**CMPS 312 Mobile App Development**

**Banking App**

**Lab Assignment 4**

**Deadline - Tuesday, November 10, 2020**

**Objective**

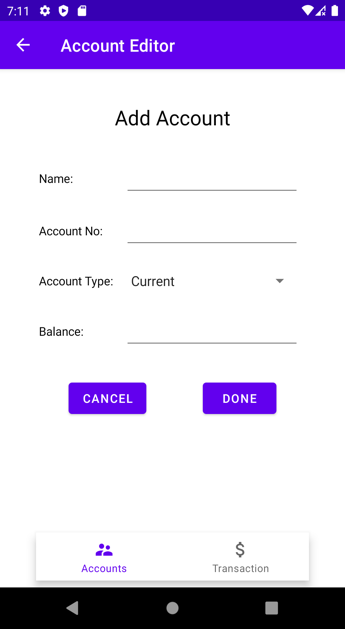
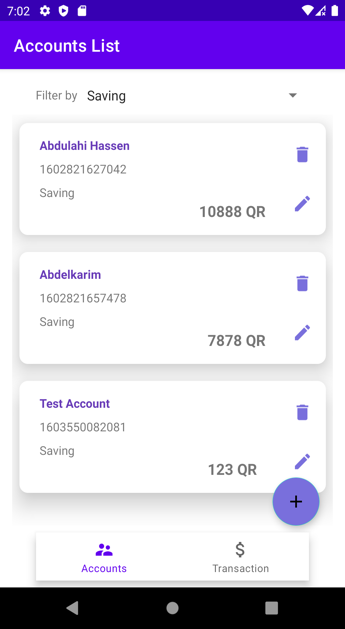
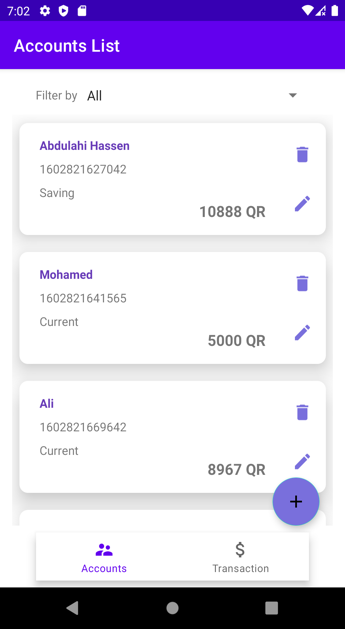
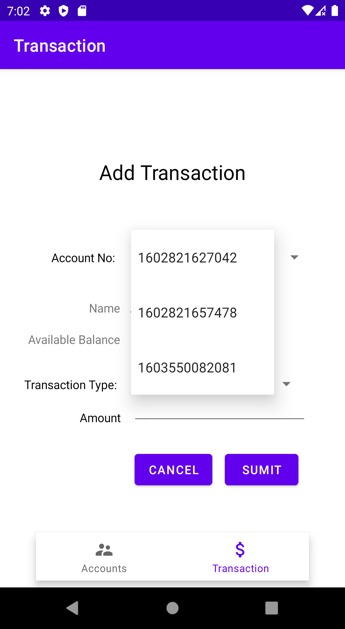
The objective of this assignment is for you to practice Room database library to write, read and query records from the SQLite database.

**Preparation**

1. Sync the Lab GitHub repo and copy the **Assignment/Assignment4** folder into your repository.
2. Open the project ***Bank App*** in Android Studio. The project has the baseline code implementation using Retrofit.
3. The majority of the code that is related to the ViewModel, fragments are implemented for you. However, the code related to the transaction is partially implemented. So. it is your task to complete any missing code.

**Overview**

In this assignment, your task is to replace the given bassline solution of ***Bank App’s*** backend which currently uses retrofit with ***Room Database*** and achieve similar functionalities as shown in the following web app <https://employee-bank-app.herokuapp.com>



**Implementation Instructions**

1. Create two Entity classes called [**Account and Transaction]** with proper Room Database annotations. Make sure you add the cascade delete/update and also enforce integrity checks using the foreign keys.

Diagram

Description automatically generated

|  |  |  |  |
| --- | --- | --- | --- |
| id | type | amount | accountNo |
| 1 | Withdraw | 200 | 12343 |
| 2 | Deposit | 500 | 12343 |
| 3 | Deposit | 200 | 22222 |

|  |  |  |  |
| --- | --- | --- | --- |
| accountNo | Name | accountType | Balance |
| 12343 | Ali | Saving | 1000 |
| 22222 | Omar | Current | 500 |

1. Create Data Access Objects (DAO) Interface and the Database classes.
2. Inside the DAO Interface create all the necessary functions and annotations that allow the user to,
   1. Get, Add, Update and Delete a specific Account
   2. Query Accounts by specific type [All, Current or Saving] (check the demo)
   3. Add new transaction
3. Implement the list account fragment and show all the list of accounts from the accounts table inside the database as shown below.

**Graphical user interface, application

Description automatically generated**

1. Implement the list account by type. When the user selects from the drop the filter type you should query your database and display the accounts according to the selected filter. Please check the shared application demo to further understand this requirement.

**Important** Do, not implement the filtering of accounts using the **kotlin .filter function**. You should write a database query that filters accounts by type and returns a LiveData.

Graphical user interface, text, application, chat or text message

Description automatically generated

|  |  |  |
| --- | --- | --- |
| **All Accounts** | **Filtered by Current** | **Filtered by Saving** |
| **Graphical user interface, application  Description automatically generated** | **Graphical user interface, application, chat or text message  Description automatically generated** | **Graphical user interface, application  Description automatically generated** |

1. Implement the delete account. When the user clicks on the delete button you should delete the specific account from the database and update the list view.

Graphical user interface, text, application

Description automatically generated

1. Implement the **add and edit** account functionalities shown below. When a user adds you should create a new account. If the user updates, you should update that specific account in your database.

Graphical user interface, application, Teams

Description automatically generated**Graphical user interface, application, Teams

Description automatically generated**Graphical user interface, application

Description automatically generated

1. **Graphical user interface, application, Teams

   Description automatically generatedGraphical user interface, application

   Description automatically generated**Allow users **to add transactions** that are either a **withdraw** or **a deposit** of specific accounts.

When a user adds a new **transaction** you should update the related account in the account table. For example

If an account no **12343**, withdraws 500 riyals then, you should first add this new transaction to the **Transaction** table. Second, you should **update the balance** of the specific account. See the below scenario to further understand this requirement.

**Before Withdraw (account table)**

|  |  |  |  |
| --- | --- | --- | --- |
| accountNo | Name | accountType | Balance |
| 12343 | Ali | Saving | 1000 |
| 22222 | Omar | Current | 500 |

**After Withdraw** **(transaction tables)**

[transactions tables] [accounts tables]

|  |  |  |  |
| --- | --- | --- | --- |
| id | type | amount | accountNo |
| 4 | Withdraw | 500 | 12343 |

|  |  |  |  |
| --- | --- | --- | --- |
| accountNo | Name | accountType | Balance |
| 12343 | Ali | Saving | 500 |
| 22222 | Omar | Current | 500 |

**Important** It is expected that each student to have a **unique app** that accomplishes the tasks. You are free to use any theme/font/colour/icons that you would like to use. Also, discussing with your colleagues is allowed, however, please avoid sharing your code. If any plagiarism is detected then both parties will get zero.

Submit the testing sheet as well as the code under “**your\_repository/assignments/assignment4**”