
            detailDate?.error = if (matcherObj.matches()) null else
"Date have to be in the right format: dd-mm-yyyy"
        }*/
```

2. **Complicated testing**

Instead of using 3 lines of code to clear the EditText view, type the text and close the keyboard, I could instead use 1 line of code only, which can be described as follows:

```
// good way
detailName.perform(replaceText("Uni"))
// my way - complicated and not recommended way
detailName.perform(ViewActions.clearText())
detailName.perform(ViewActions.typeText("Uni"))
closeSoftKeyboard()
```

3. **Snackbar**

The snackbar always appears when the user press the Back button from the DetailActivity to the MainActivity, even if they have not yet changed anything. I have not yet been able to sort this problem out.