# CAN301 Final Report

——————————————————————————————————————

Department of Computer Science and Software Engineering

Xi'an Jiaotong Liverpool University

{——————————————————

—————————————}@xjtlu.edu.cn

## Abstract

Our team developed a multifunctional campus map app integrating navigation, Q&A integrating AI chatbot, social interaction, and information search functionalities. Users can locate themselves in real time on campus, view building details by clicking on campus structures, chat with AI, record and share campus activity schedules, and search for faculty information. This report provides a detailed explanation of the app's design and implementation process. If you want to login, you can use the following account: Username: Bird, Password: 123456.

Source code is available at:

https://github.com/LKCDaniel/CAN301-GPT-Oriented-Programming

## 1. Introduction

### 1.1 Motivation

The main motivation for this project is to make campus life easier for students by providing an all-in-one, user-friendly tool that caters to multiple aspects of campus navigation and information access. This app is designed to go beyond basic navigation by incorporating advanced features such as detailed building information, real-time location tracking, and access to faculty details, thereby offering a more engaging and practical solution for students.

Currently, the existing campus map provides only basic navigation for the SIP campus that is limiting in functionality and user experience. For example, students cannot easily get detailed information about a specific building or faculty, nor can they interact with the system to log or share campus-related activities. In addition, most of the current solutions lack social and interactive features, for which users browse through other alternatives.

This project tries to close these gaps by providing an even more interactive and feature-rich solution that would ensure not only better usability but also more engagement from the students. The application will include top-notch technologies: AI chat for query resolution, leaderboard-like social interaction, personalized user management, and much more. It aspires to redefine the way students navigate and communicate on campus.

Beyond this, the app aims to cultivate a more connected campus community where activity sharing, location-based interactions, and dynamic updates will be enabled. Such advancements may cut costs in time and efforts from users and engage one in a convenient, engaging, and collaborative platform.

### 1.2 Contributions

The key contributions of the app include:

- Providing real-time campus location tracking and building detail displays.
- Integrating a Large Language Model (LLM) chatbox to enable intelligent chatting features.
- Recording users' campus activity time and enhancing engagement through leaderboards.
- Supporting user-customized profiles and location sharing.
- Facilitating the search for faculty information on campus.

## 2. App Design

### 2.1 System Architecture

The application follows a modular, fragment-based architecture centered around a single hosting activity and a bottom navigation interface [1][2]. At the core is the **MainActivity**, which orchestrates the navigation between various feature-oriented fragments—such as the Map, Chat, Friends, and Profile screens, as shown in the Figure 1. These fragments are defined independently, allowing for clear separation of concerns and easier maintainability.

### 2.2 MainActivity

- Entry Point & Navigation Host: **MainActivity** serves as the base to the application and hosts a **NavController**, which manages fragment transactions and screen flows according to a predefined navigation graph.

- Bottom Navigation Integration: The **MainActivity** sets up a bottom navigation bar by setting the **appBarConfigoration**, which provides quick access to the main functional areas (Map, Chat, Friends, Profile). Clicking on each navigation item triggers a navigation action handled by the **NavController**, thus changing the fragment displayed on the screen.

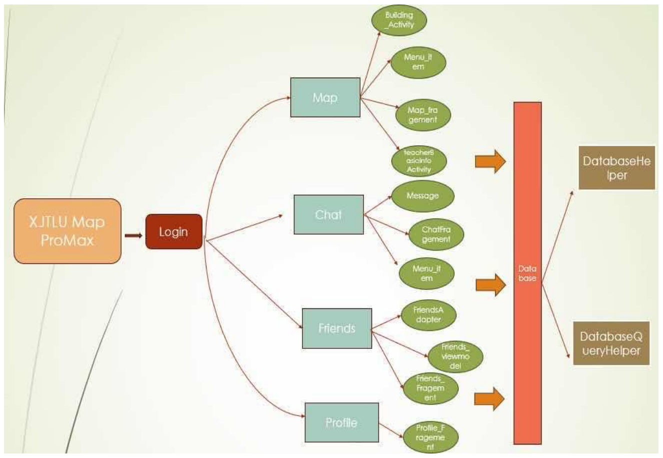


Figure 1. Architecture Diagram

- Shared Database Layer: The **MainActivity** also initializes and manages a connection to the local SQLite database through a **DatabaseHelper**. By doing so at the Activity level, data access can be facilitated for any component that needs it. Query helpers (like **DatabaseQueryHelper**) abstract low-level data access and present simplified methods to retrieve data from specific tables.

**2.3 Fragments (Map, Chat, Friends, Profile)**
Each section of the app is encapsulated within a dedicated fragment. They extend Fragment, and handle their own domain of responsibility separately:

- **MapFragment**: Shows maps for XJTLU, either SIP or TC campus. By clicking buildings or search professor names, it also directs to other activities that illustrate rich information.
- **ChatFragment**: Chat with LLM.
- **FriendsFragment**: Interact with friends, and compete with them based on the time spent on campus each day.
- **ProfileFragment**: Shows the user's personal details.
- **BuildingActivity:** This activity appears when the user clicks on any building. Specifically, the floor maps of the selected building are shown, and the user can switch between floors using a **seekBar**. These floor maps are also displayed using **InteractiveImageView**, allowing free map movement. Every room in each floor map is clickable, and clicking on an office will navigate the user to the respective **TeacherBasicInfoActivity**.

This activity has a fullscreen display and can dynamically toggle system UI visibility based on user interaction. When the user taps the screen, a toggle() method hides or shows the **ActionBar**, status bar, and navigation bar. This is done using a combination of **WindowInsetsController**, along with a Handler and Runnable tasks to schedule visibility changes.

## 2.4. Implementation Details

### 2.4.1 Map Module

**Functions Description**

Map module is the core part of this Application. Its core function is to provide user an information-rich interface with Xi'an Jiaotong-Liverpool (XJTLU) map. The information of buildings and teachers can also be queried and displayed. For instance, users can select a specific building in the map, see the basic information of its teachers, and search or click on a specific floor and room to learn the details of the corresponding teachers. Users can also search the teacher's name through the search bar, and quickly locate and query the information of the corresponding teacher. In addition, the interface will be marked on the map based on the user's actual geographical location (GPS) to help users understand their relative location on campus.

**Design Objective**

1. **Query in Anyway**: provide fuzzy search and drop-down screening function, users can quickly get all the results and the corresponding office location information through the teacher's name fuzzy search, reducing the user's manual search time cost. For example, to search CAN301 Lecturer “JianjunChen”, users can type relevant content in the search box, such as "Jian", "Ch", "Ji" .etc, and the search function will automatically return all qualified teacher names. By clicking on the corresponding teacher's name, you can directly jump to the teacher's information page

2. **Visual visualization:** The interactive map displays the building distribution and floor information in an intuitive way, enabling users to quickly find the teachers or buildings they need. Each building on the XJTLU map has a corresponding pin pointing to it. The user can click on the pin to jump directly to the interior map of the building, where all the classrooms are clearly located as a floor plan, and by clicking on the classroom, the user can also jump to the teacher's home page.

3. **Information association:** Associate teacher data (name, position, email, photo link, personal homepage and scholar page link) with the room number of the building to facilitate users to obtain comprehensive information.

4. **Location synchronization:** Using GPS location information, the user's location is mapped to the campus map to enhance the user's sense of positioning and navigation experience in the campus.

## Layout Design

1. **Main interface layout:** includes map area, search bar, drop-down list and button, map using **InteractiveImageView** custom control, support click and zoom operation. The search entry is implemented through the **SearchView** search box and Button at the top. A Spinner is used to display a drop-down list of search results.

2. **Map and floor switching:** Use **SeekBar** to switch the display status of different campuses (SIP, Taicang Campus) and building floors, as shown in Figure 2. When the user drags the progress bar, the map will be updated to display the floor plan and room information of the corresponding floor.

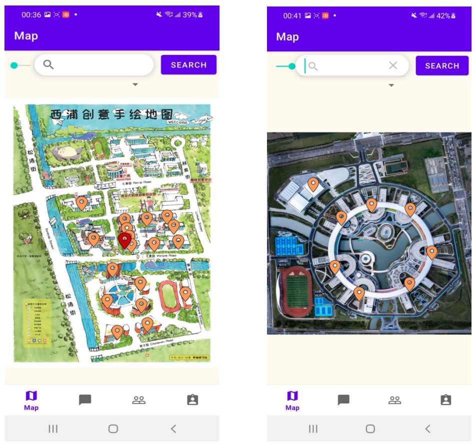


Figure 2: Switch of SIP map and Taicang

3. **Teacher information display interface:** In the **TeacherBasicInfoActivity**, use an independent layout file (including name, position, email, teacher photo, details link button) to display teacher details, and provide the function of returning and browsing more details. Figure 3 shows an example.

4. **UI element style:** The overall use of school theme color system with simple and clear ICONS (e.g. placeholder map, error chart with a XJTLU bird), through **placeholder_photo** and other resources to give the user the default feedback during data loading and when loading fails.

## User Feedback

1. **Map click feedback:** When the user clicks the building in the map, if there is no corresponding data or the building information is missing, there will be a corresponding reminder in the log, and the interface will use the "under construction" feedback to the user.

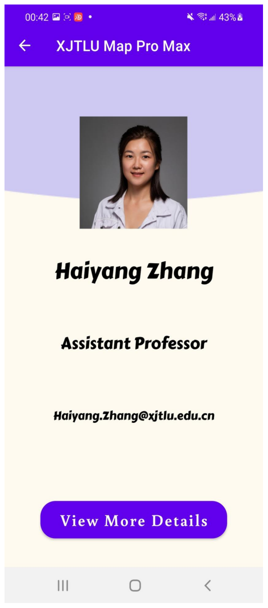


Figure 3: Teacher Information Display

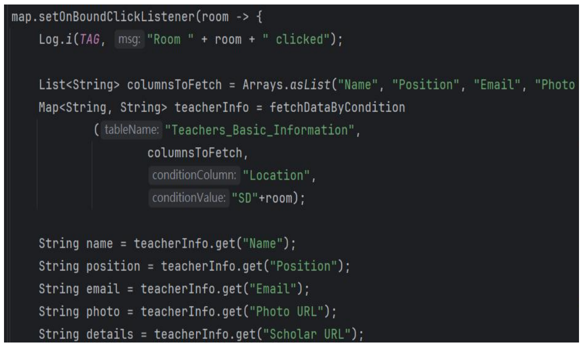
2. **Search feedback:** When the user enters the content in the search box and clicks the search button, if there is no matching result, the system will display a prompt message (such as "Please enter the search content" or provide relevant prompts when the database query has no result).

3. **Teacher information Click feedback:** After the user selects a specified teacher from Spinner, the user will jump to the **TeacherBasicInfoActivity** interface. If the teacher information is incomplete or no data is found, the default information (such as "XJTLU Member") and placeholder map and N/A are used to inform the user that the data is missing.

## Technical realization and challenge

1. **Interactive map implementation:** Through the **InteractiveImageView** to achieve map translation, zoom and click the region judgment function. HashMap is used to store the mapping relationship between buildings and coordinate areas, and the corresponding event is triggered when the user clicks on a specific area.

2. **Database query and data processing:** Use SQLite database to pre-load teacher and room data through **DatabaseHelper**. The query methods include precise and fuzzy matching, and the fast retrieval of teacher information is realized by **fetchDataByCondition** and **fuzzySearchName** methods, presented in Figure 4.



```
map.setOnBoundClickListener(room -> {
    Log.i(TAG, msg: "Room " + room + " clicked");

    List<String> columnsToFetch = Arrays.asList("Name", "Position", "Email", "Photo
    Map<String, String> teacherInfo = fetchDataByCondition
            ( tableName: "Teachers_Basic_Information",
                    columnsToFetch,
                    conditionColumn: "Location",
                    conditionValue: "SD"+room);

    String name = teacherInfo.get("Name");
    String position = teacherInfo.get("Position");
    String email = teacherInfo.get("Email");
    String photo = teacherInfo.get("Photo URL");
    String details = teacherInfo.get("Scholar URL");
```

**Figure 4: Search Teacher Info**

3. **Image loading and rollback mechanism:** This project uses Glide library to load teacher photos from the network. If **photoUrl** is unavailable or fails to load, use **placeholder_photo** to display the default profile picture to ensure a beautiful interface and smooth interaction. The process of loading the teacher's photo needs a stable network communication. This mechanism of error processing can help improve user experience by preventing blank or broken images, ensuring a seamless fallback to the default image in case of issues such as network instability or invalid photo URLs.

4. **GPS location mapping**: Through the known geographical coordinates of the campus and mapping algorithm, the GPS coordinates are converted into normalized coordinate points in the map to display the relative position of the user in the map. In this conversion process, coordinate deviation, offset and proportional conversion should be considered.

### 2.4.2 Chat Module

**Function Description**

The Chat module provides intelligent Q&A functionality for users and supports natural language interaction. Through the **RecyclerView** dynamic display of message records, and access to the API of the large language model (LLM), the user can quickly get a reply after input questions, the Figure 5 shows the interface.

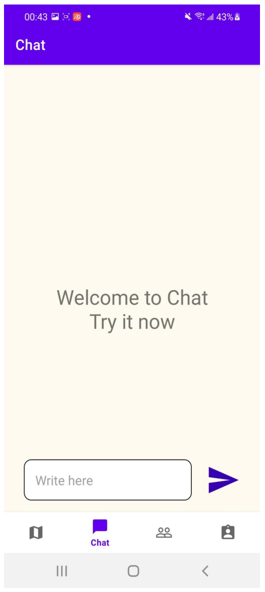


Figure 5: The chat interface

**Architecture Design**

The Chat module consists of four key parts that work closely together to implement a complete flow of message input, processing, and output:

1. **ChatFragment**: This part is responsible for initializing the user interface and handling the logic, including dynamic layout of **RecyclerViews** and calling third-party APIs from OkHttpClient, managing sending and receiving messages.
2. **Message**: This is a data model for messages that encapsulates the message content and the sender (user or LLM), and provides getter and setter methods for data access.
3. **MessageAdapter**: Inherits from **RecyclerView**.**Adapter** to bind message data to **RecyclerView** and dynamically update the display effect of user and system messages to achieve efficient message rendering.
4. API call: It implements asynchronous communication with LLM chat service through OkHttp, parses JSON format response and dynamically updates interface, and has error handling mechanism to improve reliability.

**User Interface**

The layout is defined in the 'fragment_chat.xml' file and includes:

**RecyclerView**: Dynamic display of messages

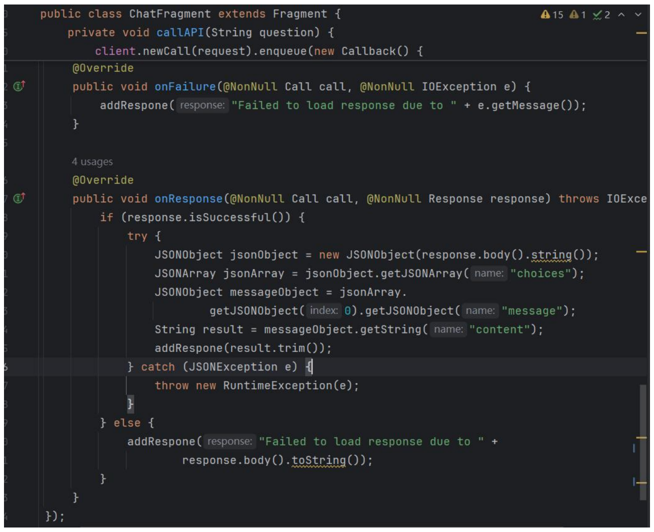
**EditText** and **ImageButton**: For users to type and send messages

**TextView**: This initially displays the welcome message.

**Technical Implementation**

Interactions with the large language model API are handled asynchronously using OkHttpClient.

The process of sending and receiving messages is as follows: when the user enters a message, the system calls the **'callAPI'** method to send the message to the server. Then, it receives the response from the server through the callback mechanism of **OkHttp**, parses the returned JSON data, and dynamically displays the parsed content in the **RecyclerView**. In addition, in order to ensure the fluency and real-time performance of the user interface, the **'runOnUiThread'** method is used to update the interface synchronously to ensure that users can quickly see the latest message feedback, so as to improve the overall interactive experience. The Figure 6 shows the implementation details.



```
public class ChatFragment extends Fragment {
    private void callAPI(String question) {
        client.newCall(request).enqueue(new Callback() {
    @Override
    public void onFailure(@NonNull Call call, @NonNull IOException e) {
        addRespone( response: "Failed to load response due to " + e.getMessage());
    }

    4 usages
    @Override
    public void onResponse(@NonNull Call call, @NonNull Response response) throws IOExce
        if (response.isSuccessful()) {
            try {
                JSONObject jsonObject = new JSONObject(response.body().string());
                JSONArray jsonArray = jsonObject.getJSONArray( name: "choices");
                JSONObject messageObject = jsonArray.
                        getJSONObject( index: 0).getJSONObject( name: "message");
                String result = messageObject.getString( name: "content");
                addRespone(result.trim());
            } catch (JSONException e) {
                throw new RuntimeException(e);
            }
        } else {
            addRespone( response: "Failed to load response due to " +
                    response.body().toString());
        }
    }
});
```

Figure 6: Implementation of the response

In the process of sending and receiving messages, the system also designs a perfect error handling mechanism to ensure the stability of functions and the optimization of user experience. When a network error occurs, the system will catch the exception and feedback the error information to the user through a user-friendly prompt message to avoid operation interruption caused by unexpected circumstances. At the same time, when parsing JSON data, if there are any problems such as malformed or missing data, the system will provide a default response as a fallback solution to ensure that the application can continue to operate normally. In addition, these handling mechanisms combined with the **'runOnUiThread'** method not only achieve real-time error feedback, but also further improve the overall interaction fluency and stability, providing a more reliable user experience for users.

**Technical Challenges and Solutions**

When dealing with complex JSON data structure, the system designs a parsing scheme according to the characteristics of nested objects, and gradually extracts the nested level data and maps it to the corresponding message model to ensure the accuracy and integrity of data processing. In addition, to maintain the responsiveness of the interface in high message volume scenarios, the system optimizes the logic of the **RecyclerView** adapter, such as reducing unnecessary data refresh operations and rational reuse of view resources to avoid performance degradation. Through these optimization measures, the system ensures the fluency of message display and the efficient operation of the interface while dealing with complex data, and provides users with a better interactive experience.

### 2.4.3 Friends Module

## Overview

The Friends page of the app offers a leaderboard displaying users based on their study hours, calculated using real GPS data collected during their time on campus. The leaderboard will show the top three users daily, showing each user's percentile ranking to build a sense of community and competition.

### Architecture

The Friends module includes three main components:

1. **Friends Adapter**
   The **FriendsAdapter** class is responsible for the presentation logic related to the leaderboard. This class is specifically designed to perform an effective binding of data with different UI elements in the view of a leaderboard. It has the following features:

   - Inflate Layout: It inflates the layout of a single leaderboard item dynamically for each Friend object.
   - Bind Data: It maps the Friend object's attributes - name, score, and rank, to corresponding UI components; for example, TextView for names and ranks.
   - Efficient Recycling: It makes use of the ArrayAdapter pattern, which assures efficient usage of memory. It does this by reusing the views in ListView instead of creating new views unnecessarily, thus improving application performance on large datasets.
   - Customizable Display: Very easily extendable to include top performers with badges or even extending information, such as weekly improvement.

2. **Friends Fragment**
   The **FriendsFragment** class will provide the entry point for the Friends page, which will be responsible for the following:

   Lifecycle Management: Manages the fragment's lifecycle, including setup and tearing down of resources with methods like **onCreateView** and **onDestroyView**. UI Binding: It utilizes **FragmentFriendsBinding** to bind the UI components defined in the XML layout to their respective logic in the fragment class, ensuring a clean separation between presentation and business logic.

3. **Friends ViewModel**
   The **FriendsViewModel** class encapsulates all the data logic for the Friends module. Its responsibilities will include the following:

   - Fetch Data: Responsible for reaching out to the database--via **DatabaseQueryHelper**--to fetch data like the top three users and all users' ranking.
   - Data Transformation: It converts raw database results to structured Friend objects, easily consumable by the adapter.
   - State Management: It keeps the state of the leaderboard data and provides this state to the UI for changes without directly accessing the database or doing any heavy computations in the fragment.

4. **Backend Integration**
   The module is designed to interact with a backend database via the **DatabaseQueryHelper** class. This component will execute SQL queries to fetch real-time data for user rankings and scores and ensures data consistency by handling user updates and leaderboard **recalculations**.The method **getTopFriends**() is used for retrieving pre-processed data for the top-ranked users.

### Features and Implementation

1. **Dynamic Leaderboard Display**
   The leaderboard is populated dynamically through the FriendsAdapter, ensuring users always see updated rankings based on the latest GPS data. Figure 7 shows the page.

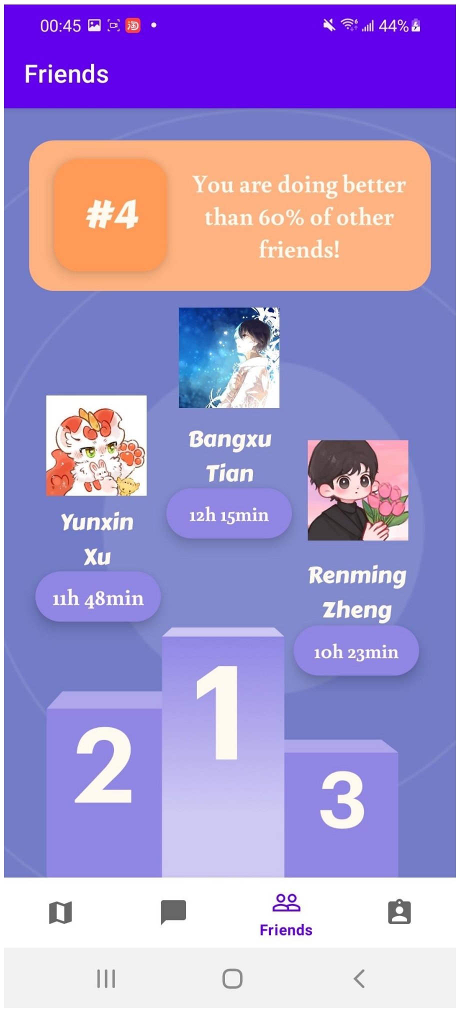


Figure 7: Leaderboard

2. **Top Three Highlights**
   The **getTopFriends** method retrieves and displays the top three users daily, emphasizing their achievements and encouraging others to increase campus engagement.

3. **Percentile Display**
   Each user's percentile ranking is calculated and shown to foster engagement and competitiveness. This ensures that users who are not in the top three can still see their progress and feel motivated.

**Future Enhancements**

Potential improvements to the Friends module include:

- Adding graphical visualizations, such as bar charts or progress indicators, to make rankings more engaging.
- Providing personalized tips or suggestions for increasing study hours based on a user's activity history.
- Allowing users to form groups and compete collectively for increased social interaction.

### 2.4.4 Profile Module

**Function Description**

The Profile module provides an interface for the user to manage personal information while dynamically displaying the current location. Users can edit, save personal information, and display geographical location in real time through integration with the Map module. All data is stored locally to ensure consistency.

**Module Contents**

1. Manage user information: Support users to enter, update, and save personal information such as name, gender, major, and interests.

2. Dynamic display location: It is integrated with the Map module to obtain and display the user's current location in real time.

3. Provide a simple and intuitive display interface.

**Layout Design**

Figure 8 presents the interface of this functionality. The Profile module's layout file fragment_profile.xml uses ConstraintLayout to achieve a responsive design and contains the following main components:

1. Background and user avatar

2. Input field: Contains five input fields for displaying and editing, including name, gender, major, interest, and location. The name, gender, major, and interest input fields (**EditText**) allow users to edit. The location input field is set to not editable and dynamically displays the current location obtained from the Map module.

3. Save button: Positioned at the bottom of the interface, the Save button enables users to efficiently save their edited personal information.

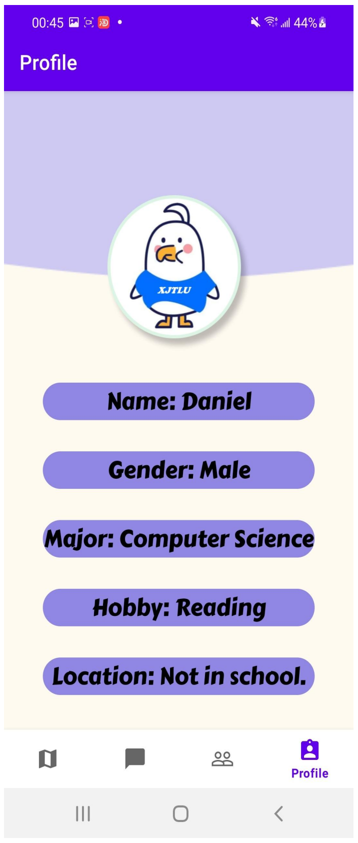


Figure 8: Profile page interface

**Technical Implementation:**

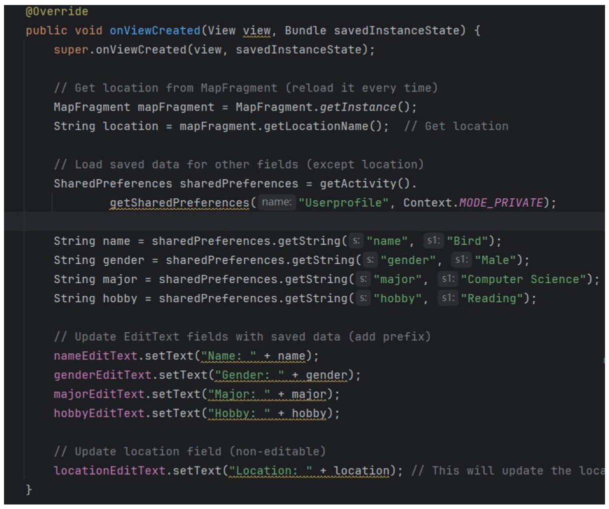
- **Data Management:** To manage user data efficiently, the application **usesSharedPreferences** to save and load data asynchronously. This ensures consistency and responsiveness when retrieving user preferences. When the Fragment loads, it checks whether user data exists in **SharedPreferences**. If no data is found, it displays default values, ensuring a smooth user experience even on first use or after data loss.

- **Location Synchronization:** Location data is synchronized with the Map module through the singleton pattern. By calling **MapFragment getInstance**() and **getLocationName**(), the system retrieves real-time location information. Upon Fragment loading, the geolocation information is updated in the location field, ensuring the user always sees the latest location data. This guarantees that the displayed location is always current, improving accuracy and reliability.

- **User Input:** User input is captured using the getText() method from input fields. To ensure clean data, the input is processed by calling trim() to remove extra white spaces and replace() to remove any default prefixes. The processed input is then stored in **SharedPreferences** using the apply() method, which saves the data asynchronously for improved storage efficiency and better performance.

- **Feedback:** After successfully saving the user's input, the system provides immediate feedback through a Toast message. This confirmation helps reassure the user that their data has been saved successfully, enhancing the user experience by providing timely responses to their actions.

- **Life Cycle:** The **onViewCreated** method is used to load saved user data and dynamically update the values of all input fields each time the Fragment is loaded. This ensures that the user sees their previously entered data, preserving continuity across sessions. Additionally, the geolocation information is re-fetched during each Fragment load, ensuring that the displayed location is always up to date with the latest geolocation data. Its details are shown in Figure 9.



```java
@Override
public void onViewCreated(View view, Bundle savedInstanceState) {
    super.onViewCreated(view, savedInstanceState);

    // Get location from MapFragment (reload it every time)
    MapFragment mapFragment = MapFragment.getInstance();
    String location = mapFragment.getLocationName();  // Get location

    // Load saved data for other fields (except location)
    SharedPreferences sharedPreferences = getActivity().
            getSharedPreferences( name: "Userprofile", Context.MODE_PRIVATE);

    String name = sharedPreferences.getString( s: "name",  s1: "Bird");
    String gender = sharedPreferences.getString( s: "gender",  s1: "Male");
    String major = sharedPreferences.getString( s: "major",  s1: "Computer Science");
    String hobby = sharedPreferences.getString( s: "hobby",  s1: "Reading");

    // Update EditText fields with saved data (add prefix)
    nameEditText.setText("Name: " + name);
    genderEditText.setText("Gender: " + gender);
    majorEditText.setText("Major: " + major);
    hobbyEditText.setText("Hobby: " + hobby);

    // Update location field (non-editable)
    locationEditText.setText("Location: " + location); // This will update the loca
}
```

Figure 9: Implementation of onViewCreated()

**Technical Challenges and Solutions:**

Firstly, in order to ensure the synchronization between the location field and the Map module, we adopt the singleton pattern to solve the communication problem between different modules. With this design, the location data can be consistent in the global scope, avoiding display errors or delay problems caused by data out of sync. In addition, in order to improve the user experience, especially on different devices, we carefully designed the responsive UI. The UI can automatically adapt to different screen sizes and resolutions, ensuring that the interface can be displayed clearly, smoothly and provide a consistent user interaction experience on mobile, tablet and other devices.

## 3. Evaluation

### 3.1 Challenges

- Accuracy of Map Positioning: Addressed by optimizing the positioning algorithm.
- Response Speed of Chat Interface: Improved by increasing the concurrency of API requests.
- Reliability of Data Synchronization and Storage: Enhanced using **SharedPreferences** and the Singleton pattern to ensure data consistency.
- Working with SQLite in this project presented several key challenges, particularly around database design, synchronization, and memory management.

**Thread synchronization:** SQLite itself is not thread-safe, so when multiple threads try to access or modify the database at the same time, I must ensure proper synchronization. To fix this challenge, I implemented a thread-safe approach, using synchronous blocks or locks if necessary, to ensure that only one thread can perform database operations at any given time, thus preventing race conditions from occurring.

**Singleton design pattern:** During the database creation process, it is common to instantiate the database multiple times, so that each class cannot correctly reference the instance of the same database. I implemented a singleton pattern for the **DatabaseHelper** class. This design ensures that only one instance of a database connection is created throughout the application lifecycle, avoiding unnecessary overhead and maintaining a single point of control for all database interactions. At the same time, I also covered getter and setter functions, which conform to Object Oriented Programming (OOP) principles.

**Memory Leaks Prevention**: Proper memory management was a critical challenge, particularly with SQLite connections and cursors, which need to be explicitly closed to avoid memory leaks. To solve this problem, I made sure to properly close all database connections and cursors after use. Specifically, I utilized the **onDestroy** () lifecycle method in Android to release resources when an activity or fragment is destroyed. Which is used to ensure that database connections, cursors, and any other resources associated with the activity are cleaned, thereby preventing potential memory leaks. So that optimizing memory usage and preventing unnecessary memory retention.

**Database design principles:** When designing database schemas, I followed key principles such as standardization to reduce redundancy and ensure data integrity. I use foreign keys to establish relationships between tables, ensure referential integrity to prevent isolated records. This approach also allows for easier maintenance and querying of relevant data, such as linking teachers to their respective office locations and details and link teacher Name of unique characters through various tables.

**Efficient Querying**: Given that teacher data can grow over time, I optimized the database schema for fast queries. To further improve search efficiency, I implemented a combination of fuzzy search and exact search methods. Fuzzy search allows users to quickly retrieve results even with partial or misspelled queries, while exact search ensures precise results for more specific queries. This dual approach minimized the load on the database, reduced search time, and enhanced overall performance.

### 3.2 User Feedback

- The map feature is intuitive and easy to use.
- The chat feature provides accurate responses to user queries.
- The personal information module is user-friendly, with reliable data persistence.
- The app's UI design is clean and visually appealing, with well-coordinated colors enhancing the user experience, the colors used are shown in Figure 10.

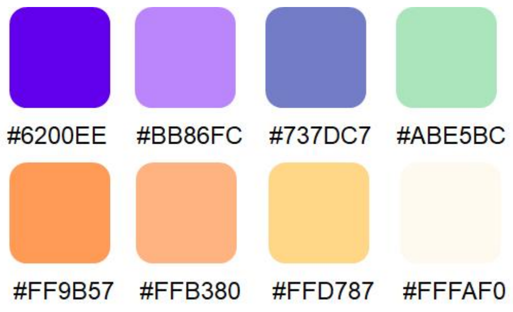


**Figure 10: Colors mainly used in UI**

### 3.3 Comparison

- Compared to similar campus apps:
- This app offers more precise positioning functionality.
- Integrates AI chat for added interactivity and fun.
- Supports dynamic updates of personal location in the interface.
- Allows viewing detailed information about campus buildings and searching for faculty information.

## 4. Conclusion

Our team developed a multi-functional campus map app, XJTLU Pro Max, which successfully integrated a variety of functions, including real-time campus location tracking, office information viewing, intelligent AI chat interaction, teacher information search, school duration ranking and other functions. By combining these different features into one platform, a more comprehensive and engaging user experience is provided.

This design and implementation of such a multi-functional app involved several key challenges, such as creating interactive maps with variable sizes, integrating AI chatbots, and ensuring seamless user interaction and database initialization through asynchronous communication. The application strictly adheres to the Object-Oriented Program philosophy and architecture, using a modular architecture and data-driven approach to make navigation and information retrieval smooth, while also considering the user experience. Using SQLite for local data management and advanced APIs for real-time communication demonstrates ability of our team to work with a variety of technologies. Overall, the app not only simplifies campus navigation, but also promotes a more connected and interactive campus community, ultimately providing students with a more efficient and engaging one-stop information integration platform.

In the future, our team may consider adding multiple languages for greater accessibility; Moreover, the addition of cloud services, as well as multi-user interaction functions, to deal with the synchronization of information on different users' mobile phones; In addition, the team will supplement all the information of other teaching buildings except SD building in the same way to create a complete XJTLU map interactive system.

## REFERENCES

[1] W. Ngaogate, “Memory usage comparison in an android application: Basic object-oriented programming vs decorator design pattern: Coding styles for keeping low memory usage in a mobile application.” in Proceedings of the 4th International Conference on Information Technology and Computer Communications, 2022, pp. 125–131.
[2] F. Onu, O. Ikedilo, B. Nwoke, and P. Okafor, “The importance of object-oriented programming in this era of mobile application development,” IOSR Journal of Computer Engineering (IOSR-JCE) e-ISSN: 2278-0661, p-ISSN: 2278, vol. 8727, pp. 30–40.