Database of Bus Stops

ICT2105 Mobile Application Development Spring 2020

Bus Stop Database

Fork the repo **ict2105-quiz02-2020** and inspect the code within the project.

The goal is to create an app using Android Architecture Components with a SQLite relational database that saves all the bus stop data and display them within a list in the application.

Hitting the Floating Action Button on the bottom right of the screen will open another separate Activity that will provide an screen to input the necessary fields for the bus stop information to be saved to database. Returning to the main screen after adding the item will refresh and display the newly added bus stop within the RecyclerView.

All screens should survive rotation and be refreshed with the latest data in the database.

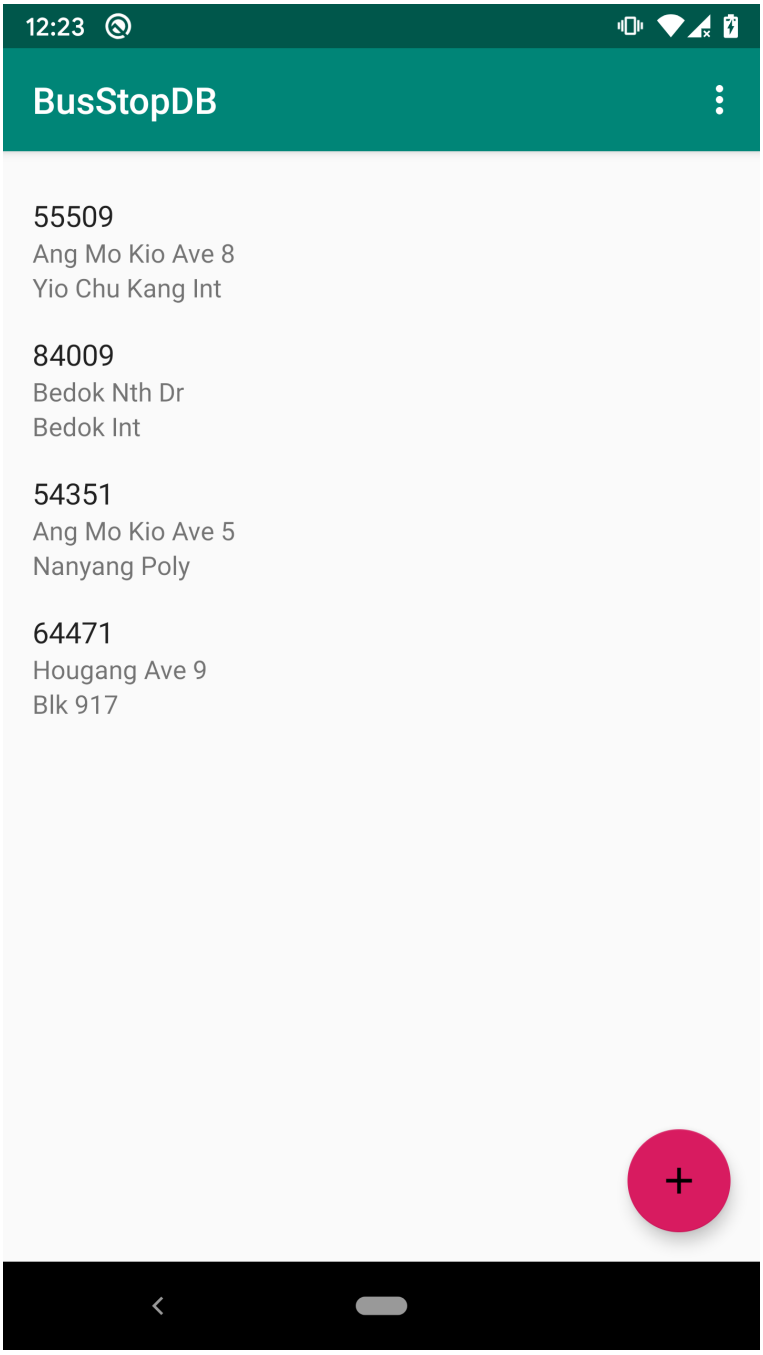
Implement the main screen

Design the layout and logic of the `MainActivity` similar to the given screenshot. Implement the RecyclerView on the MainActivity to display the bus stop data, and the floating action button. Each RecyclerView item should be able to display 3 rows of text that correspond to the columns that are retrieved from the database.

Note: All bus stop data should be saved and loaded from a SQLite Relational Database, using the Room architecture component. Refer to the PDFs in the root folder for example bus stop names and bus routes.

IMPORTANT:

- **Ensure the RecyclerView id is named recyclerViewBusStops within the XML**
- **The 3 TextView fields ids in the RecyclerView row XML should be named textViewBusStopCode, textViewRoadName and textViewBusStopDesc**
- **The floating action button id should be named fabAdd in the XML**



MainActivity (contains a custom RecyclerView)

Hints: A custom implemented RecyclerView item will be necessary for the display of 3 rows of text. ViewModel might be necessary to survive rotation and keep the list updated. Information on implementing the floating action button is here: https://developer.android.com/guide/topics/ui/floating-action-button

Implement the database

Implement the Room SQLite database to store the following information about an individual bus stop: **the id, bus stop code, road name,** and the **bus stop description:**

- ID: <primary key autoincrement>
- Bus Stop Code: 55509
- Road Name: Ang Mo Kio Ave 8
- Bus Stop Description: Yio Chu Kang Int

Implement the logic for the database access itself, and also to add and retrieve from this Room SQLite database with the necessary columns.

Enter a new bus stop

Create a new activity called `AddBusStopActivity` that will provide the text fields to add the necessary item into the database. Returning to the main screen will display the newly added bus stop within the database.

The “Add” button adds the bus stop to the database and closes the current activity and returns to the main screen. The “Clear” button clears all the fields. Hitting the “Back” arrow on the navigation bar simply returns to the main screen.

Do these simple checks on this Activity to prevent invalid entry of the fields:

- Check for **empty strings** being entered for the bus stop code, road name and bus stop description
- Check that there are **only digits** in the bus stop code

The checks should be done within a method with the following signature:

```
fun isInputValid(): Boolean {

}
```

Display a “Toast” message if there are invalid entries in the text fields. The toast message should EXACTLY say “Invalid input”

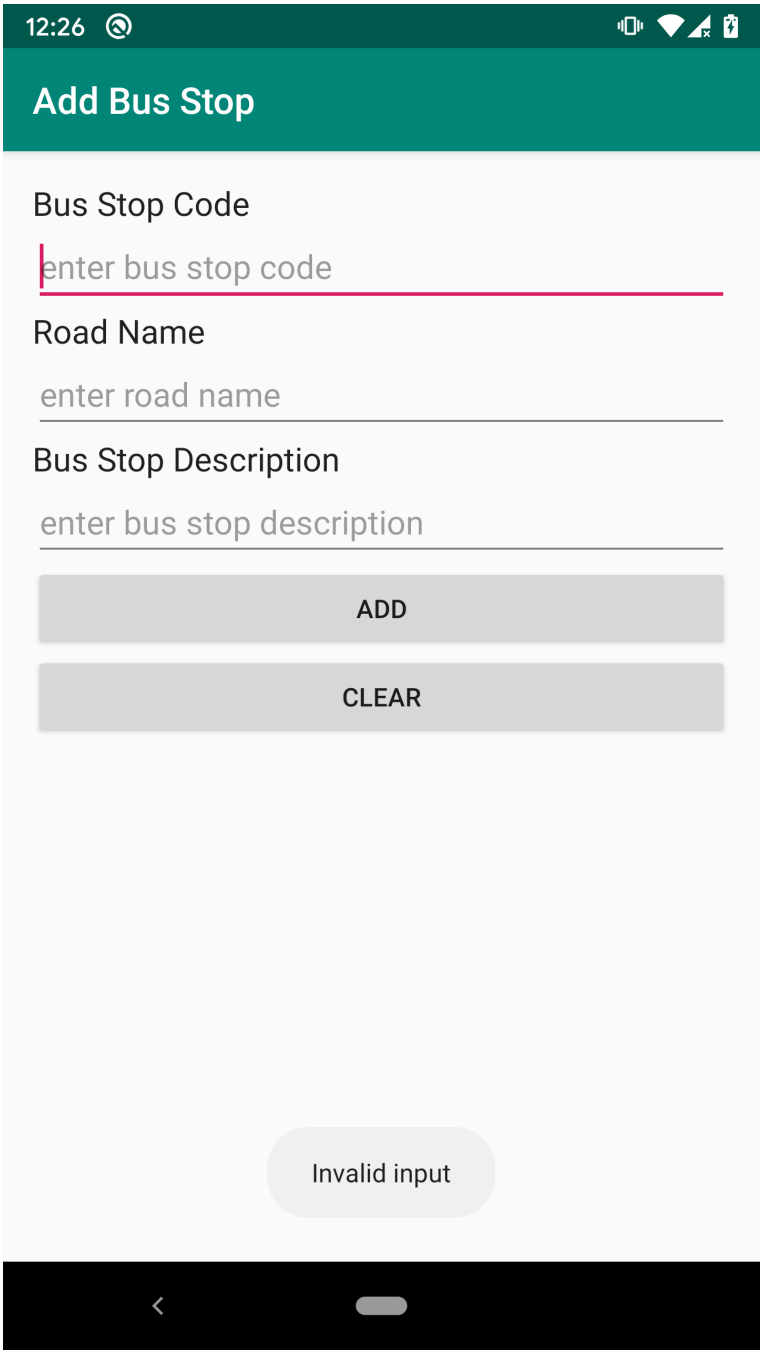
IMPORTANT:

- **Ensure that the Activity class name is AddBusStopActivity**
- **Ensure there are no spelling errors on the buttons and text views**
- **The three field ids in the XML should be named: editTextBusStopCode, editTextRoadName and editTextBusStopDesc**
- **Toast message should say “Invalid input”**

Save last entered values

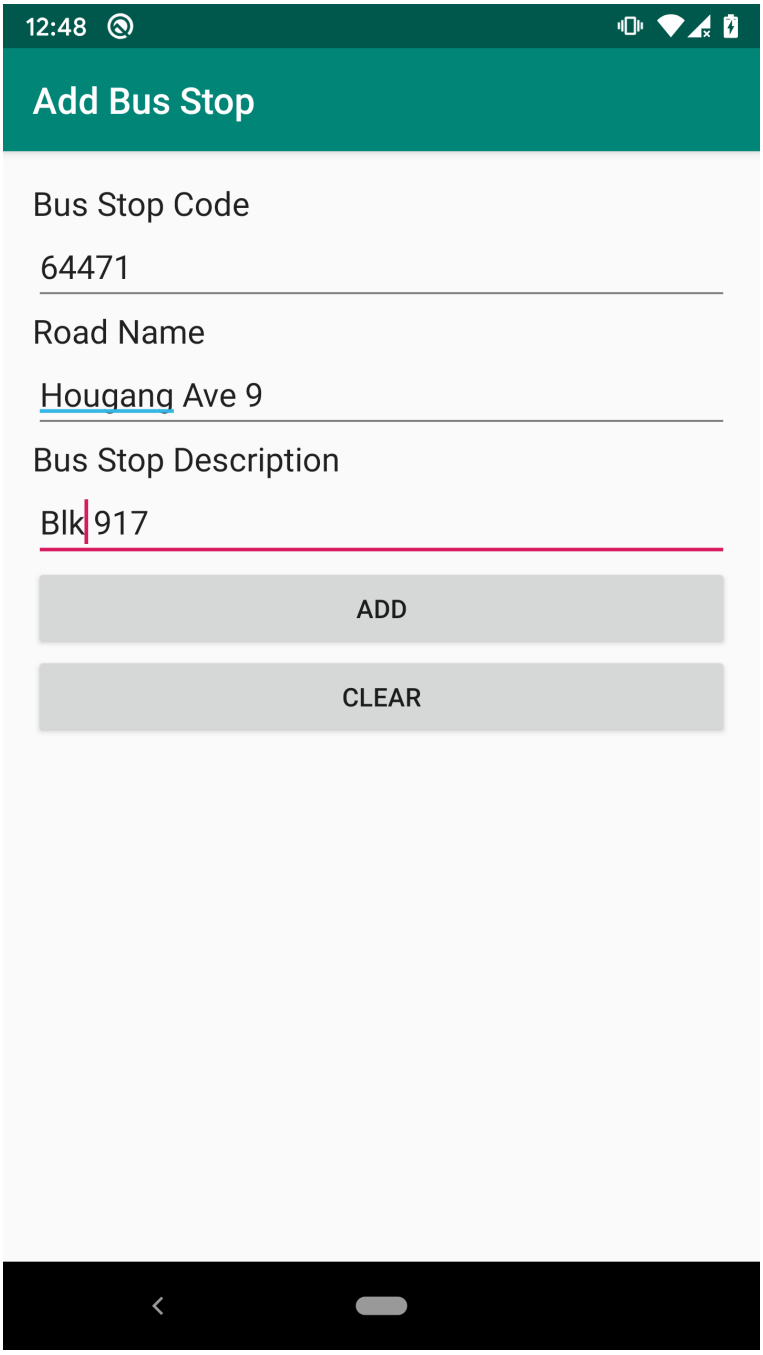
If the `AddBusStopActivity` is closed and reopened, or if the app process is killed and restarted, the last entered and added data of the bus stop code, road name and bus stop description should be reloaded and displayed in the `AddBusStopActivity`.

Implement the logic to save and reload the data that was last entered on the text fields on the screen. When the “Add” button is pressed, any valid values should be saved to shared preferences. (And also inserted into the Room database as implemented earlier). Do not save to shared preferences if the values are invalid.



AddBusStopActivity (empty fields, with toast)

IMPORTANT: Activity should be named AddBusStopActivity



AddBusStopActivity (with fields filled in)

Lab Quiz 2

1. Fork the repo **ict2105-quiz02-2020.**
2. Design the layouts of each of the screens similar to the given screenshots.
3. Implement the logic for each of the Activities.
4. Implement the missing tags within `AndroidManifest.xml` to launch the correct Activities.
5. Implement the logic and classes to save the bus stop data within a Room SQLite relational database.
6. Retrieve and display the bus stop data from the Room database onto the MainActivity within a RecyclerView with list items that display 3 rows of text.
7. Implement the logic to add a new bus stop to the Room database via the AddBusStopActivity.
8. Implement the following simple logic and validity checking before adding a new Bus stop, with the method called `isInputValid()`
   - Check for **empty strings** entered for the bus stop code, road name and bus stop description
   - Check that there are **digits only** in the bus stop code
9. Display a "Toast" message saying "Invalid input" if there are invalid entries in the text fields.
10. Save the bus stop code, road name and bus stop description to shared preferences, and reload and display in AddBusStopActivity when reopened
11. Retrieve and display the updated bus stop data from the database onto the MainActivity in the RecyclerView **after** adding a bus stop.
12. Remember to comment and indent the code, and implement it in a modular fashion, using different methods or classes as appropriate. Ensure it conforms to Kotlin coding conventions.
13. Commit and **push** all changes to your forked repository **ict2105-quiz02-2020.**

IMPORTANT: Do not change the activity names, method names, method signatures or package name. Please ensure all spelling and ids are correct.

Grading Criteria

1. MainActivity layout and UI elements are similar to screenshot (1 mark)
2. AddBusStopActivity layout and UI elements are similar to screenshot (1 mark)
3. Open AddBusStopActivity when floating action button is pressed (1 mark)
4. Logic on the clear button to clear the fields (1 mark)
5. Check for **zero length** strings as invalid input (1 mark)
6. Check for **only digits** within the bus stop code (1 mark)
7. Toast if the input is invalid (1 mark)
8. Successfully add new bus stop to database (1 mark)
9. Save the bus stop code, road name and bus stop description to shared preferences, and reload and display in the AddBusStopActivity when reopened (1 mark)
10. Add to database and close AddBusStopActivity, return to MainActivity (1 mark)
11. Refresh the RecyclerView with new record from database (1 mark)
12. MainActivity and AddBusStopActivity survive screen rotation and retain values (1 mark)
13. Modular code design and good naming conventions for methods and variables (1 mark)
14. Well-commented code (1 mark)

END OF DOCUMENT