

CMPS 312 Mobile Application Development

Lab 8 –Model-View-ViewModel (MVVM) Architecture and Navigation

Review Lab

In this Lab, you will create a simple CRUD application that will help you understand the MVVM model architecture. You will be required to implement a simple student management app that allows,

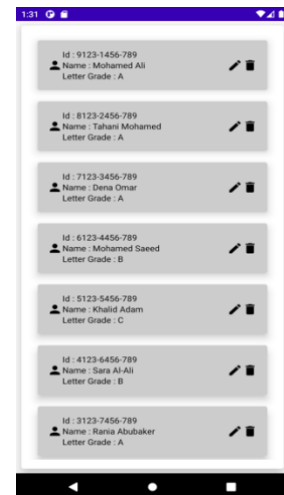
1. Display all students information as a list
2. Display a specific student's complete information on a separate screen
3. Edit/Update a particular student's information
4. Delete a student information from the Screen

Student Management App

Task 1- StudentList: You are given the "students.json" file. So, your first task is to implement the first Screen, which lists all the students on the Screen. Then, you should use LazyColumn to display the students. Also, you need each row to be inside a card and have the necessary buttons such as edit and delete.

Implementation Details

1. Create a new project and name it **Student Management App**
2. Create assets folder and add the **students.json** file
3. Add the necessary dependencies that allow you to serialize JSON files [Refer to previous labs 5,6,7]
4. Create a data class called Student that is serializable inside a package called **model**
5. Create a class called **StudentRepository** inside a new package named repository
6. Implement the **StudentRepository** class. This class should have one single method called **getStudents()**. This method should return the list of students inside the **students.json** file.
7. Add the necessary dependencies for the viewmodel.
8. Create a new package named viewmodel. Inside the viewmodel package, Implement the **StudentViewModel**, which extend the **AndroidViewModel**
9. The StudentViewModel should have the following
 - a. an object named **students**, which is MutableList. You should populate the data of this students list by using the StudentRepository class's **getStudents()**
 - b. a method that allows you to add a new Student. The method should receive a student object
 - c. a method that allows you to remove a student from the list of students. This method will receive a studentId as a parameter
 - d. a method that will enable you to edit a student. The method should receive a studentId and the updated student object
10. Create a new package called view
11. Create a file named **StudentList** that contains **StudentList composable** and **StudentCard composable**. This composable should get the students list and display them in a LazyColumn, as

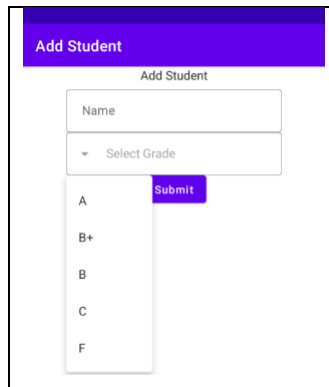


shown in Figure1. Add the necessary entities, repository and ViewModel class object you need to implement this task.

12. Make sure you add **edit** and **delete** icons to each student row card.

13. Test the **StudentList** composable separately by creating a preview composable

Task 2-Add Student. You need to implement the Screen that allows you to add a new student. The design of the Screen is shown below.

A screenshot of a mobile app form titled "Add Student". The form has a purple header bar with the title. Below the header, there is a text input field for "Name", a dropdown menu for "Select Grade" with options A, B+, B, C, and F, and a purple "Submit" button.

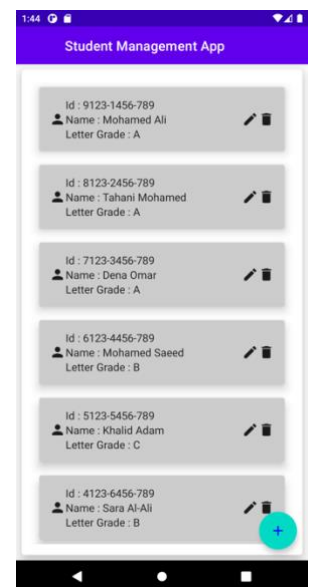
Implementation Details

1. Create a new composable named **StudentScreen**.
2. Get an instance of the **StudentViewModel**
3. Implement a similar design as the figure shown above.
4. You can hardcode the grade letters to create the dropdown menu
5. When the user presses on the Submit button, call the **StudentViewModel's object** and add the Student to the list
6. Test the composable separately by creating a preview composable

Task 3-Navigation. Before moving to the edit and update, you need to implement the navigation between the list and add composable. So, when the user clicks on the Add button, you should navigate them to the **StudentScreen**, and you should allow the user to add a new student.

Implementation Details

1. Created a **Screen** sealed class and add the following two objects **StudentList** and **StudentScreen**. The Screen class should be like the one you created in Lab 6 and 7.
2. Add the necessary dependences for the navigation
3. Create a new file called **AppNavHost** that implements the NavHost composable. You should add the necessary code for the NavHost. You can refer to previous weeks labs to understand how the NavHost is implemented.
4. Create **MainScreen** that has a Scaffold composable. The Scaffold should display the floating action button, the **AppNavHost** and the top app bar with the back navigation.
5. Test your code by injecting the MainScreen composable into your MainActivity. You can delete the default greetings composable and replace it with MainScreen.



Task 3-Delete Student. When the user clicks on the delete button, you should delete the information of the selected Student from the list of students

Implementation Details

1. Open the StudentList file
2. Add a click listener to the delete icon in the row
3. When the user clicks on the delete icon call the deleteStudent function inside the StudentViewModel
4. Test your code

Task 4. Update Student When the user click the edit icon, then load the selected Student's details into the Student Screen to allow the user to update the student details. Upon clicking submit the app should save the student details and navigate back to the StudentList screen.

Implementation Details

1. Open the StudentList file
2. Add a click listener to the edit icon in the row
3. When the user clicks on the edit icon, navigate the user to the Student Screen and indicate that you are in edit mode. In addition, you can pass a simple flag to the Student Screen indicating that you are in an edit mode.
4. When the user submits, try to call the check if you are in edit mode, then call the updateStudent function inside the StudentViewModel
5. Test your code