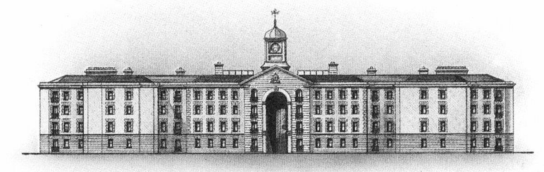


GRIFFITH COLLEGE DUBLIN

Assignment Cover Sheet

Student name: Auizio Diniz Neto

Student number: 3001825

Faculty: **Computing Science**

Course: Computer Science **Stage/year:** 4

Subject: Mobile Development [BSCH-MD/Dub/PT]

Study Mode: Full time _____ Part-time x

Lecturer Name: Dr. Abubakr Siddig

Assignment Title: Developing an Android App - Milestone 3 - final

No. of pages:

Disk included? Yes _____ No x

Additional Information: (ie. number of pieces submitted, size of assignment, A2, A3 etc)

Date due: 13/12/2022

Date submitted: 13/12/2022

Plagiarism disclaimer:

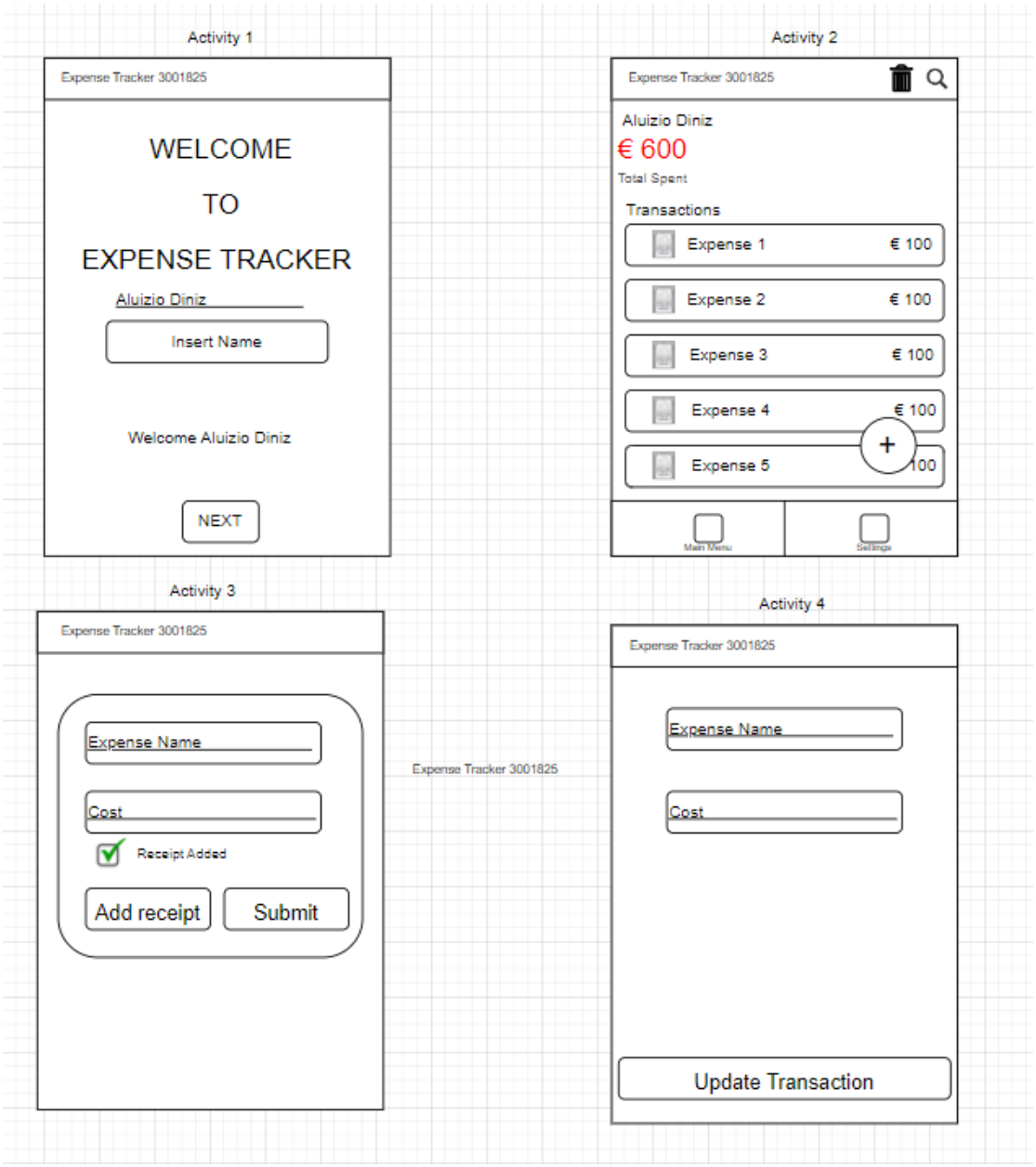
I understand that plagiarism is a serious offence and have read and understood the college policy on plagiarism. I also understand that I may receive a mark of zero if I have not identified and properly attributed sources which have been used, referred to, or have in any way influenced the preparation of this assignment, or if I have knowingly allowed others to plagiarise my work in this way.

I hereby certify that this assignment is my own work, based on my personal study and/or research, and that I have acknowledged all material and sources used in its preparation. I also certify that the assignment has not previously been submitted for assessment and that I have not copied in part or whole or otherwise plagiarised the work of anyone else, including other students.

Signed: _____Aluizio Diniz _____ Date: _____13/11/2022_____

Design

The main design of the project consists of 4 different activities that trade information among them. The design can be seen below:



- **Activity 1:** It consists of the initial screen where the user will be able to insert their name.

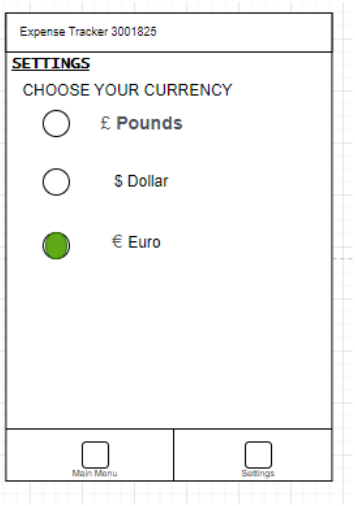
- **Activity 2:** This is the main screen to display all the necessary information to the user such as their total spent, all their expenses and an active button to take the user to input a new expense. It also has the bottom navigation bar which should take the user to the settings screen, and the main menu which prompts them back to activity 2 again. In addition to that there are also two

extra features added to the top bar, which are the search method and the delete database method.

- **Activity 3:** Once the user clicks on the plus icon(+) they can see a pop window where they can name their expense, its cost and add a receipt button(will trigger the camera). An activity can only be inserted into the database if it contains all 3 elements: label, amount and receipt picture (id will be automatically populated by code)

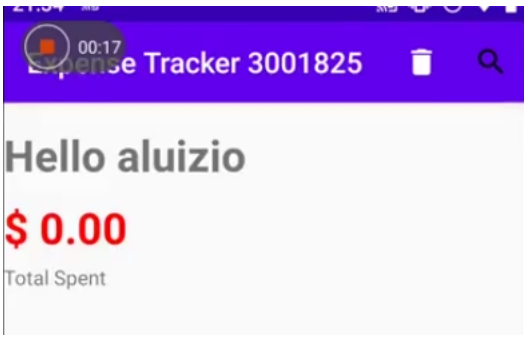
- **Activity 4:** it works in a very similar way to Activity 3, the only difference between them is that in activity 4 the user cannot change the receipt for the current transaction.

- **Settings (Extra):** This screen was not implemented, in fact this was supposed to be a fragment not an activity, where the user would be able to select the currency for their project. The total amount shown in the Activity 2 then would change it's bill sign according to the currency picked by the user. It's idealization design can be seen below:



Code Description

1. **First activity name**




```
fun callActivity(){
    val editText = findViewById<EditText>(R.id.name)
    val message = editText.text.toString()
    // the text that was input in the editText is stored inside of a va
    val intent = Intent( packageContext: this, SecondActivity::class.java).
        it.putExtra( name: "EXTRA_MESSAGE",message)
        startActivity(it)
    }
}

val editText = findViewById<EditText>(R.id.name)
val namee = editText.text.toString()
//function to call the next activity
val secondActivityButton : Button = findViewById(R.id.buttonNext)
secondActivityButton.setOnClickListener { it: View!
    callActivity()
}
```

In the first activity we have the name that is being input by the user being stored into a variable and being sent over to SecondActivity via the method putExtra.

putExtra is a very useful method to transfer data over to different activities, and it will be used more times during the project to transfer data over to the activity that is being called.

```
//first we need to initialize the name that has been passed from the main activity
    val name1 = intent.getStringExtra( name: "EXTRA_MESSAGE")
    //when the message is received I'm sending it to the textView
    val textView = findViewById<TextView>(R.id.welcomeUser).apply { text = "Hello "+ name1 }
```

In the second activity the message that was passed is inserted into the textView so the user can see their name up on screen on the top.

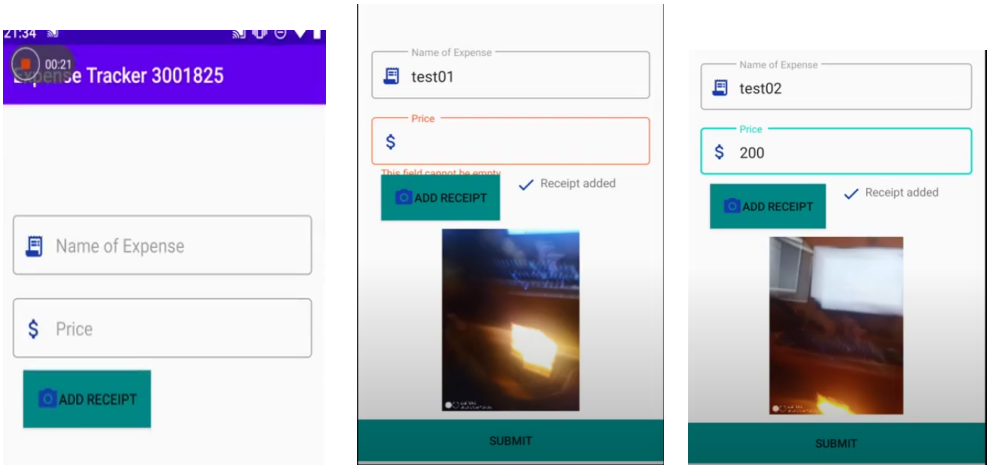
2. **Second activity to plus transaction activity**


```
val thirdActivityButton : Button = findViewById(R.id.addButton)
thirdActivityButton.setOnClickListener {
```

```
                                    val           Intent           =
Intent(this,plusTransactionActivity::class.java)
    startActivity(Intent)
}
```

Similar to what happened for the Main Activity to the Second one is then used to prompt the user to the activity plus transaction.

3. Plus Transaction Activity



Once the button add receipt is pressed it triggers the function below

```
    addReceiptButton.setOnClickListener {  it: View!
    val takePictureIntent = Intent(MediaStore.ACTION_IMAGE_CAPTURE)

    if (takePictureIntent.resolveActivity(this.packageManager) != null) {
        startActivityForResult(takePictureIntent, REQUEST_CODE)

    } else {
        Toast.makeText( context: this,  text: "Unable to open camera", Toast.LENGTH_SHORT).show()
    }
}
```

After the image has been captured we store it as a bitmap and we can change the visibility of the button and the receipt sign. The button will remain invisible until an image has been added to force the user to insert an image.

```
override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {
    if (requestCode == REQUEST_CODE && resultCode == Activity.RESULT_OK) {
        receipt = data?.extras?.get("data") as Bitmap
        imageView.setImageBitmap(receipt)
        receiptSign.setVisibility(View.VISIBLE)
        imageView.setVisibility(View.VISIBLE)
        submitTransactionButton.setVisibility(View.VISIBLE)
    } else {
        super.onActivityResult(requestCode, resultCode, data)
    }
}
```

I have also set a few constraints to force the user to input a name and a cost, and the user should not be able to proceed to submit data until these two values have been input. To input the data we use the function SubmitData.

```
submitTransactionButton.setOnClickListener{ it: View!

    val expense = expenseInput.text.toString()
    val cost = costInput.text.toString().toDoubleOrNull()



    if(expense.isEmpty())
        expenseLayout.error = "This field cannot be empty"

    if(cost == null)
        costLayout.error = "This field cannot be empty"
    else{
        val transaction = Transaction( id: 0,expense, cost, receipt)
        submitData(transaction)
        Toast.makeText(applicationContext,
             text: "Transaction added", Toast.LENGTH_LONG).show()
    }
```

If constraints are met we can submit data using the code below. The notice inserts an object of type transaction into the database using the Data Access Object(DAO) interface class.

```
private fun submitData(transaction: Transaction){
    val  database= Room.databaseBuilder( context: this,
        AppDatabase::class.java,
        name: "transactions").build()

    GlobalScope.launch { this: CoroutineScope
        database.userDao().insertAll(transaction)
        finish()
    }
```

4. Transaction class and Transaction Dao

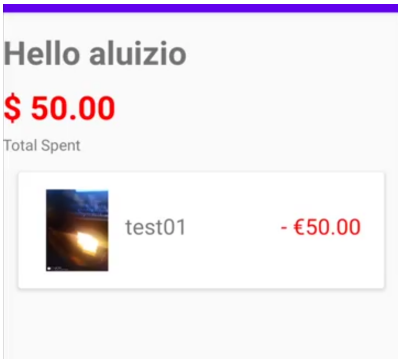
```
@Dao
interface transactionDao {
    @Query("SELECT * from transactions")
    fun getAll(): List<Transaction>

    @Insert
    fun insertAll(vararg transaction: Transaction)

    @Delete
    fun delete(transaction: Transaction)

    @Query("DELETE FROM transactions")
     fun deleteAll()


    @Update
    fun update(vararg transaction: Transaction)
}
```

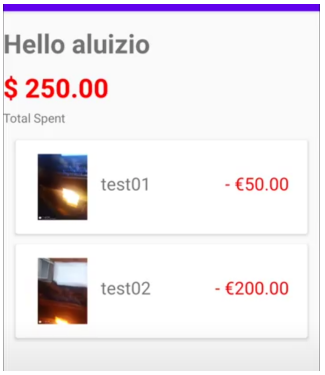
```
@Entity(tableName = "transactions")
data class Transaction(
    @PrimaryKey(autoGenerate = true) val id: Int,
    val label: String,
    val amount: Double,
    val receipt: Bitmap? = null
    ) {

}
```

Here I have two classes that will be used constantly, the first class is the Transaction which describes a new object of type transactions that take an id(which will be auto-incremented and this one is the primary key - unique identifier), a label(a name), an amount(double amount) and a receipt which will be type Bitmap because we are converting our photo to bitmap before storing it.

We also have the DAO which defines all the functions and their respective query to insert the data(which always receives a transaction object) into the database (table “transactions”).

5. Second Activity -> total spent and card view





When the data (transaction recently added) gets added it is now shown in the recyclerView in a card format, where all the data that was input can be seen. Also notice that the total spent by the user has been updated to take the amount value of the transaction.

This is possible thanks to the function below that takes a list of transactions(with all the transactions in the database) and sums all the values in the amount field.

```
private lateinit var allTransaction: List<Transaction>
```

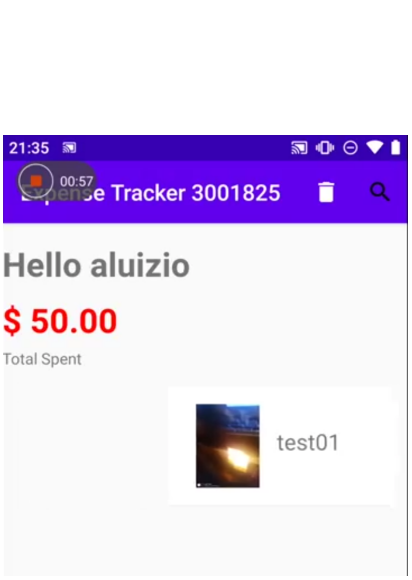
```
private fun updateTotalSpent(){
    val total = allTransaction.map { it.amount }.sum()

    val totalSpent = findViewById<TextView>(R.id.totalSpent)
    totalSpent.text = "$ %.2f".format(total)
}
```

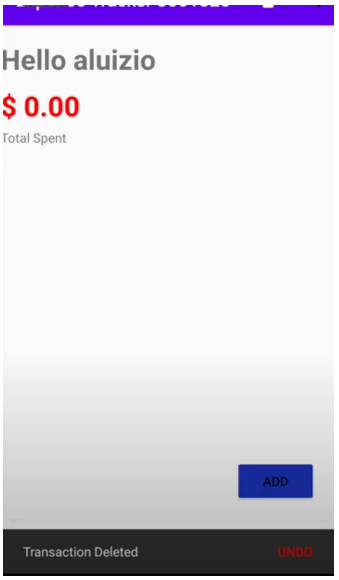
Now that we have our function that describes all the work we just need to display the changes to the user and constantly keep our list of transactions (allTransaction list) updated.

```
private fun fetchAll(){
GlobalScope.launch {  this: CoroutineScope
    allTransaction = database.userDao().getAll()
    runOnUiThread {
        updateTotalSpent()
        transactionAdapter.updater(allTransaction)
    }
}
```

```
fun updater(transactions: List<Transaction>){
    this.transactions = transactions
    notifyDataSetChanged()
}
```

6. Second activity -> Delete transaction and Undo delete





The ability to individually delete transactions is only possible when swiping the card to the left. When it gets swiped it deletes the activity by calling the function below.

```
override fun onSwiped(viewHolder: RecyclerView.ViewHolder, direction: Int) {
    deleteTransaction(allTransaction[viewHolder.adapterPosition])
}
```

Now it's important to understand the behavior of the deleteTransaction function that is being triggered, notice that this function receives a parameter of type Transaction that will be exactly the position of the item that got swiped. Also notice that I'm storing the list of transactions that is about to be deleted in a variable so I can return it later.

```
//this method is used to delete transactions
    private fun deleteTransaction(transaction: Transaction){
        //first I stored the current transaction in a variable to be able to undo
        deletedTransaction = transaction
        oldTransaction = allTransaction

        //we use the delete from userDao to delete the transaction that is passed to the method
        GlobalScope.launch {  this: CoroutineScope
            database.userDao().delete(transaction)
            //once the transaction is deleted we need to refresh the total
            allTransaction = allTransaction.filter { it.id != transaction.id}
            runOnUiThread {
                //we use the update this function to refresh the total and refresh the total spent number on the top
                updateTotalSpent()
                transactionAdapter.updater(allTransaction)
                //to be able to undo de the deletion we create the bar
                createBar()
            }
        }
    }
```

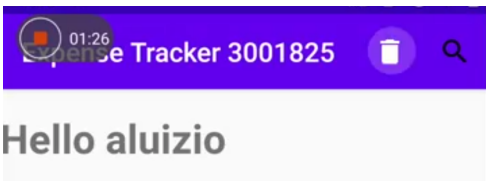
At the end we call the function createBar() that will prompt a bar with an undo button that will return the old list of transactions to the database if pressed.

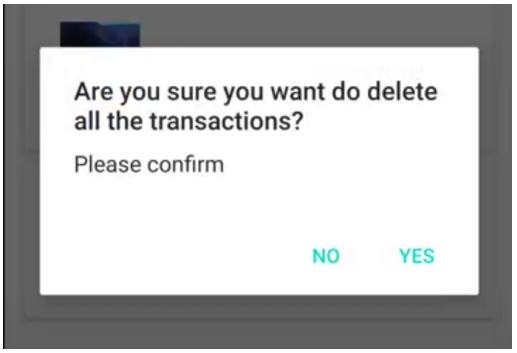
```
private fun createBar(){
    val view = findViewById<View>(R.id.coordinatorLayout)
    //the bar used the function snackbar that creates this little bar in the very bottom of the screen
    val bar = Snackbar.make(view, text: "Transaction Deleted",Snackbar.LENGTH_LONG)
    bar.setAction( text: "Undo"){  it: View!
        //if the user happens to click on the bar action button we created the undoDelete function gets triggered
        undoDelete()
    }

        .setActionTextColor(ContextCompat.getColor( context: this,R.color.red))
        .show()

}
```

And at the end of the undo delete I have to again call the function to update total spent so the new amount values can be added up and refresh the number in the top of the activity.

```
private fun undoDelete(){
    GlobalScope.launch {  this: CoroutineScope
        database.userDao().insertAll(deletedTransaction)
        allTransaction = oldTransaction
        runOnUiThread {
            transactionAdapter.updater(allTransaction)
            updateTotalSpent()
        }
    }
}
```

7. Delete all transactions in the database and custom menu





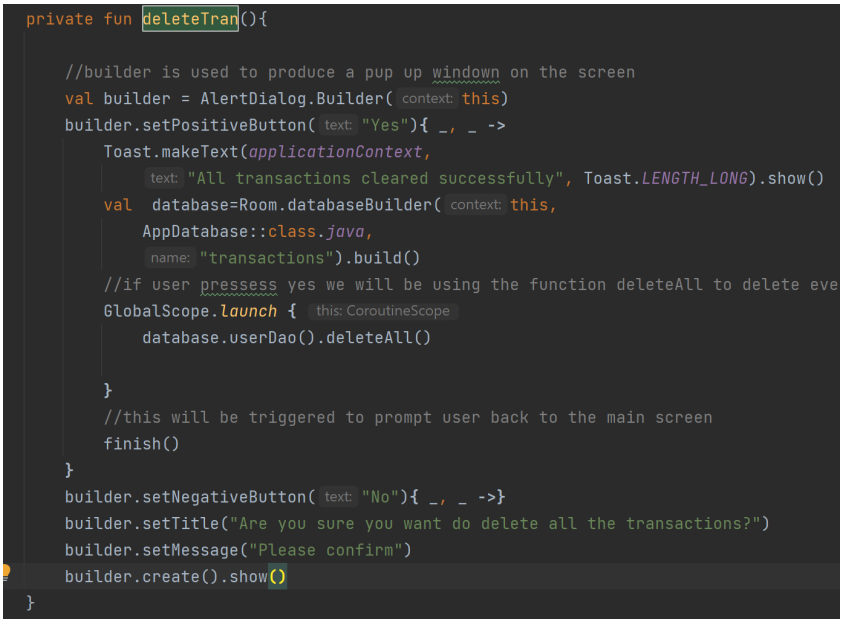
In order to add the top bar I have created a custom menu with two different items, one of them will be the trash icon that triggers the deleteTran function.

In order to add the custom menu to the current activity, I override the onCreateOptionsMenu with our new menu.

```
<item android:id="@+id/menu_delete"
    android:title="Delete"
    android:icon="@drawable/ic_binwhite"
```

```
//function to trigger our custom menu
    override fun onCreateOptionsMenu(menu: Menu?): Boolean {
        menuInflater.inflate(R.menu.delete_menu,menu)
        return true
    }
```

The DeleteTran() function gets triggered when the button on the top bar gets pressed, and it builds a pop up screen on the top of the current activity using a variable of type Builder. All the variables for this variable are set using the setter functions.



```
private fun deleteTran(){

    //builder is used to produce a pup up windown on the screen
    val builder = AlertDialog.Builder( context: this)
    builder.setPositiveButton( text: "Yes"){ _, _ ->
        Toast.makeText(applicationContext,
             text: "All transactions cleared successfully", Toast.LENGTH_LONG).show()
        val  database=Room.databaseBuilder( context: this,
            AppDatabase::class.java,
            name: "transactions").build()
        //if user pressess yes we will be using the function deleteAll to delete eve
        GlobalScope.launch {  this: CoroutineScope
            database.userDao().deleteAll()

        }
        //this will be triggered to prompt user back to the main screen
        finish()
    }
    builder.setNegativeButton( text: "No"){ _, _ ->}
    builder.setTitle("Are you sure you want do delete all the transactions?")
    builder.setMessage("Please confirm")
    builder.create().show()
}
```

If the yes button gets pressed the deleteAll function from the DAO gets called, deleting everything from the database and then the user is taken to the main screen by the function finish().

Difficulties

During this project I encountered many difficulties but I handled them really well being able to implement all the functionalities that were described in the project orientation and in my original project.

However there are two functionalities which I tried to implement in order to make the project more user friendly which were not successfully implemented. These functionalities are:

- **Search:** The search icon which was inserted in the top menu bar is not functioning correctly, the button gets triggered but the main idea(which was filtering transactions by their label) was not successfully implemented.

- **Bottom Navigation bar:** The bottom navigation bar is successfully implemented(by that I mean the different fragments are being pulled) however they are being generated on top of the second activity that has a lot of information. A workaround for this would be leaving the second activity blank and recreating the second activity in the home fragment.

GitHub Link

https://github.com/aluiziod/Kotlinapp.git