ChatBot

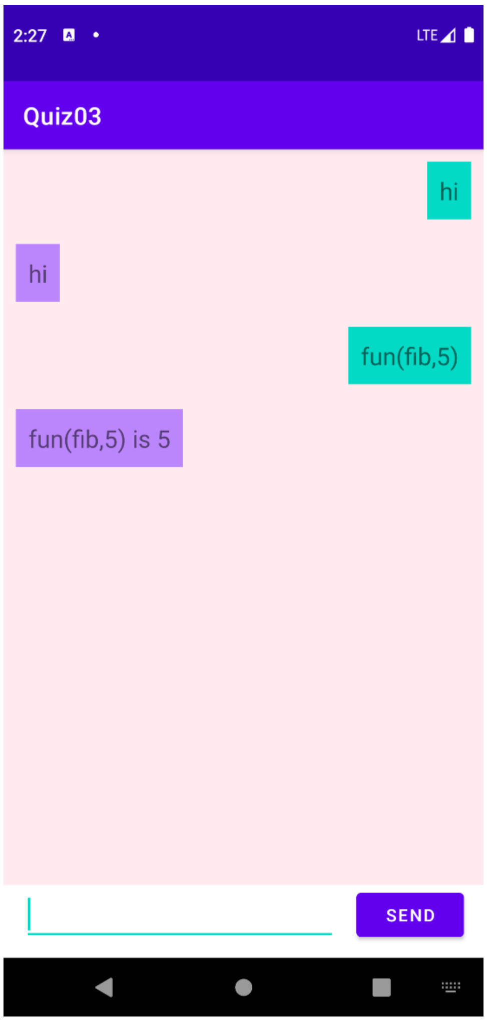
CSC2007 Mobile Application Development Spring 2023

Fork the repo **csc2007-quiz03-2023**.

A chatbot is software that simulates human-like conversations with users via chat. In this quiz, you are going to implement a chabot with Room database to store all the chats and provide responses for computing a simple function.

Design the UI

This app has only one activity: MainActivity. Design the layout of the screen for the MainActivity to be similar to the following. The views need to be visible after rotation.



- A **RecyclerView** has to be used to display the chats. (id: **recyclerView**).
    - The messages on the **right** are the inputs sent by the users and on the **left** are the responses from the ChatBot.
    - You may need to define two xmls for these two types of items/messages, e.g., left_msg_item.xml and right_msg_item.xml.
    - You can override the function "override fun getItemViewType(position: Int): Int" in the RecyclerView Adapter to pass the message type.
    - Accordingly, you can create two ViewHolders, inflate them in "onCreateViewHolder", bind them in "onBindViewHolder"
- A **Button** has to be used to display **SEND** (id: **chatSend**).
- A **EditText** has to be used for user input (id: **chatInput**).

IMPORTANT: Ensure there are no spelling errors on the texts and buttons.

IMPORTANT: Ensure the above IDs within the XML.

IMPORTANT: Ensure the above views are visible after rotation.

Implement the functionality

1. Implement the database

Implement the Room SQLite database to store the chat messages, each chat message contains 3 fields (columns):

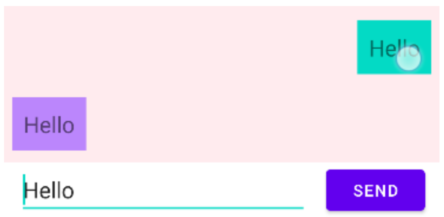
- chat_id: <primary key autoincrement>, with type Long.
- chat_msg: The text of the message, with type String.
- msg_type: The type of the message: send or response, with type Int.

Implement necessary insert, update, retrieve from this Room SQLite database. You can choose to use Flow or LiveData to implement the data pipeline.

2. Implement simple chatting, displaying, storing, and copying

- After hitting the **SEND** button (id: **chatSend**), the user input in **chatInput** will be inserted into the database, displayed in the **recyclerView**, and aligned to **right**.
- Ignore the hitting if the user input is empty or of empty spaces.
- After hitting, the text in **chatInput** will be cleared.
- The **recyclerView** will be scroll to the last message.

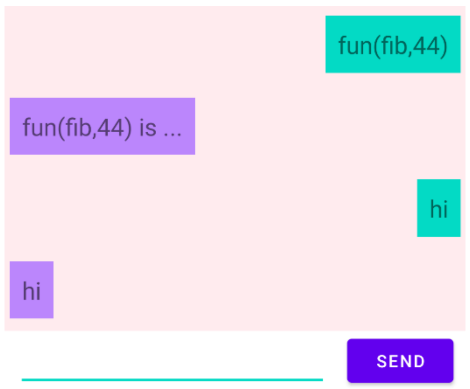
- If the user input in **chatInput** is not starting with "**fun**", the **same text** as the user input will be replied from the ChatBot. This response will be inserted to the database, and displayed in the **recyclerView**, and aligned to **left**.
- A click on one item in **recyclerView**, will copy the message to the **chatInput** (no need to send)**.**

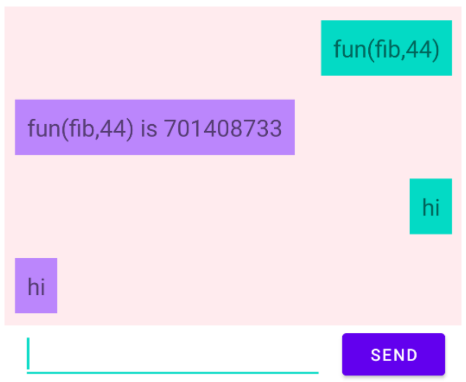


- If the app is relaunched: All chat history will be loaded and displayed in recyclerView and scrolled to the last message.

3. Implement computation

- If the msg starts with **fun**, then a computation is required. This computation returns an Int and takes two arguments:
  - The first one is a String which is the function name.
  - The second one is an Int which is the function input.
- E.g., fun(fib,6) will call Fibonacci function fib(6) and return 8:
  - If "fun(fib,6)" is sent by the user, a message "fun(fib,6) is …" will be **immediately replied** to avoid ANR or blocking the main UI. The user can continue chatting.
  - This computation will be dispatched to another thread (you can choose to use Coroutine, WorkManager or Thread to implement this), after the result is computed, the "fun(fib,6) is …" will be **updated** to "fun(fib,6) is 8".
  - Assume the format of user input is correct for computation.
- A map of high-order functions is provided in the **FunctionMap.kt** (a slow function **self** is also provided for testing), please add it into your "\src\main\java\edu\singaporetech\quiz03, the function fib can be referenced by using FunctionMap.map[key] with key="fib". During the testing, the **FunctionMap.kt** will be replaced, and more functions will be added.





Lab Quiz 3

(12 Marks)

Fork the repo **csc2007-quiz03-2023.**

1. Design the layouts the screen similar to the given screenshots. The views need to be visible after rotation. (1 Mark)
2. Implement a RecyclerView (id: **recyclerView**) to display the chats. (1 Mark)
3. Implement the alignment in the RecyclerView (id: **recyclerView**). (1 Mark)
4. Implement the non-empty checking, text clearance and scrolling to the last when hitting the SEND button (id: **chatSend**). (1 Mark)
5. Implement the hitting on one item in **recyclerView** to copy the message text to the **chatInput.** (1 Mark)
6. Non-fun simple chat message with response (same as the user input) will be inserted and displayed. (1 Mark)
7. Implement the Room database to save all chat history. All chats will be loaded and displayed in **recyclerView** and scrolled to the last message the app is relaunched. (1 Mark)
8. Invoke the high-order function in **FunctionMap.kt** if the chat starts with “fun”. (1 Mark)
9. Perform the high-order functions in other theads than the main thread and the user can still chat. (1 Mark)
10. The message will be updated after the computation is done. (1 Mark)
11. Remember to comment and indent the code and implement it in a modular fashion, using different methods or classes if appropriate. Ensure it conforms to Kotlin coding conventions. (2 Marks)

Commit and push all changes to your forked repository **csc2007-quiz03-2023.**

IMPORTANT: Do not change the activity names or package name. Ensure the job list follows its default order when firstly launched. Ensure there are no spelling errors on all the text. Remember to commit and push to repo!

END OF DOCUMENT