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

Link to GitHub repository: https://github.com/aalexandros47/Software_Development_For_Mobile_Devices

Goals:

- accomplished every goal listed in the Task 2 core spike report.
- gained the ability to use intents to send data back and forth between two activities.
- handled user input efficiently by using validation and conditional checks, guaranteeing that the right error messages are shown.
- Espresso was used to implement UI and unit tests for comprehensive testing.

Tools and Resources Used

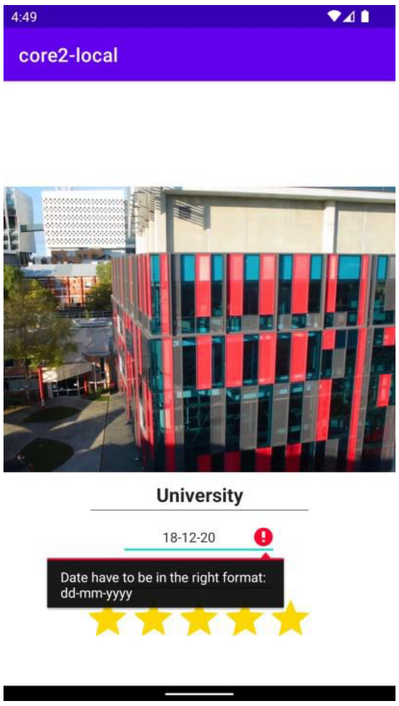
- Android Studio IDE
- Git and GitHub
- Kotlin programming language and XML files
- The course’s modules
- UI and unit testing: https://developer.android.com/training/testing/espresso/basics
- Passing data:
  https://developer.android.com/reference/kotlin/androidx/activity/result/contract/ActivityResultContracts

Knowledge Gaps and Solutions

Gap 1: EditText input validation

In order for the user to modify the data and have more contact with the application, we must convert the TextViews in DetailActivity to EditText for this task. Verifying user input is a crucial part of dealing with it in order to ensure that the format is accurate. In order to allocate the error widget based on the incorrectly formatted EditText field, I inserted a conditional block of code.

```
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"
```



Date have to be in the right format: dd-mm-yyyy

Figure 1: Error message

Gap 2: Communicate data through Intents and ActivityResultContracts

We must override the DetailActivity's onBackPressed function in order to transfer the data from the DetailActivity back to the MainActivity. Additionally, a fresh assignment of the current data in the views to the location object's attributes must be made. Next, we send a key and the location object back to the MainActivity using the intent's putExtra() function. Along with the intent, the value RESULT_OK must also be sent back.

Returning to the MainActivity, we reassign the TextViews to the newly received modified Parcelable object using ActivityResultContracts. After that, the attributes of the altered object are also applied based on which image or object was clicked. I gave the Location class a new attribute called id in order to determine which object needed to be modified. Each item has a unique id that is checked when we receive the Location object sent from the DetailActivity to the MainActivity through intent; this id won't change during the program.

```
// 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

I used Espresso to test the program. Here is the test function that I put into practice:

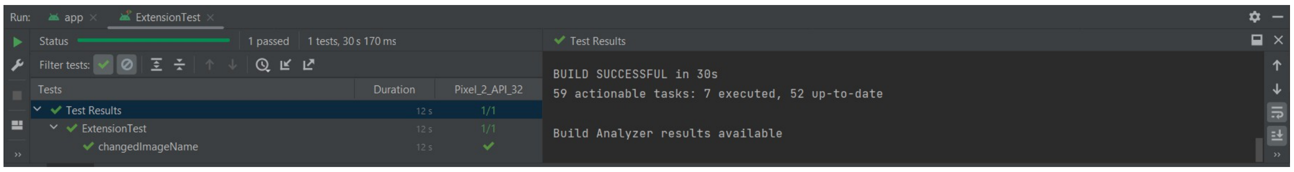
1. In the MainActivity, click on an image.
2. Modify the DetailActivity's name.
3. To go back to the Main Activity, press Back.
4. Verify that the image's name was appropriately modified in the MainActivity.

Checking for every photograph that was available was the aim. This may result in duplicate code blocks, which is not advised at this point in the course. In order to save the ID of the picture and the name of every image presented in MainActivity, I made a new class called TestLocation with two attributes: mainName and mainImage. After that, the necessary objects are produced and put into testLocationArray. For testing, I also made a testStringArray array. After that, all that was needed to go through the testLocationArray array and check for every image was a for loop.

```
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])
```

```
    }
```



Run: app ExtensionTest

Status 1 passed 1 tests, 30 s 170 ms

Filter tests:

Tests	Duration	Pixel_2_API_32
✓ Test Results	12 s	1/1
✓ ExtensionTest	12 s	1/1
✓ changedImageName	12 s	✓

✓ Test Results

```
BUILD SUCCESSFUL in 30s
59 actionable tasks: 7 executed, 52 up-to-date

Build Analyzer results available
```

Figure 2: Test run successfully

Gap 4: Implementation of snackbar to display updated objects

At the bottom of the screen, a snackbar will show up when you click on an image and it updates. Adding the name of the modified object to the snackbar text string makes this simple to accomplish.

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

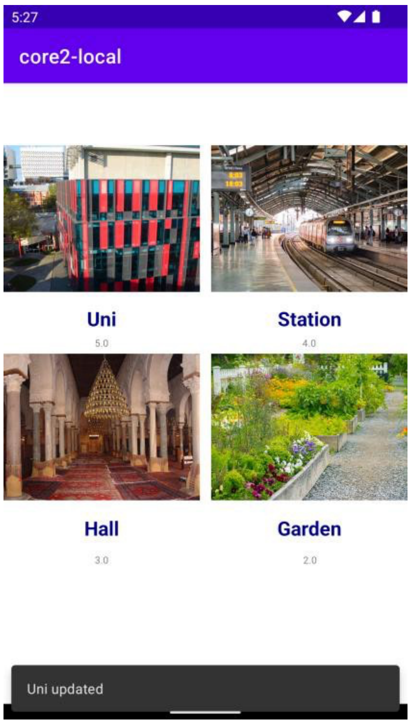


Figure 3: Snackbar appeared

Open Issues and Recommendations

1. **Error messages appear time**

Initially, I wasn't sure whether the error message should show up when the user edits text or when they hit the Back button while closing the keyboard. I think it should be the latter after closely examining the sample video. But since I wasn't sure, I decided to leave the first scenario's code commented rather than deleting it.

```
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

I could use just one line of code, which is as follows, to clear the EditText view, type the text, and then dismiss the keyboard rather than three lines of code:

```
// 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

Even if the user hasn't made any changes yet, the snackbar always shows up when they press the Back button to go from the DetailActivity to the MainActivity. I haven't been able to resolve this issue yet.