Department of Computer Science

The University of Hong Kong

COMP 4801 Final year project

Final Report

Group:

fyp23029

Project Title:

All-in-one mobile application for elderly

Group Member (UID):

Ng Tsz Hei (3035780192)

Supervisor:

Professor Wu, Chuan

Date of Submission:

24, April 2024

I) Abstract

In recent times, there has been a noticeable increase in telephone deception cases in Hong Kong, with a significant number of elderly individuals falling victim to such scams. Moreover, incidents involving missing elderly persons and accidents have raised concerns within society. To address these issues, it is crucial to develop specialized software that prioritizes the personal and property safety of the elderly. However, existing software in the market does not cater specifically to the needs of the elderly.

This paper aims to document and demonstrate the entire software development lifecycle of an all-in-one smartphone application designed for the elderly. The development process will encompass preliminary design, implementation, and testing. The deliverables of this project will include the application itself, user interface design, database design, system architecture, and test case design.

The key features to be incorporated in this application are location tracking, body status monitoring, fall detection, telephone deception prevention, and integration with LLM technology. By integrating these features, the application will ensure the safety of the elderly and make their lives easier. It can serve as a solid foundation for a general-purpose application catering to the needs of the elderly.

In the future, potential integrations could involve collaborating with the government or non-governmental organizations (NGOs) to provide assistance in elderly services and welfare through the application. By expanding its functionality and partnerships, the application can further enhance its support for the elderly, promoting their well-being and security.

II) Acknowledgement

I would like to express my deepest gratitude to my supervisor, Professor Wu Chuan, from the Computer Science department of HKU, for her invaluable advice and guidance throughout the project. Without her assistance, this paper would not have been successfully and smoothly completed.

I would also like to acknowledge Mr. Matthew Anderson from the Centre for Applied English Studies of HKU for his language and writing support. With his assistance, the paper has been written in a more structured and professional manner.

Table of Contents

III) List of Figures

IV) Abbreviations

Abbreviation	Definition
UI	User Interface
IDE	Interactive Developing Environment
LLM	Large Language Model
API	Application Programming Interface
UML	Unified Modelling Language

1. Project Introduction

This section will emphasize the introduction and relevant information regarding this project, and it consists of five parts. Section 1.1 will focus on the background of this project, while Section 1.2 will discuss the motivation behind it. Section 1.3 will highlight the current research gap in this topic, and Section 1.4 will state the objectives and deliverables of this project. Lastly, Section 1.5 will provide an outline of the remaining report.

1.1 Project background

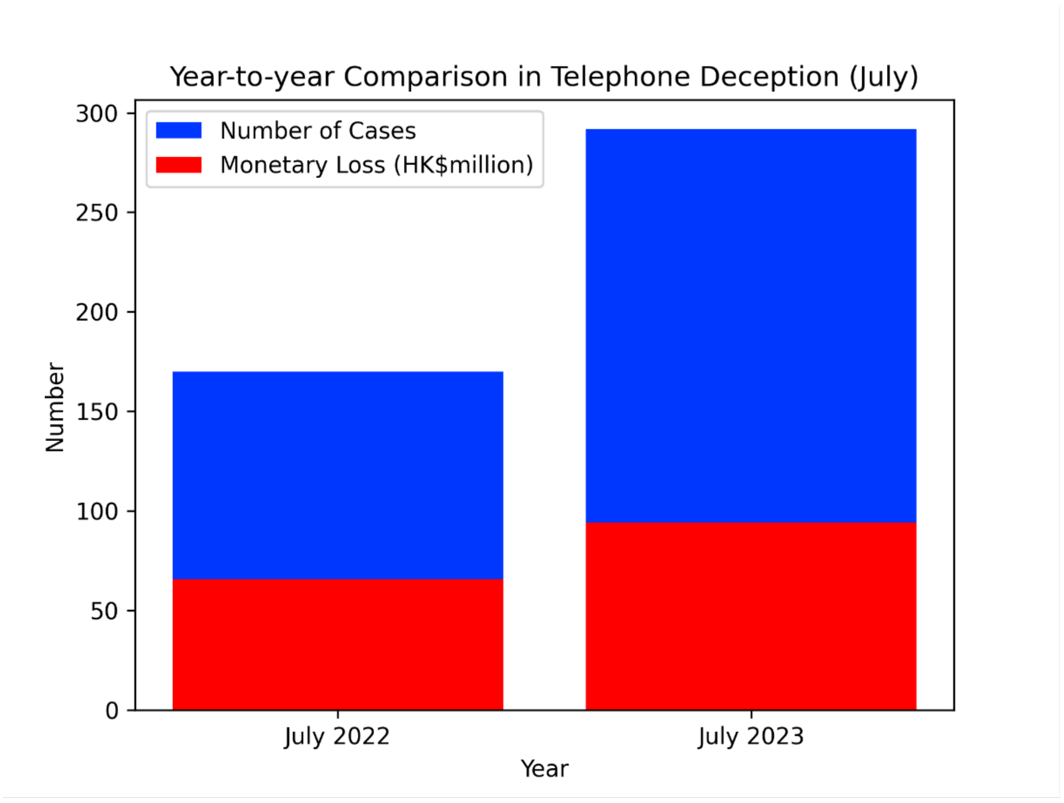


Figure 1 Year-to-year comparison in telephone deceptions' figures (July) [1]

Figure 1 shows a year-to-year comparison of telephone deceptions in Hong Kong provided by the Anti-Deception Coordination Centre of the Hong Kong government. The figure indicates a rise in the number of telephone deception cases from 170 to 292, representing a 71.76% growth, and a rise in monetary loss from 65.65 million to 94.2 million, with a 43.49% growth, from July 2022 to July 2023 [1]. This highlights the surge in the frequency and monetary losses of telephone deception in recent years, necessitating the need for action to address the worsening situation.

In addition, according to the *Legislative Council Panel on Security Initiatives for Preventing and Combating Deception Cases*, there are two prevalent modi operandi of telephone deception known as "Pretend Officials" and "Guess Who." In 2020, "Pretend Officials" cases accounted for approximately 57%

of all telephone deception cases (a total of 1150 cases), with 30% of the victims aged 61 or above. On the other hand, "Guess Who" cases accounted for 43% of the cases, with 60% of the victims aged 61 or above [2, p.4]. Overall, more than 40% of telephone deception victims are elderly individuals aged 61 or above, which represents a significant portion. This concerning trend highlights the need for extra awareness and measures to address the issue.

1.2 Project Motivation

In addition to telephone deception, the physical vulnerability of the elderly and their susceptibility to geriatric diseases such as Alzheimer's disease and dementia, which cause memory and cognitive decline, are concerning factors. Elderly individuals going missing or experiencing accidents, such as fainting and falling, are events that require our attention. Furthermore, the elderly may need assistance in problem-solving for daily life challenges and integrating into a rapidly changing society. However, their family members may not always be available to provide help. It would be preferable to have someone available to offer guidance and support to the elderly, such as providing advice on traveling to specific destinations and helping them bridge the gaps they may face in society.

1.3 Research Gap

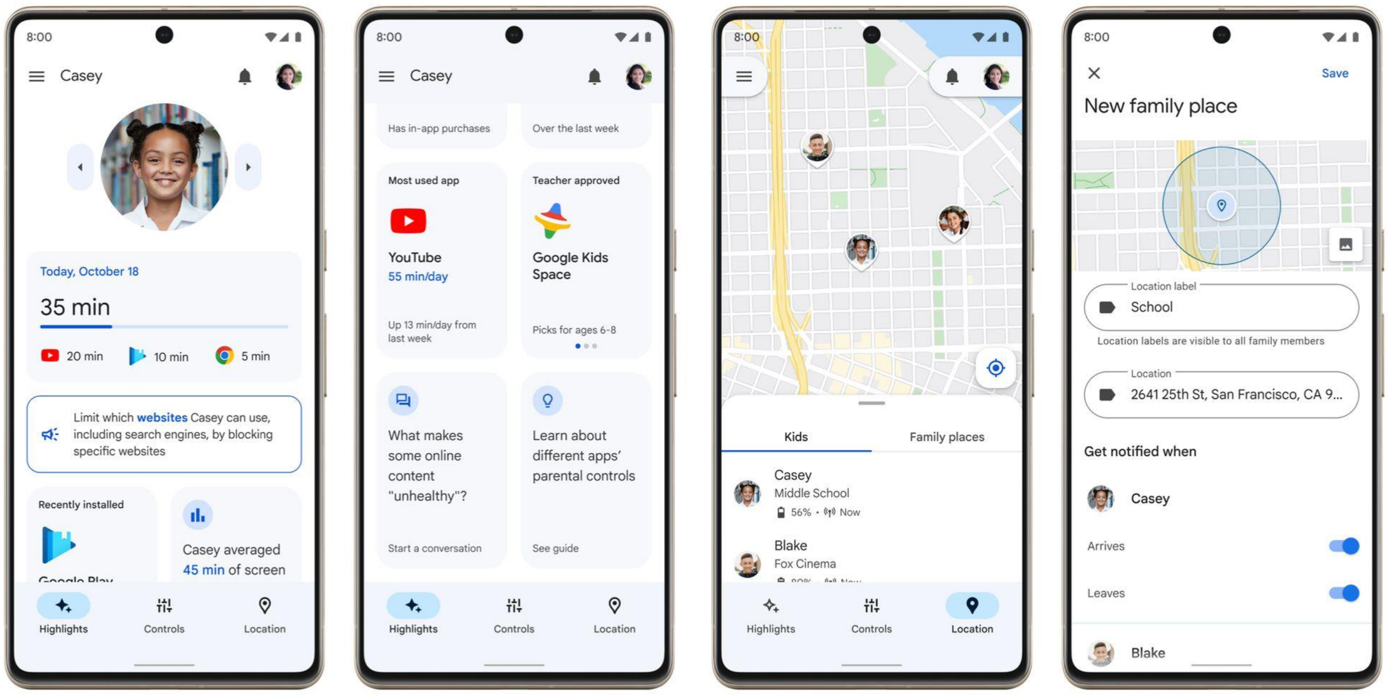


Figure 2 User interface of Google Family Link App [3]

As solutions to the aforementioned problems, several smartphone applications already exist in the market. For instance, Whoscall is an unknown call identification application that utilizes data science to analyze a large volume of call data and create an optimal model for call identification [4]. Google Family Link is a parental control application that offers a location tracking function and allows parents (or family members) to manage the digital devices and accounts of their children (or elderly relatives) [5]. Apple Health is an application that connects to smartwatches and organizes users' important health information, which can be used to monitor the health conditions of the elderly [6].

However, these applications are not specifically designed with the elderly in mind, and their design language may not cater to their specific needs. For example, Figure 2 displays screenshots of the Google Family Link app [3], where the user interface appears complicated with an abundance of information. Additionally, the small text size may make it difficult for the elderly to comfortably read the content. These issues can create challenges for the elderly in learning and using these applications. Moreover, managing multiple applications simultaneously can be problematic for elderly users. Therefore, an all-in-one smartphone application that is specifically designed to be elderly-friendly would be a better solution to help them address the aforementioned problems.

1.4 Project Objective and Deliverables

The goal of this project is to develop an all-in-one, cross-platform mobile application called Elder Link, that is designed specifically for the elderly to address the issues mentioned above, enhance their personal and property safety, and improve their quality of life. To achieve the goal, there are 5 main features to be implemented and they are as follows:

1) A calls filtering feature to identify suspicious call and prevent telephone deception.
2) A location tracking feature to assist searching in case of elderly missing.
3) A body status monitoring feature to detect dangerous situation like high blood pressure and prompt the family members to take emergent action.
4) A fall detection feature to detect falling cases and notify family members to take subsequent action.
5) A LLM chatbot feature to provide personalized support to the elderly.

The deliverables of this project will include the application itself, as well as related documentation such as user interface design, database design, system architecture, and test case design.

1.5 Outline of remaining report

The subsequent sections of this paper will concentrate on the complete software development process of the application. Section 2 will discuss the methodology employed to accomplish the project. Section 3 will cover the current results, along with the accompanying discussion. In section 4, expected results in the coming semester and the remaining work plan will be discussed. Finally, in section 5, a conclusion will be provided, summarizing the project and outlining the next steps.

2. Project Methodology

This section will emphasize the methodology utilized to complete this project, which is the tools and environment setup employed for developing the application. Section 2.1 will introduce the front-end tool and Interactive Development Environment (IDE) utilized for developing the application, highlighting their advantages. Section 2.2 will focus on the back-end setup of the application. Section 2.3 will discuss the software employed for modeling the application structure. Section 2.4 will delve into the software used for designing the user interface. Lastly, Section 2.5 will explore the software utilized for version control and project management, and section 2.6 will explore the methods used for testing.

2.1 Front End and IDE

To build the user interface of the application, React Native, a Javascript library for creating user interfaces, is used. React Native's declarative UI paradigm and JavaScript enable it to wrap native code and interact with native APIs, allowing for cross-platform development. This means that the application will run seamlessly on both Android and IOS smartphones, providing users with a consistent experience regardless of the brand of their smartphone [7].

For the IDE, Visual Studio Code will be utilized. It is a powerful text editor that offers a range of helpful features for coding, such as syntax highlighting, quick debugging, built-in Git commands, and customizable extensions based on the programming languages and services required by developers [8]. These robust and convenient features can significantly ease the development process and accelerate the pace of development.

2.2 Back End

For the database and server of the application, we will be utilizing Firebase by Google. Firebase is a robust app development platform that offers a wide range of cloud services for various backend functionalities, including database setup, server hosting, authentication, and user analytics. With Firebase, developers can benefit from high performance and stability in their applications. Additionally, Firebase ensures a high level of security, prioritizing the protection of user privacy [9].

2.3 Modeling of application structure

To create UML diagrams, specifically use case diagrams, sequence diagrams, class diagrams, and ER diagrams for the database design, Lucidchart will be utilized. Lucidchart is an intelligent diagramming application that facilitates team collaboration and communication [10].

A use case diagram provides a high-level overview of the relationships and interactions between actors (users) and the system. It helps in understanding the system's functionalities and the roles of different actors involved. [11]

A sequence diagram illustrates the order in which objects interact and represents runtime scenarios. It visualizes the flow of messages between objects and helps in understanding the dynamic behavior of the system [11].

A class diagram describes the structure of the system by modeling its classes, attributes, operations, and relationships between objects. It provides a static view of the system's architecture and aids in understanding the relationships and dependencies between different classes [12].

In addition to these UML diagrams, Lucidchart will also be used to create an ER diagram for modeling the database design, which helps in visualizing the structure and relationships of database entities.

2.4 UI prototype

Figma will be used to design the Graphical User Interface for the application. Figma is a UI design application that enables designing realistic interactive prototypes for quick iteration of flows and states. It offers high team visibility, which enables team cooperation, as well as providing tools for turning the UI prototype into real code [13] to enhance development efficiency.

2.5 Version control and project management

GitHub will be used for version control and project management. It is a website and cloud-based service designed to assist developers in storing and managing their code, as well as tracking and controlling changes to it. GitHub utilizes Git, an open-source, distributed version control system that allows each developer to have a complete copy of the codebase and its history on their own computers.

Version control is implemented through branching and merging. Branching involves creating a duplicate of the source code in a new branch, allowing changes to be made without affecting the original main branch. Merging, on the other hand, entails integrating the finished and well-tested code back into the main branch, making it official. Every branching and merging action is tracked and can be reverted if necessary, enabling a return to previous checkpoints in case critical bugs are encountered.

Furthermore, GitHub provides project management features such as progress statistics and contribution tracking [14].

2.6 Testing

The React Native Testing Library will be utilized for both unit testing and integration testing. Unit testing is a type of software testing that focuses on validating the functionality of individual units or components within a software system. Its goal is to ensure that each unit functions correctly and meets the specified requirements [15].

Integration testing, on the other hand, is conducted after unit testing and aims to verify that the interface between two units or modules is functioning correctly and transferring data accurately. This testing process ensures that no errors occur when merging different units together [16].

User acceptance testing, on the other hand, is performed manually by real end users. Its purpose is to evaluate whether the application fulfills their requirements in a real-world scenario and determines if the overall user experience is satisfactory [17]. Testing on whether the fall detection will work as expected in falling demonstrations will also be a part of the user acceptance test.

3. Results and Discussions

This section will focus on presenting the current results achieved so far. At this stage of the project, the preliminary design of the application has been completed. Section 3.1 will describe the use cases of the application. Section 3.2 will describe the underlying execution logic for each use cases. Section 3.3 will summarize the components involved in the execution logic. Section 3.4 will focus on the database design. Section 3.5 will discuss the user interface of the application.

3.1 Use case design

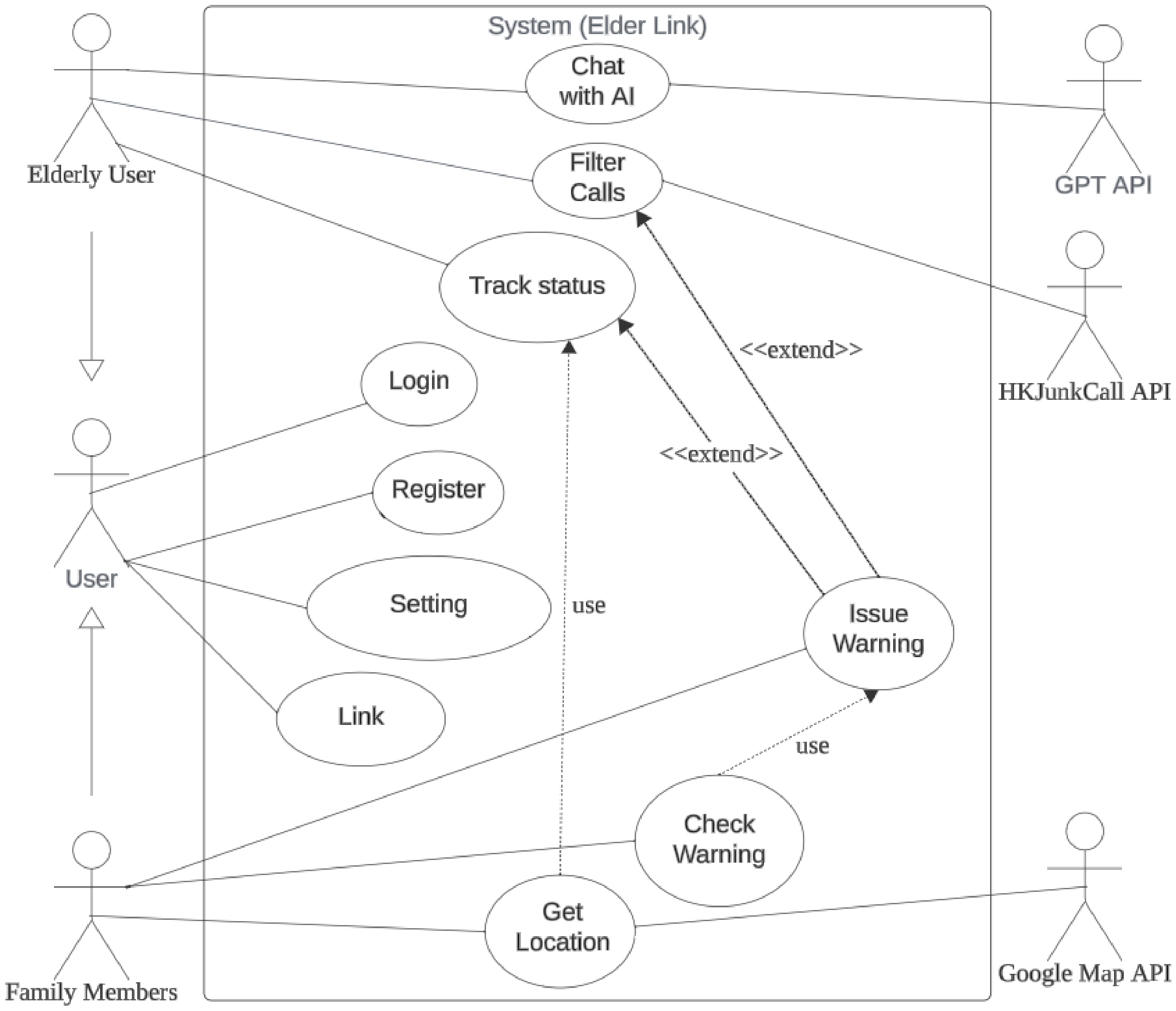


Figure 3 Use Case Diagram of Elder Link

The above diagram shows the use cases of the application. The users are specialized into two types of targeted users, namely elderly user and their family members.

Firstly, all the users are expected to perform the Register, Login, Setting, and Link use cases. Register use case allows users to register an account. Login use case allows users to access the registered account. Setting use case allow users to enable or disable the features in the application according to their own needs and set an upper and lower limit of blood pressure to trigger the body status monitoring mechanism. Link use case allows users to establish connections to other users.

Secondly, the elderly users are expected to perform the Chat with AI, Filter calls, and Track status use cases. Chat with AI use case allows elderly users to send a query to a GPT model through an external GPT API and retrieve its response as if chatting with a real person in a messenger app. Filter calls use case will be triggered automatically when there is an incoming call, and it will identify if the incoming call is suspicious by interacting with HKJunkCall API, which is an API that takes a phone number as input and return the identified type of phone number. If the call is identified suspicious, issue warning use case will be triggered, and a warning will be issued to the linked family members. Track status use case will be triggered constantly for every predefined period of time to get the up-to-date status of the elderly users. The location status will be stored and issue warning use case may be triggered if falling is detected or body status of elderly is abnormal.

Lastly, linked family members are expected to perform the Check warning and Get location use cases. Check warning use case allows family members to check previous issued warnings by elderly users. Get location use case allows family members to retrieve the location of elderly users. It also interacts with Google Map API to visualize the location.

3.2 Execution Logic

This section will focus on the underlying execution logic and sequence for each of the use case. A sequence diagram will be used to visualize and assist the presentation of the execution logic.

3.2.1 Register

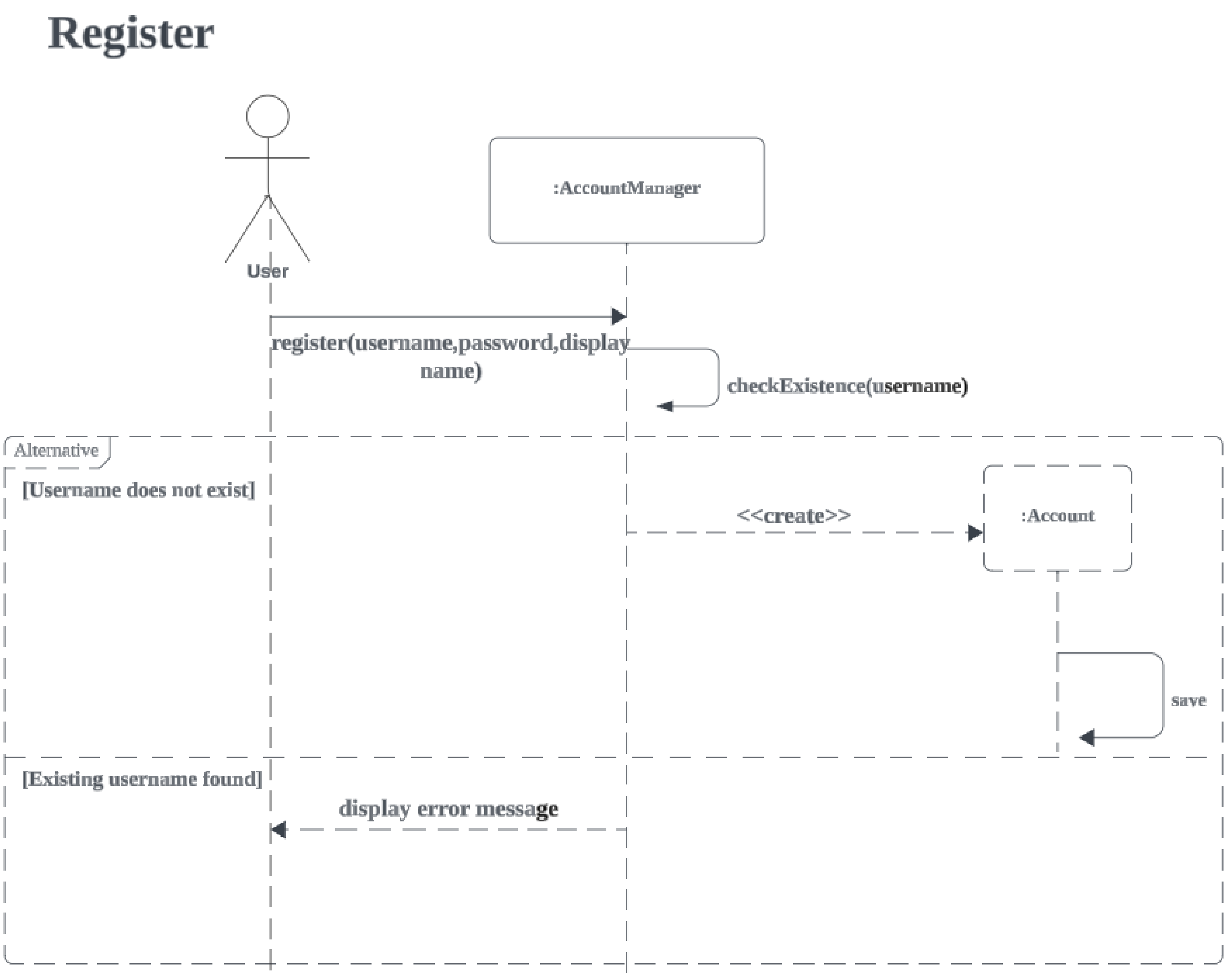


Figure 4 Sequence Diagram of register

Above is the sequence diagram of Register use case. During registration, the user will send the desired username, password and display name (name displayed to other users) to the AccountManager. Then the AccountManager will query the username in the database to check if it exists. If the username does not exist and it is valid, A new Account object is created and saved in the database by the AccountManager. If an existing username is found in the database, the registration is invalid and AccountManager will display an error message in the user interface.

3.2.2 Login

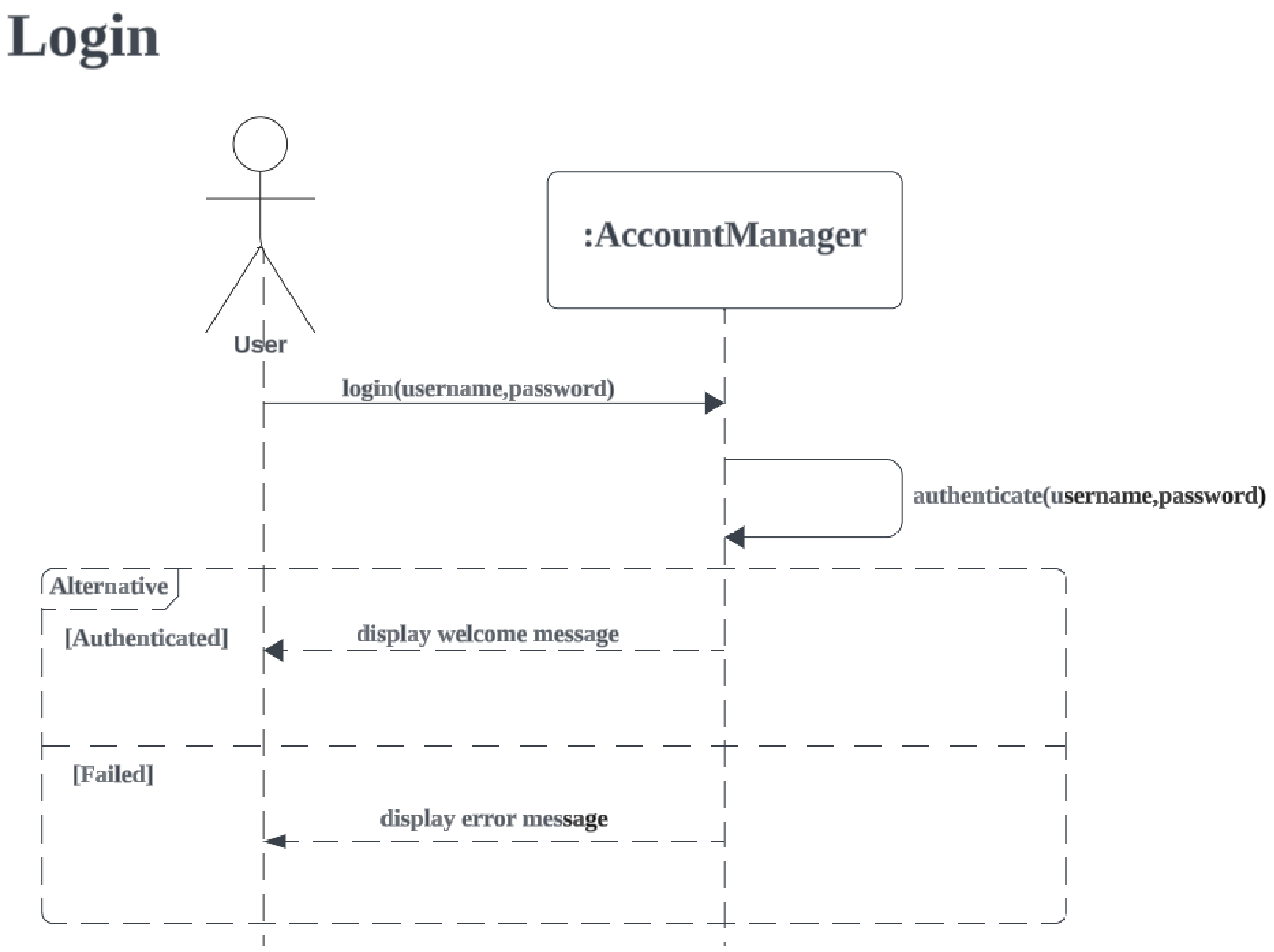


Figure 5 Sequence Diagram of Login use case

Above is the sequence diagram of login use case. During login, user will first send their username and password to the AccountManager. Then the AccountManager will query and authenticate the username and password in the database. If authentication succeeds, AccountManager will display a welcome message to the user. If authentication fails and the login is invalid, AccountManager will display an error message.

3.2.3 Setting

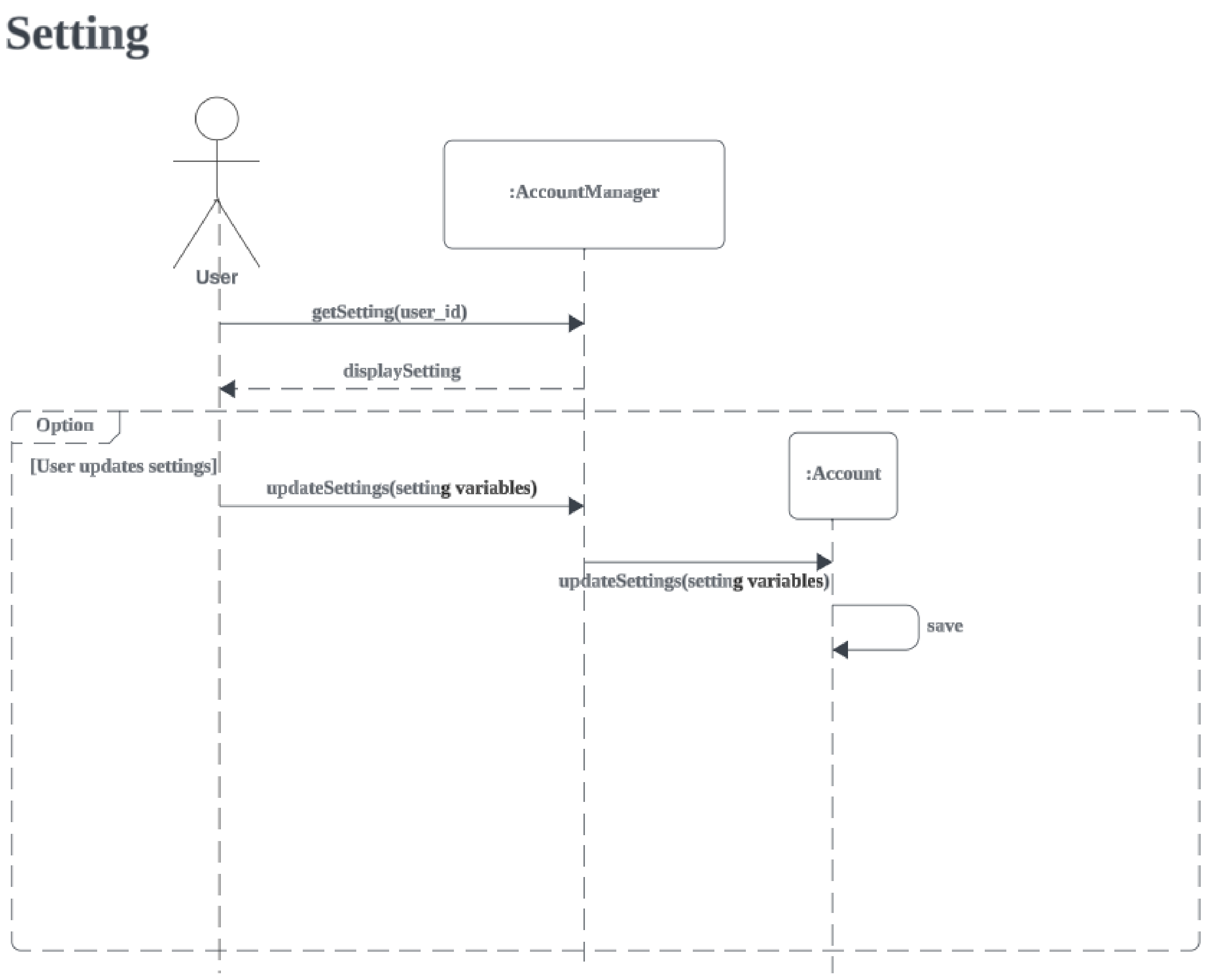


Figure 6 Sequence Diagram of setting use case

Above is the sequence diagram of the setting use case. The user will first retrieve the up-to-date settings from the AccountManager. Then AccountManager will retrieve the Account object in database and display the settings to the user. The user can choose the update the setting. Upon choosing to update the settings, AccountManager will update the Account object with the new set of settings and save it in the database.

3.2.4 Link

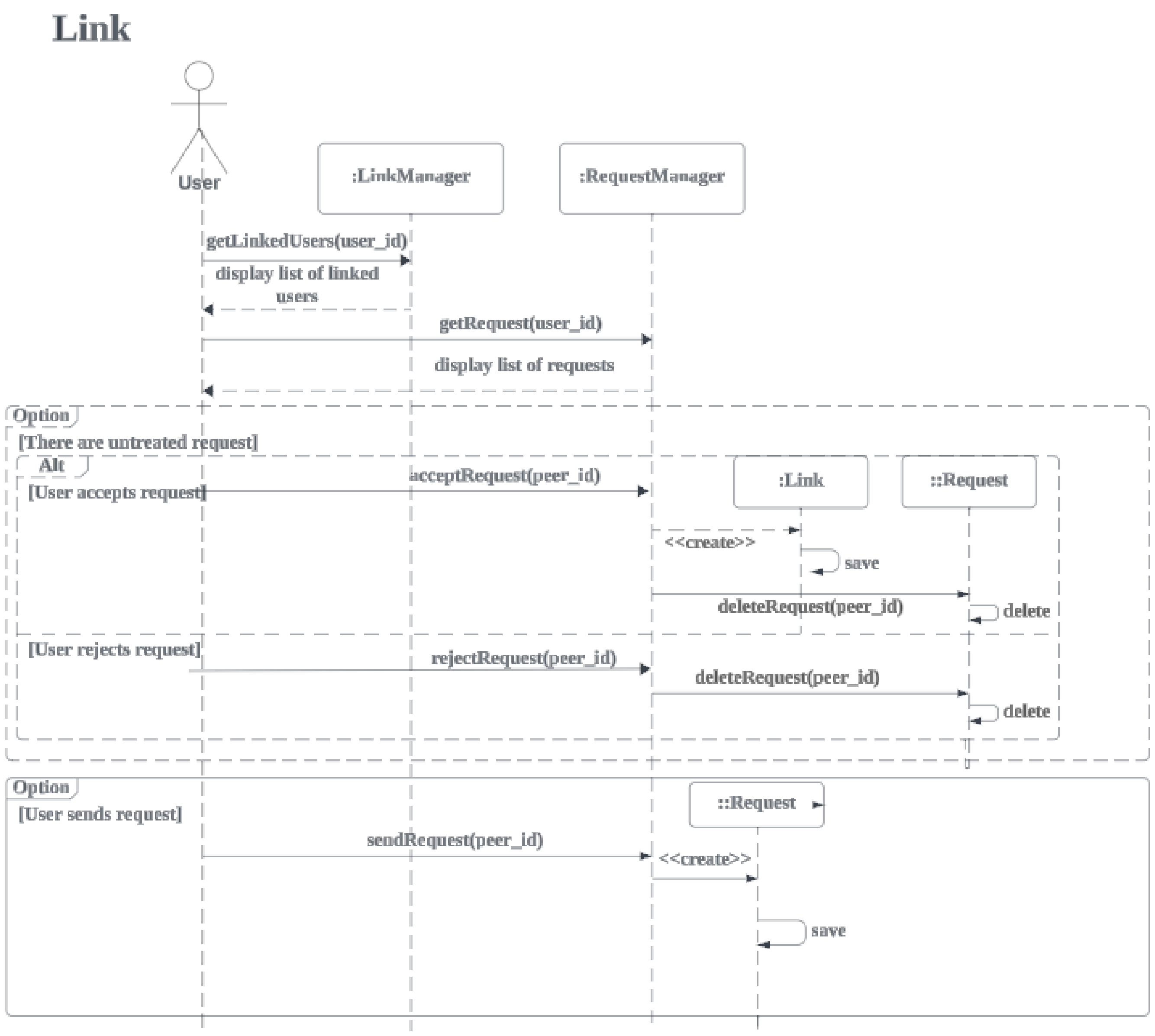


Figure 7 Sequence Diagram of Link

Above is the sequence diagram of Link use case. The user will first get the linked users list and request list from LinkManager and RequestManager. Then LinkManager and RequestManager will display the list of linked users and list of request respectively to the user.

If there are untreated link request, user can choose to accept or reject it. If user chooses to accept, RequestManager will create a new Link object that contains the (user_id,peer_id) pair and save it in the database. Afterwards, it will delete the current request record in the database as a result. If user chooses to reject the request, RequestManager will directly delete the request record in the database.

The user can also choose to send a request to a peer. A new Request object specifying (user_id,peer_id) will then be created and saved in database.

3.2.5 Chat with AI

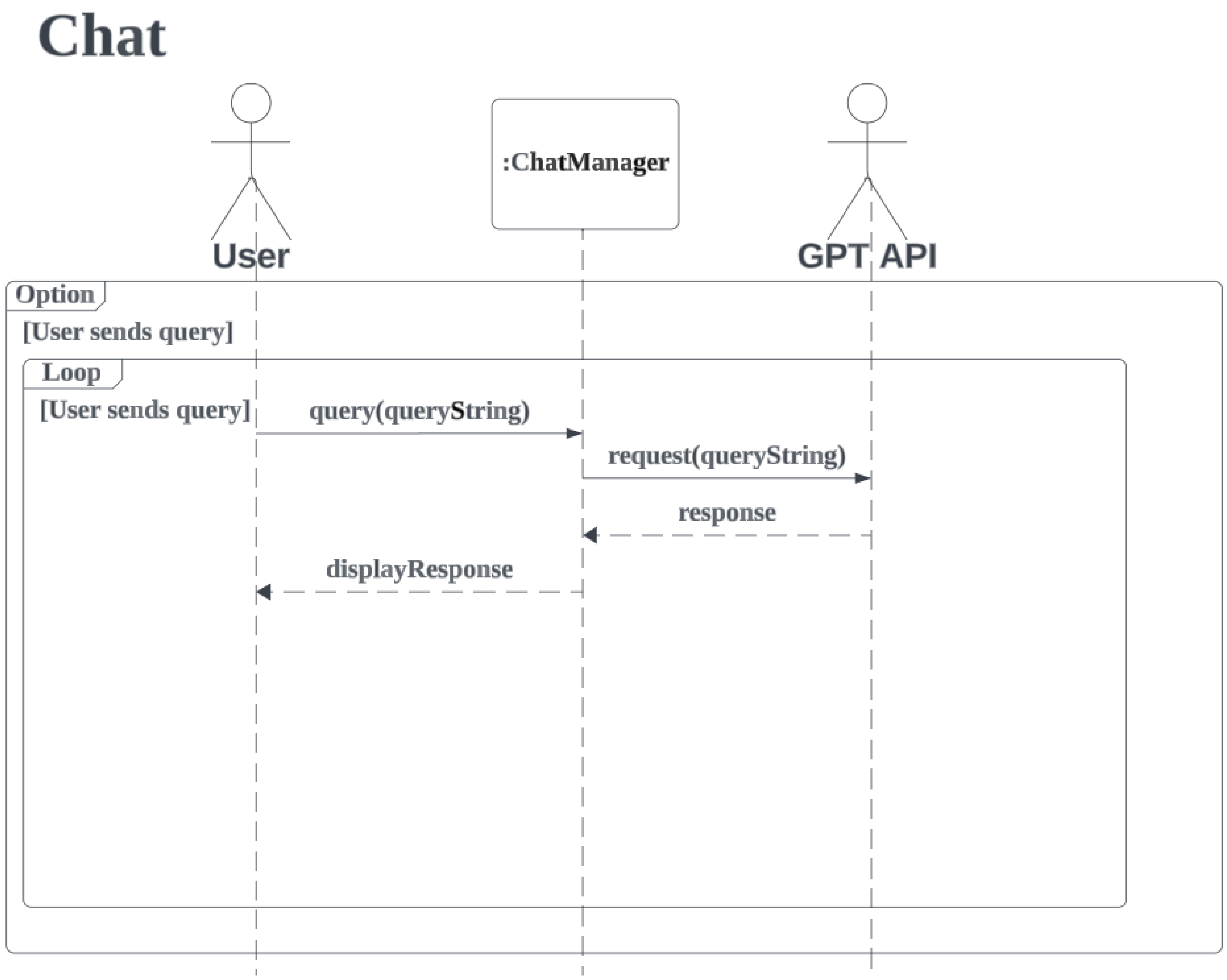


Figure 8 Sequence diagram of Chat with AI use case

Above is the sequence diagram of Chat with AI use case. The user can choose to send a query to the chatbot and every time a query is sent, ChatManager will send a request that contains the query string to the external LLM API and retrieve its response. Upon receiving the response, ChatManager will display the response to the user.

3.2.6 Filter Calls

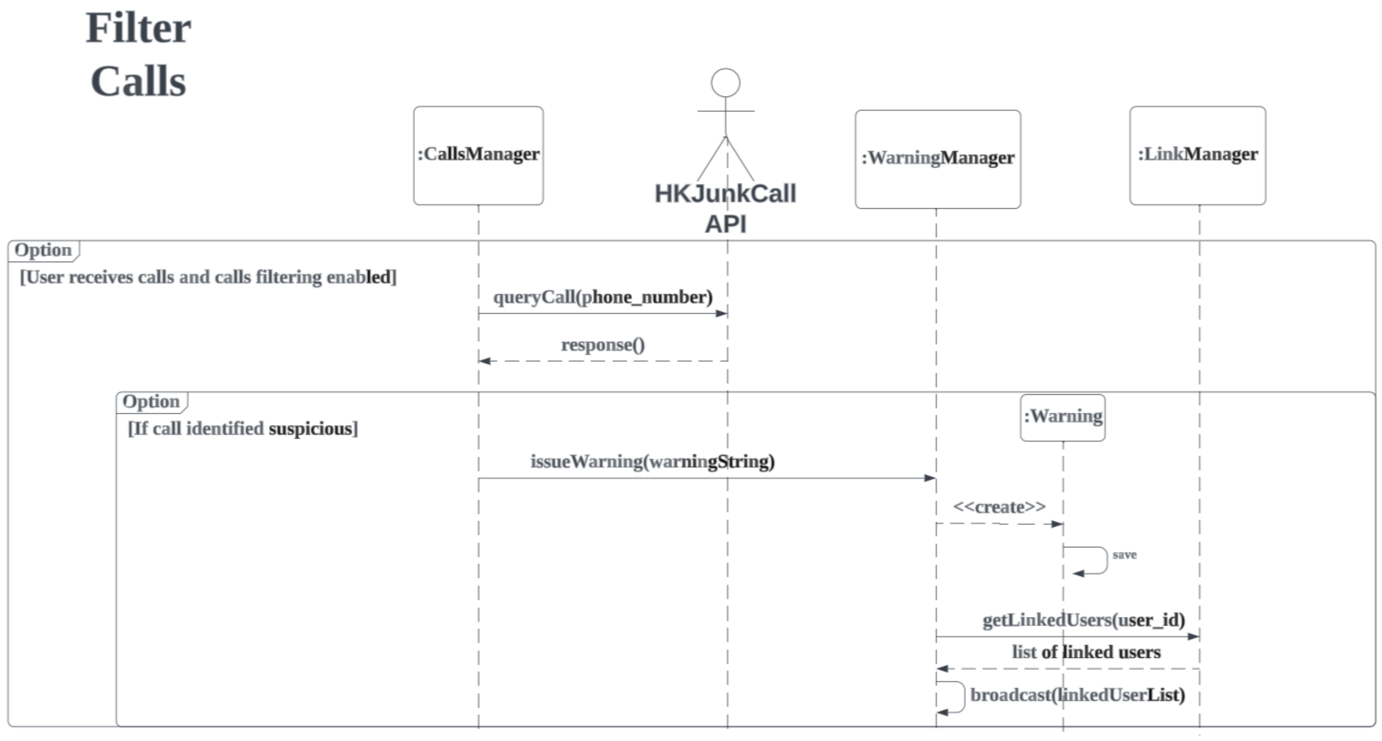


Figure 9 Sequence diagram of Filter Calls use case

Above is the sequence diagram of Filter Calls use case. If the calls filtering feature is enabled, every time there is an incoming phone call, CallManager will query the call via the HKJunkCall API and retrieve the response. If the response indicates that the phone number is suspicious, an warning string will be generated by the CallsManager and WarningManager will be called with the warning string. WarningManager will then first create a Warning object that contains the user_id and warning string and save it in database. After that, it will get the list of linked users from LinkManager and broadcast the warning message to all the linked users.

3.2.7 Track Status

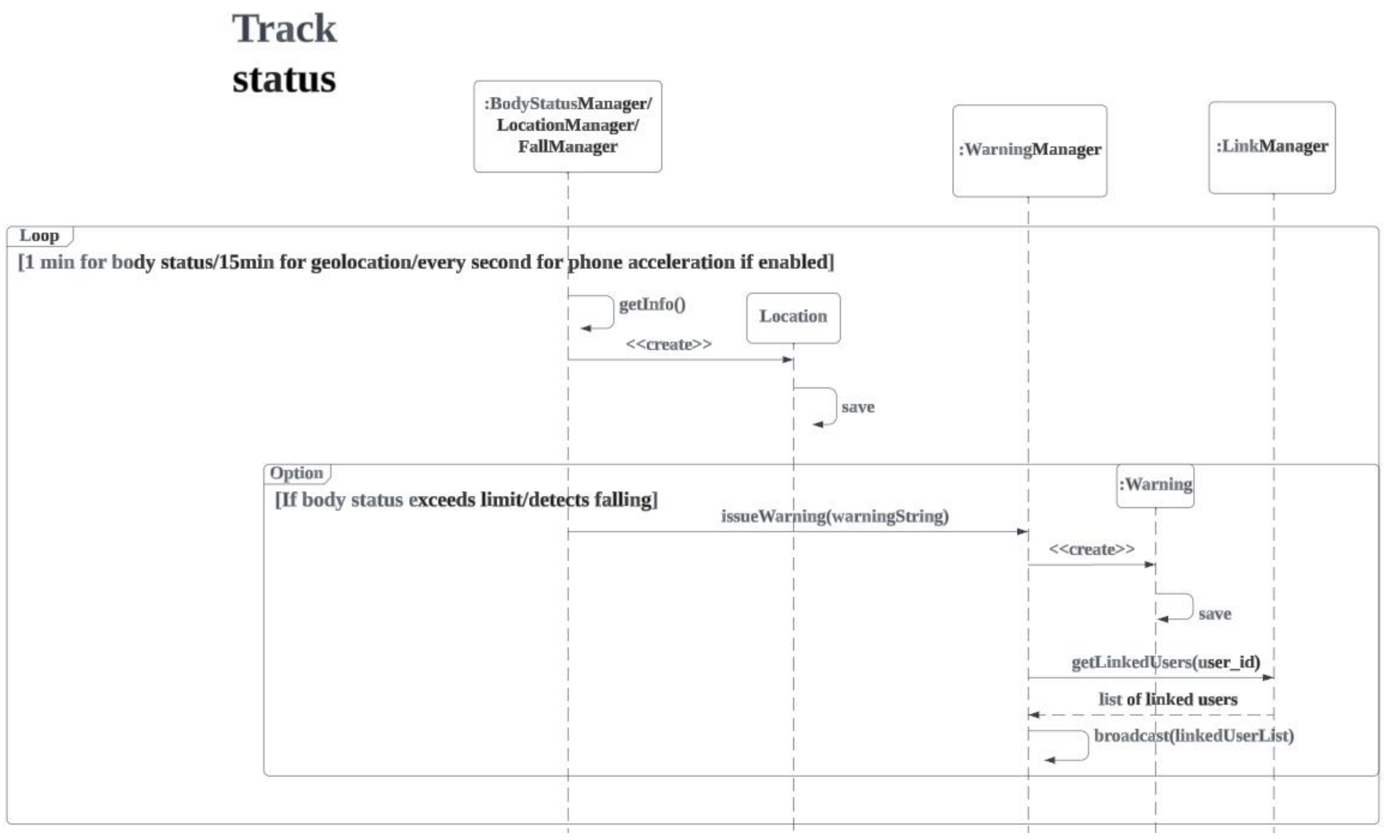

Track status
:BodyStatusManager/ LocationManager/ FallManager
:WarningManager
:LinkManager
Loop
[1 min for body status/15min for geolocation/every second for phone acceleration if enabled]
getInfo()
Location
<<create>>
save
Option
[If body status exceeds limit/detects falling]
:Warning
issueWarning(warningString)
<<create>>
save
getLinkedUsers(user_id)
list of linked users
broadcast(linkedUserList)

Figure 10 Sequence diagram of Track Status use case

Above is the sequence diagram of Track Status use case. If the corresponding feature is enabled, BodyStatusManager will get the body status (blood pressure) for every minute, LocationManager will get the geolocation and store the latest location in database for every 15 minute, while FallManager will get the phone acceleration for every second. If the body status exceeds the predefined limit or a falling is detected, BodyStatusManager and FallManager will generate the corresponding warning string and call WarningManager. The detail of issuing a warning is the same as described above.

3.2.8 Check Warning

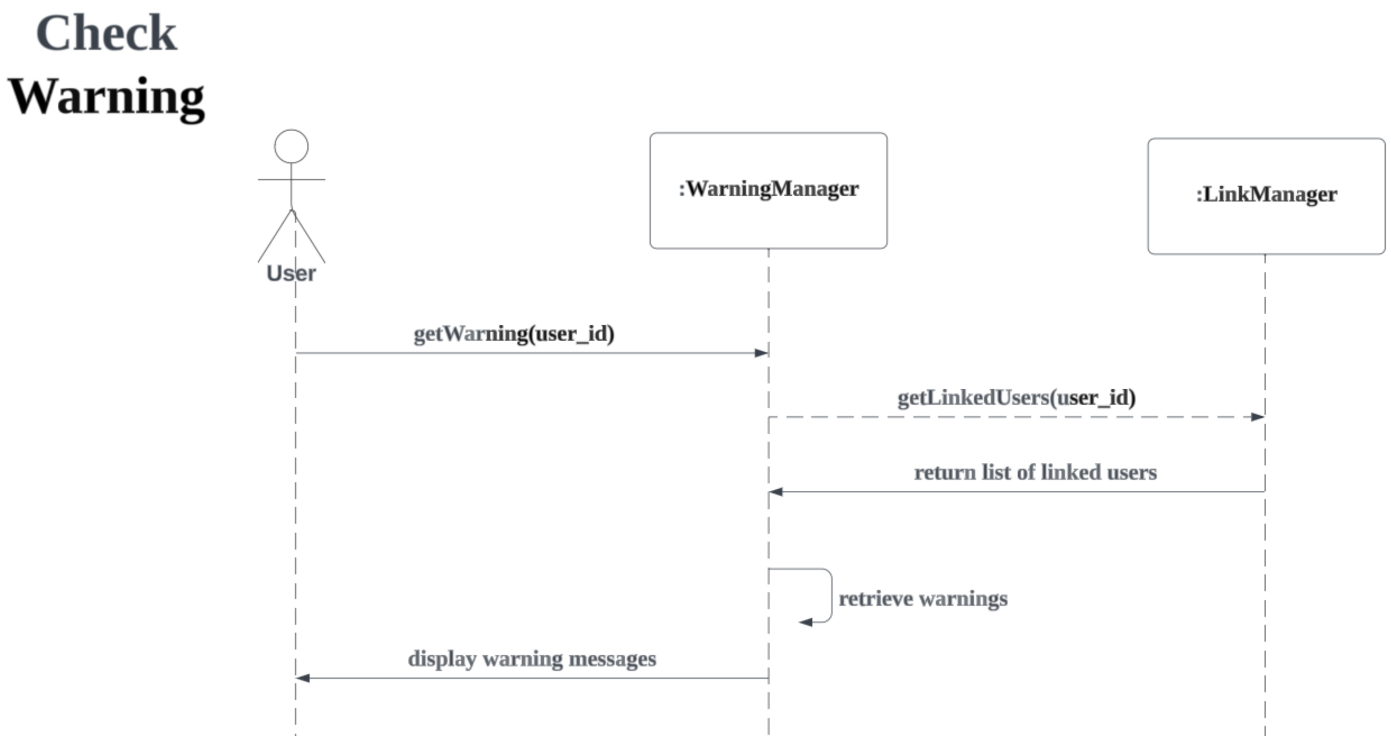


Figure 11 Sequence diagram of Check Warning use case

Above is the sequence diagram of the Check Warning use case. User can get previous issued warnings from WarningManager. WarningManager will then get the list of linked user from LinkManager and retrieve the associated warnings in the database, finally display it to the user.

3.2.9 Get Location

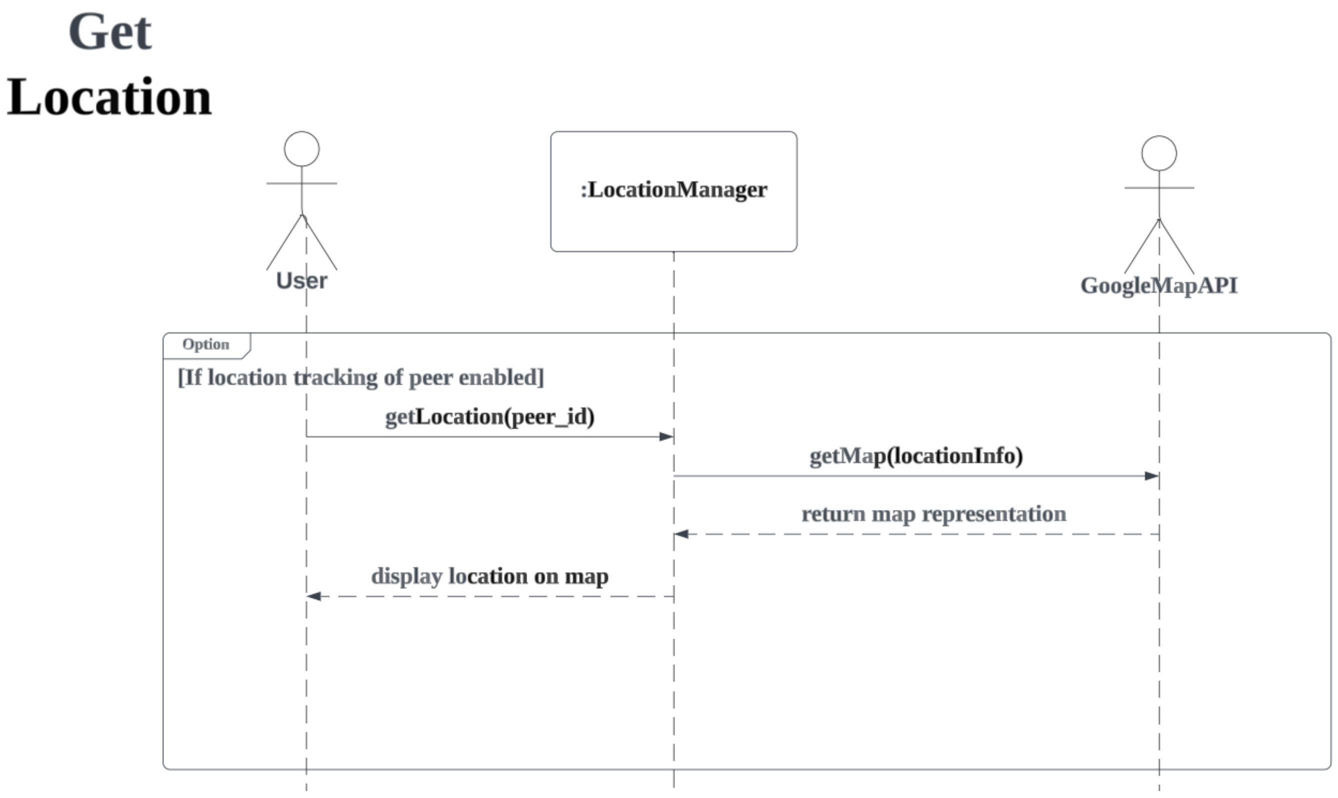


Figure 12 Sequence diagram of Get Location use case

Above is the sequence diagram of Get Location use case. If the location tracking feature of the peer is enabled, the user can get the location of the peer from LocationManager. The LocationManager will then retrieve the location information of the peer in the database and get the map representation by calling the GoogleMapAPI. Finally the LocationManager will display the location visualized on the map to the user.

3.3 Classes involved in the application

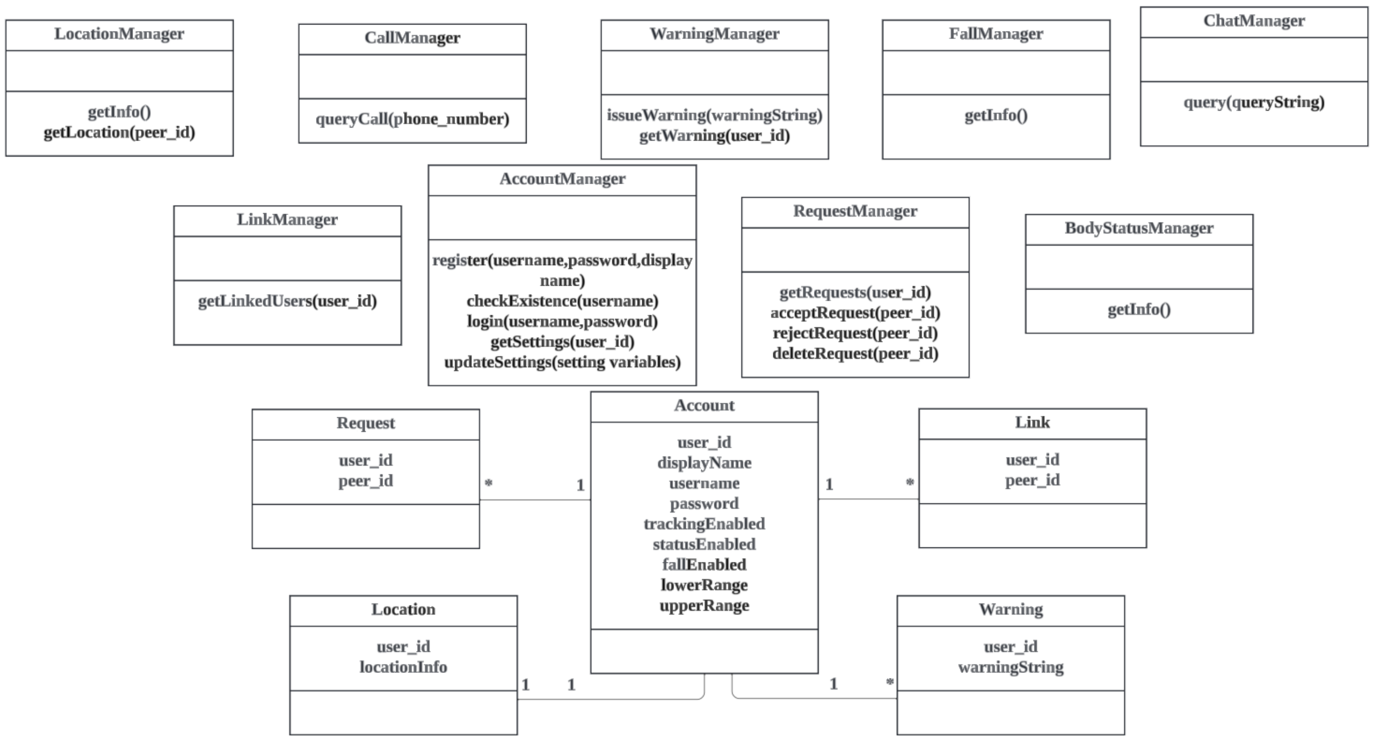


Figure 13 Class diagram of Elder Link

Above is the class diagram of the application. It is a summary of the managers and objects involved in the execution logic (shown by previous sequence diagram). Managers are classes that play a specific role and are responsible for a specific component or functionality of the system. Other classes are used to organize the required information or data related to a specific functionality. The class diagram can be used to set up classes efficiently during the development process.

3.4 Database Design

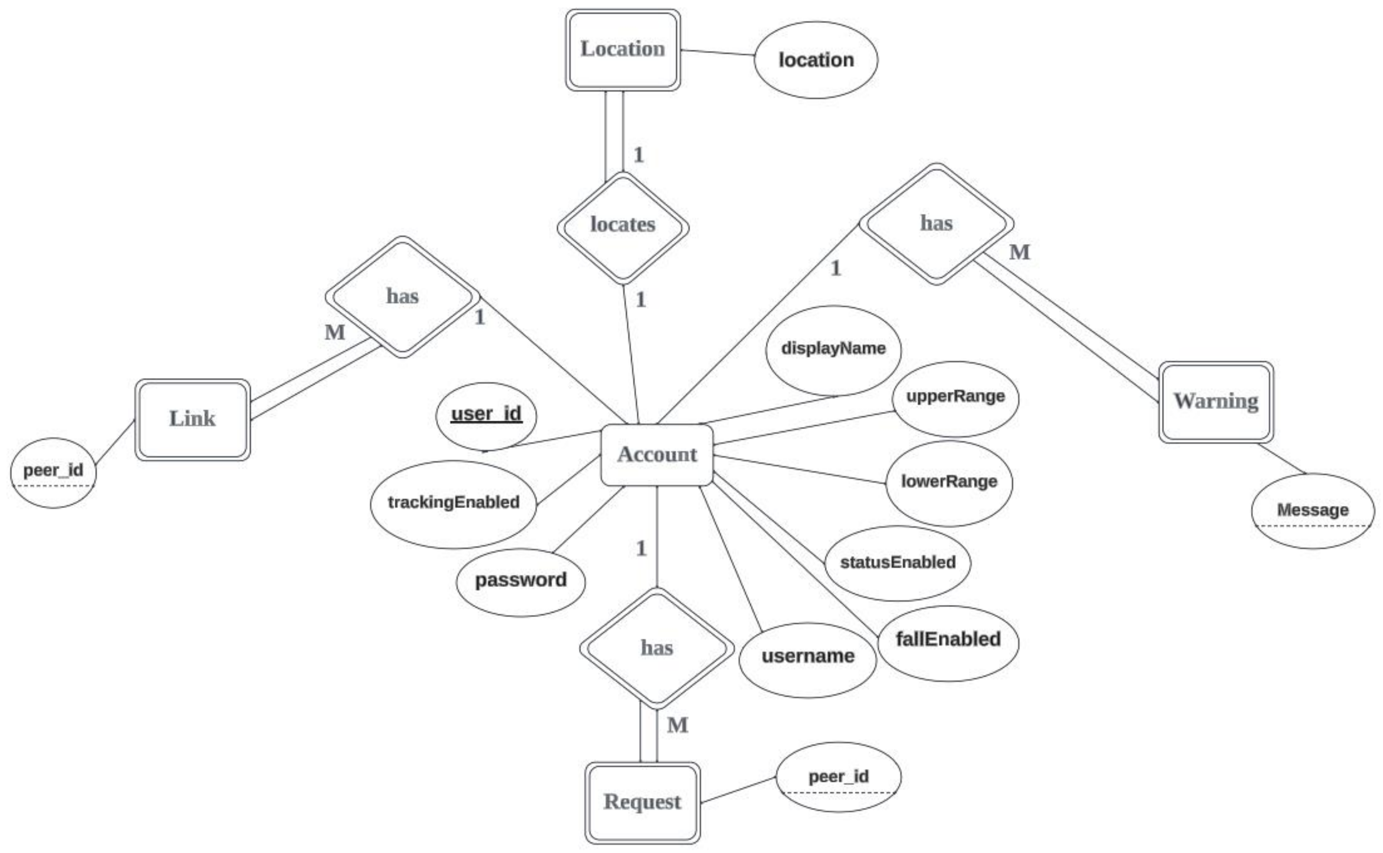


Figure 14 Entity Relationship diagram of Elder Link's database

Above is the entity relationship diagram of the application's database, which outlines the tables and their columns to be stored in the database as well as the relationship between tables.

Account entity has 9 attributes, including a primary key user_id; username, password, and display name that are strings; trackingEnabled, statusEnabled, and fallEnabled that are Booleans; as well as upperRange and lowerRange (for blood pressure) that are intergers. Account has an one to one relationship with Location, which means one account has at most one current location. It has an one to many relationship with Link, Warning and Request, which means that an account can create multiple links, issue multiple warnings and involved in multiple requests. All the participations in the relationships are partial, which means it is not necessary for an account to have any location, link, warning and request.

Location entity, which is a weak entity, has 2 attributes, including user_id (foreign key from Account) and location that is represented in a string. Location has a one to one relationship with Account, which means that each location can only associate with one account. The participation is total, which means that when every location must be linked to an account.

Link entity, which is a weak entity, has 2 attributes, including user_id (foreign key from Account) and peer_id that is an integer. Link has a many to one relationship with Account, which means that each link can only involve at most two accounts but not more. The participation is total, which means that when every link must be linked to two accounts.

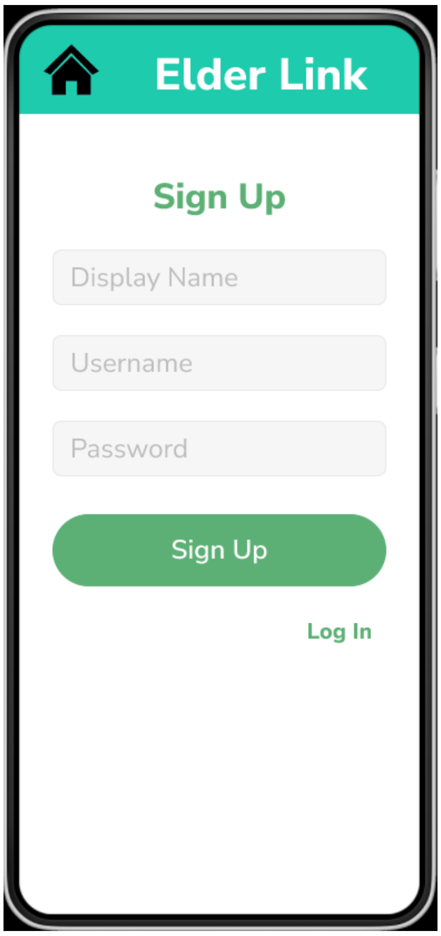
Warning entity, which is a weak entity, has 2 attributes, including user_id (foreign key from Account) and warning message that is a string. Warning has a many to one relationship with Account, which means that each warning can only associate with one account. The participation is total, which means that when every warning must be linked to one account.

Request entity, which is a weak entity, has 2 attributes, including user_id (foreign key from Account) and peer id that is an integer. Request has a many to one relationship with Account, which means that each request can only associate with two accounts but not more. The participation is total, which means that when every request must be linked to two accounts.

3.5 User Interface Design

This section will focus on the Figma user interface design of different pages in the application.

3.5.1 Login and Register Pages



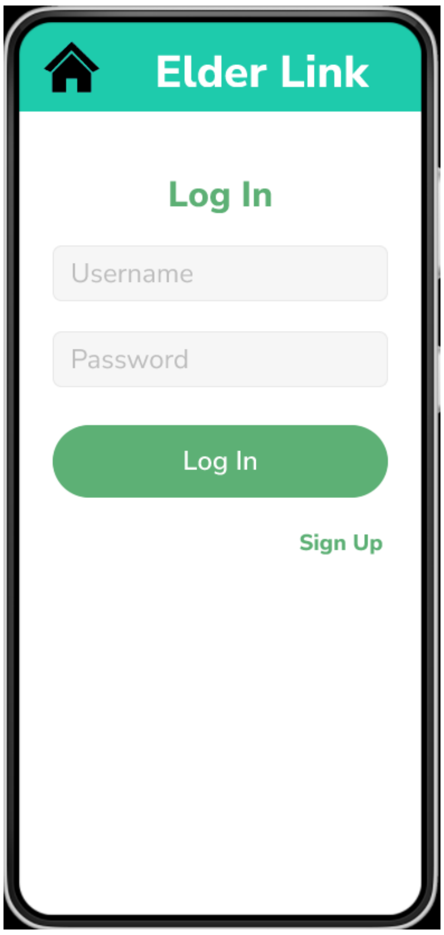


Figure 15 & 16 Register Page and Login Page

Above are the register and login pages of the application. In the Register page, user can fill in the display name, username, and password, then click the sign up button to submit their registrations. In login page, user can fill in their username and password, then click the login button to authenticate and log in.

3.5.2 Main Page

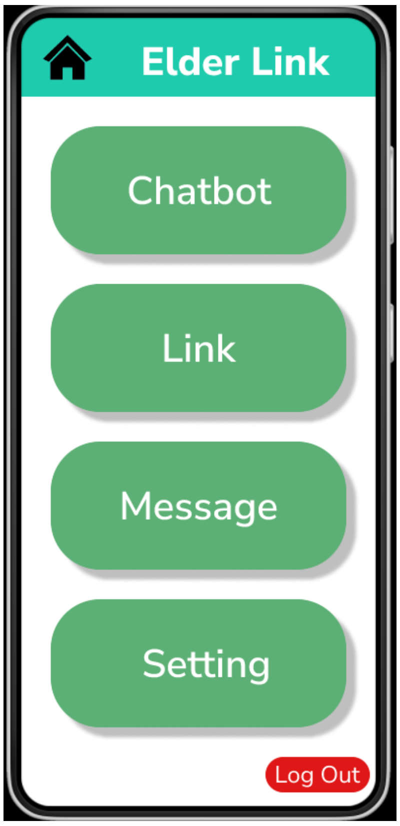


Figure 17 Main Page

Above is the main page of the application. It consists of navigation to chatbot, link, message, and setting pages, as well as a log out button which will navigate back to the login page.

3.5.3 Chatbot Page

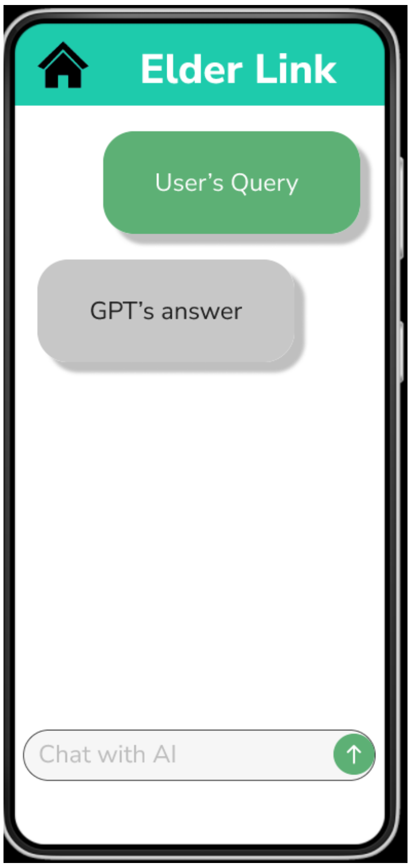


Figure 18 Chatbot Page

Above is the chatbot page of the application. User can type in their queries in the textbox and click on the button to submit their queries. The queries and responses will be displayed as if messages in messenger apps.

3.5.4 Link Pages

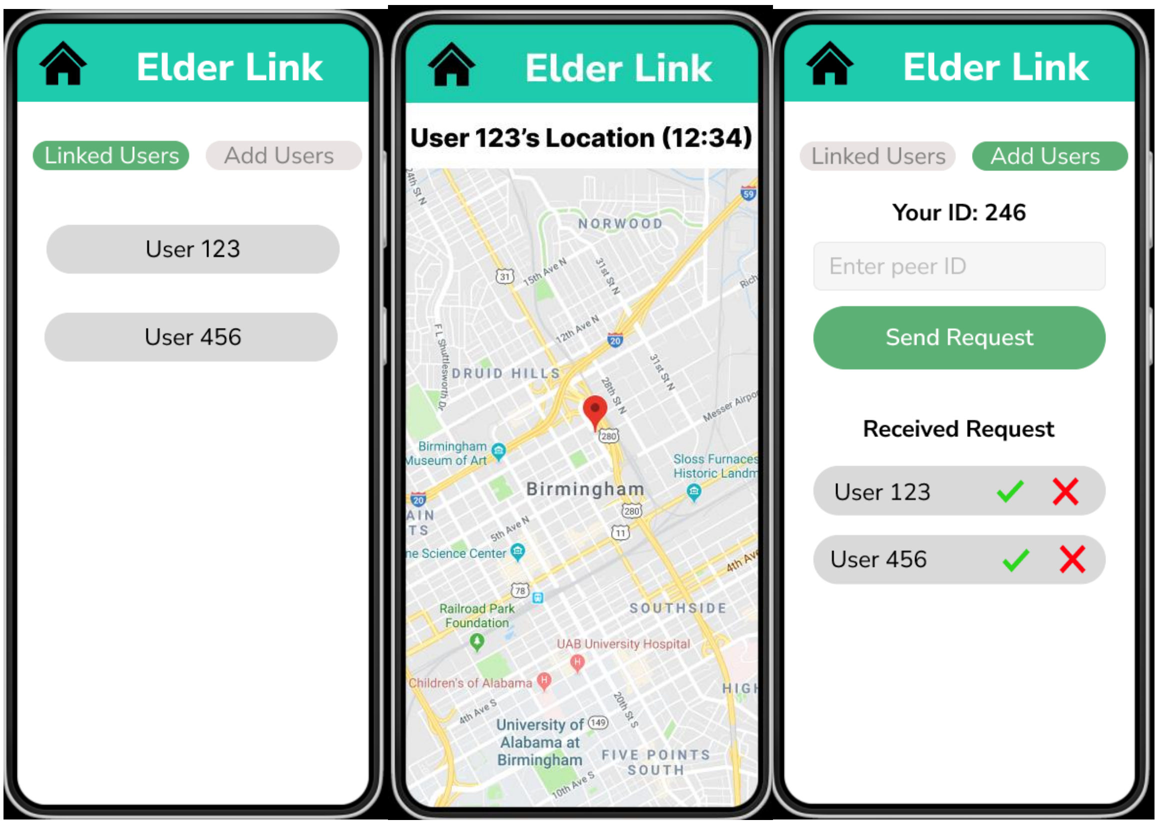
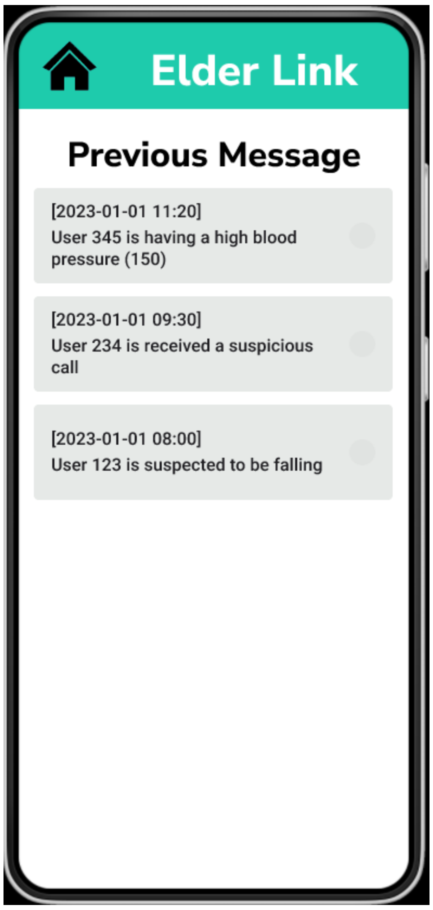


Figure 19 & 20 & 21 Link Pages

Above are pages in the Link component of the application. In the first page, the list of linked users will be displayed. If any of the linked users is clicked, it will enter the second page, which displays the location of the selected linked user. If the Add Users button in the first page is clicked, the third page will be shown. The third page displays the user id of the user, a text box and a Send Request button for user to enter their peer's id and send a request. There is also an area that displays all received requests and allows users to accept and reject the requests by clicking the tick or cross buttons.

3.5.5 Previous Message Page



[2023-01-01 11:20]
User 345 is having a high blood pressure (150)

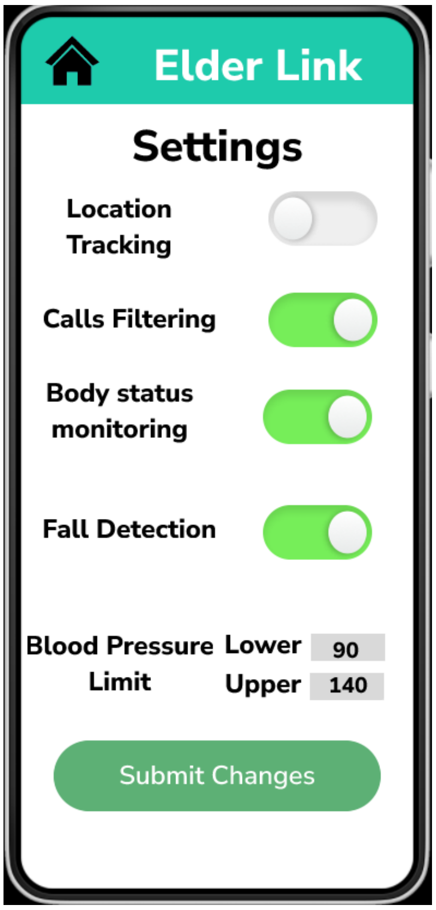
[2023-01-01 09:30]
User 234 is received a suspicious call

[2023-01-01 08:00]
User 123 is suspected to be falling

Figure 22 Previous Message Page

Above is the previous message page which displays all the previous warning messages issued by linked users.

3.5.6 Setting Page

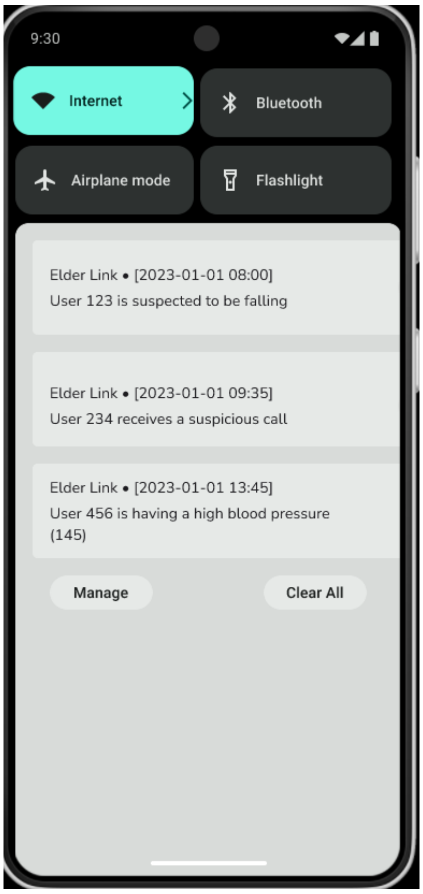


Location Tracking: off
Calls Filtering: on
Body status monitoring: on
Fall Detection: on
Blood Pressure Limit Lower 90
Upper 140

Figure 23 Setting Page

Above is the setting page of the application. It consists of toggle buttons to enable or disable the features, text boxes to enter the lower and upper limit of blood pressure, as well as a summit button to save the changed to database.

3.5.7 Notification



Elder Link • [2023-01-01 08:00]
User 123 is suspected to be falling

Elder Link • [2023-01-01 09:35]
User 234 receives a suspicious call

Elder Link • [2023-01-01 13:45]
User 456 is having a high blood pressure (145)

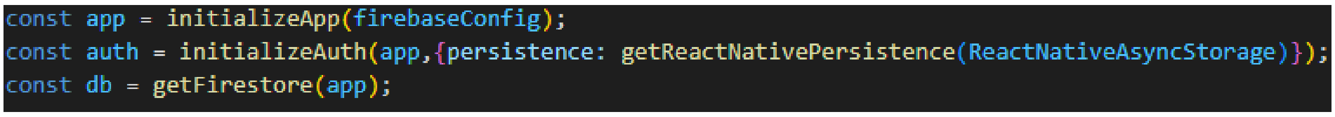
Figure 24 Notification

Above is the sample notification when a linked user triggers any of the warning.

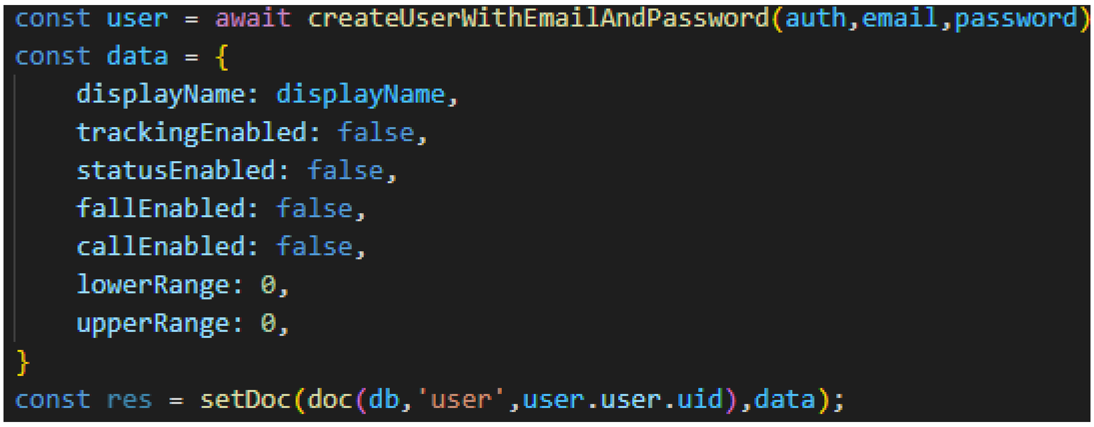
3.6 Implementation

This section will focus on the implementation details of the application, including the services or libraries used to achieve the features. Sections 3.6.1 to 3.6.6 will describe the implementation details of Authentication and database, LLM integration, location tracking, fall detection, deceptive calls detection, and body status monitoring, respectively.

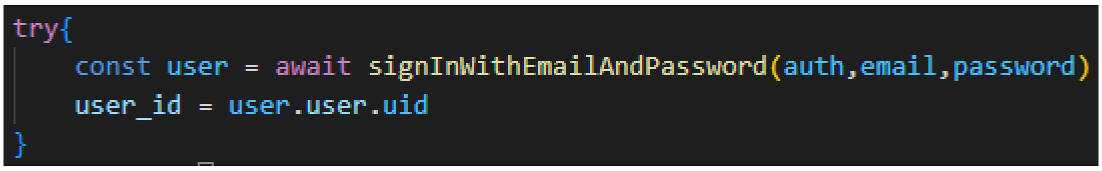
3.6.1 Authentication and database



```
const app = initializeApp(firebaseConfig);
const auth = initializeAuth(app,{persistence: getReactNativePersistence(ReactNativeAsyncStorage)});
const db = getFirestore(app);
```



```
const user = await createUserWithEmailAndPassword(auth,email,password)
const data = {
    displayName: displayName,
    trackingEnabled: false,
    statusEnabled: false,
    fallEnabled: false,
    callEnabled: false,
    lowerRange: 0,
    upperRange: 0,
}
const res = setDoc(doc(db,'user',user.user.uid),data);
```



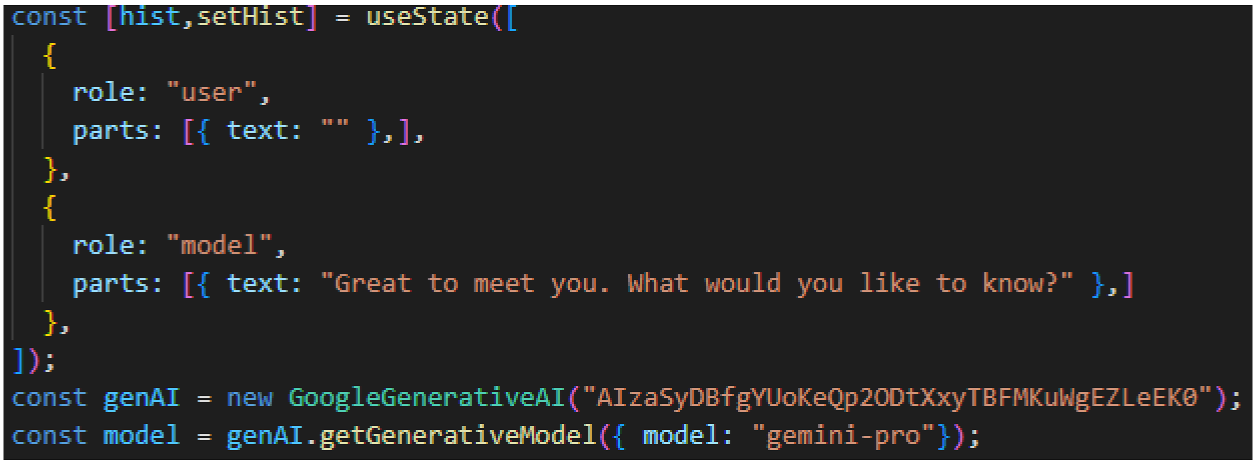
```
try{
    const user = await signInWithEmailAndPassword(auth,email,password)
    user_id = user.user.uid
}
```

Figure 25& Figure 26& Figure 27 Implementation of authentication and database using firebase

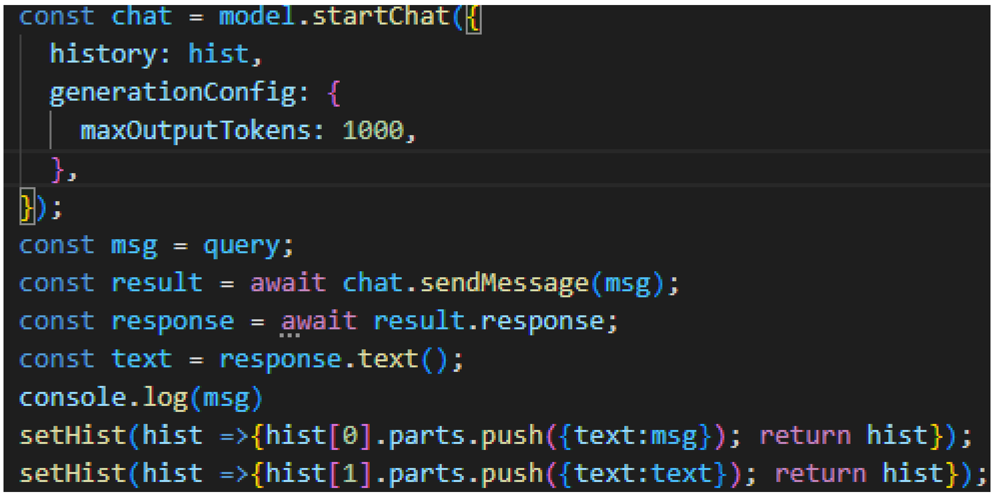
To use Firebase for authentication and database purposes, a Firebase app instance must be initialized with the project credentials. Once the Firebase app instance is initialized, authentication and database instances can be initiated based on the Firebase app instance. This allows for the use of methods such as `createUserWithEmailAndPassword` and `signInWithEmailAndPassword` from the Firebase authentication instance, which can be called to register and authenticate user accounts using email and password credentials.

For reading and writing to the database, the `getDoc` and `setDoc` methods of the Firebase database instance can be used. These methods require the collection name and document name as parameters, allowing access to the NoSQL, key-value based Firestore database. [18]

3.6.2 LLM integration



```javascript
const [hist,setHist] = useState([
  {
    role: "user",
    parts: [{ text: "" },],
  },
  {
    role: "model",
    parts: [{ text: "Great to meet you. What would you like to know?" },]
  },
]);
const genAI = new GoogleGenerativeAI("AIzaSyDBfgYUoKeQp2ODtXxyTBFMKuWgEZLeEK0");
const model = genAI.getGenerativeModel({ model: "gemini-pro"});
```



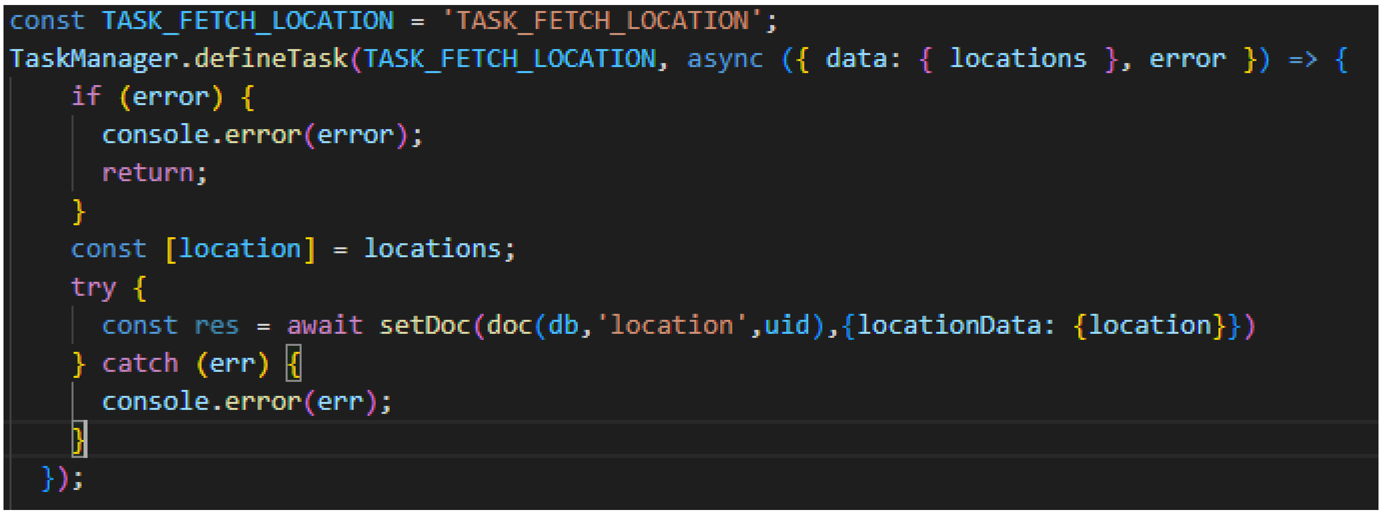
```javascript
const chat = model.startChat({
  history: hist,
  generationConfig: {
    maxOutputTokens: 1000,
  },
});
const msg = query;
const result = await chat.sendMessage(msg);
const response = await result.response;
const text = response.text();
console.log(msg)
setHist(hist =>{hist[0].parts.push({text:msg}); return hist});
setHist(hist =>{hist[1].parts.push({text:text}); return hist});
```

Figure 28 & Figure 29 Implementation of GPT integration

The chosen LLM model for this project is Google Gemini, primarily selected for its strong performance and fast inference time. In order to utilize the Gemini model, a GoogleGenerativeAI instance needs to be initialized with the appropriate API key. Subsequently, an instance of the chosen model, specifically Gemini-Pro in this case, should be initialized.

To engage the model in multi-turn conversation mode (chat), it is necessary to maintain a chat history. This chat history, along with generation configurations such as the maximum output tokens, will be used to initialize a chat instance. By using the `sendMessage` method of the chat instance, queries can be made to the Language Model (LLM), and the response will be returned accordingly. [19]

3.6.3 Location Tracking


```
const TASK_FETCH_LOCATION = 'TASK_FETCH_LOCATION';
TaskManager.defineTask(TASK_FETCH_LOCATION, async ({ data: { locations }, error }) => {
    if (error) {
      console.error(error);
      return;
    }
    const [location] = locations;
    try {
      const res = await setDoc(doc(db,'location',uid),{locationData: {location}})
    } catch (err) {
      console.error(err);
    }
  });
```

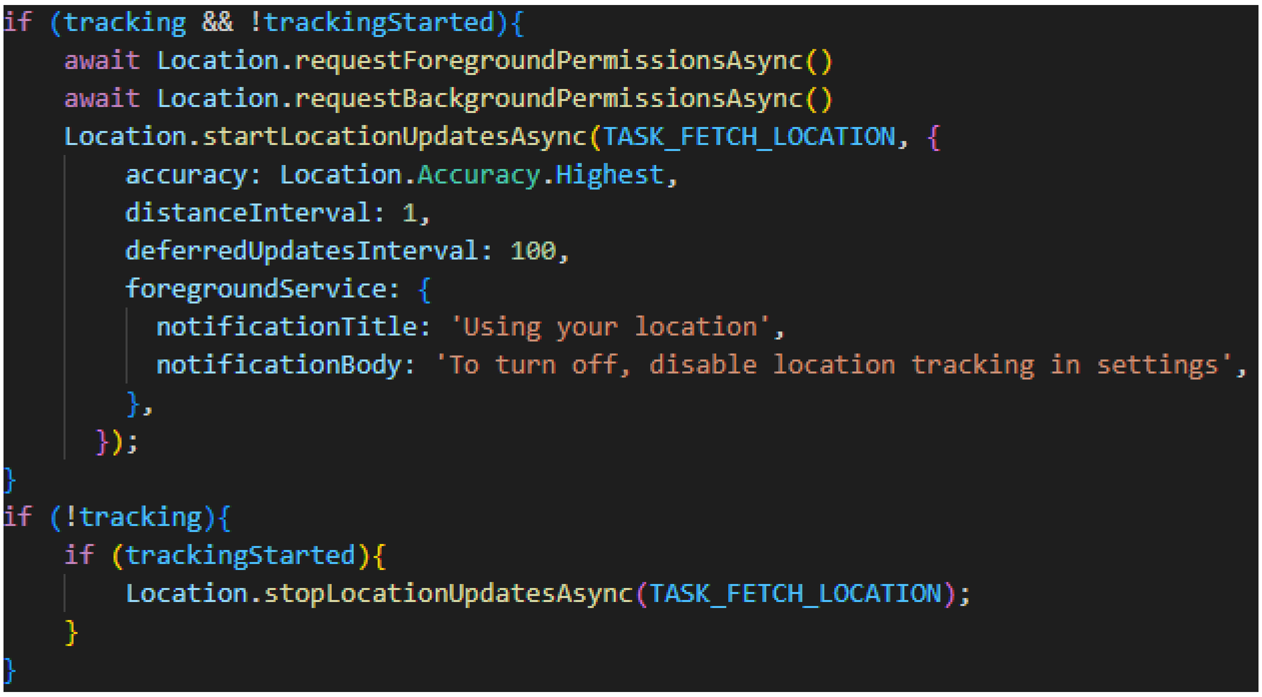

```
if (tracking && !trackingStarted){
    await Location.requestForegroundPermissionsAsync()
    await Location.requestBackgroundPermissionsAsync()
    Location.startLocationUpdatesAsync(TASK_FETCH_LOCATION, {
        accuracy: Location.Accuracy.Highest,
        distanceInterval: 1,
        deferredUpdatesInterval: 100,
        foregroundService: {
          notificationTitle: 'Using your location',
          notificationBody: 'To turn off, disable location tracking in settings',
        },
      });
}
if (!tracking){
    if (trackingStarted){
        Location.stopLocationUpdatesAsync(TASK_FETCH_LOCATION);
    }
}
```

Figure 30 & Figure 31 Implementation of Location Tracking

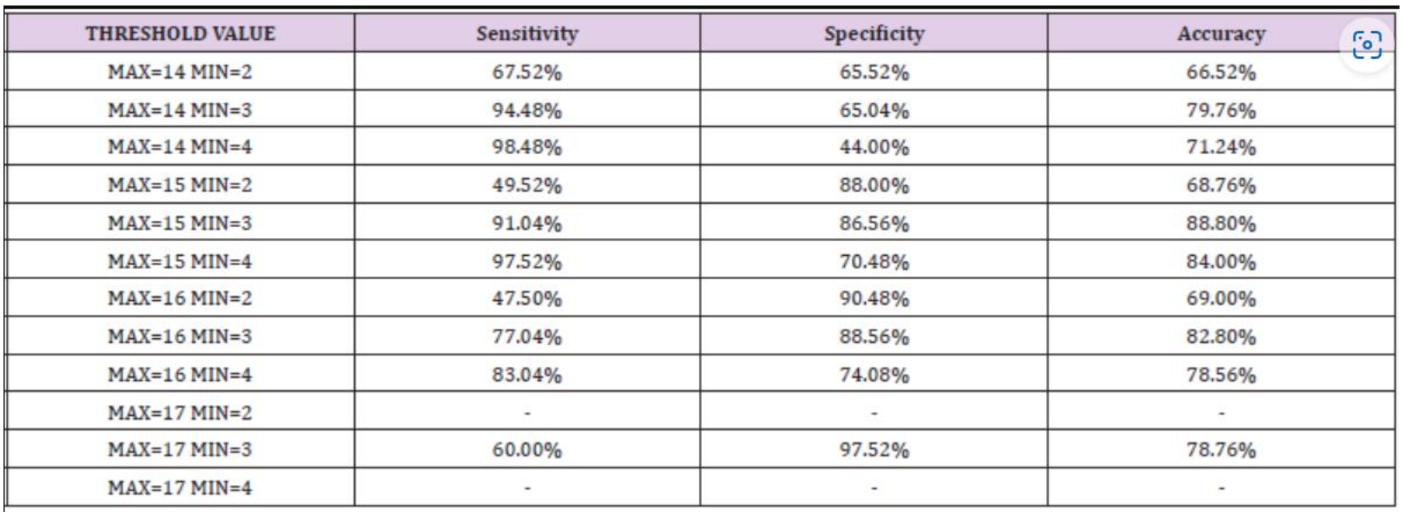
To enable background location tracking, the first step is to define a task in the TaskManager within React Native. This task will execute the specified callback function to handle the retrieval of location information. In this case, the callback function is responsible for uploading the location data to the database.

In order to constantly track the location of users who have enabled this feature, it is necessary to obtain permissions for retrieving location information beforehand. Once the necessary permissions are granted, the `startLocationUpdatesAsync` method from the 'expo-location' library can be used. This method allows for configuration of parameters such as accuracy level, minimum distance, or time interval that would trigger an update. If an update is triggered, the predefined callback function will be called asynchronously. [20]

3.6.4 Fall Detection

$$\text{Signal Magnitude Vector} = \sqrt{|A_x|^2 + |A_y|^2 + |A_z|^2} \quad (1)$$

Figure 32 Formula to calculate signal magnitude vector from accelerometer reading



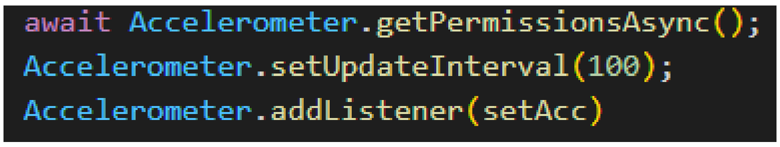
THRESHOLD VALUE	Sensitivity	Specificity	Accuracy
MAX=14 MIN=2	67.52%	65.52%	66.52%
MAX=14 MIN=3	94.48%	65.04%	79.76%
MAX=14 MIN=4	98.48%	44.00%	71.24%
MAX=15 MIN=2	49.52%	88.00%	68.76%
MAX=15 MIN=3	91.04%	86.56%	88.80%
MAX=15 MIN=4	97.52%	70.48%	84.00%
MAX=16 MIN=2	47.50%	90.48%	69.00%
MAX=16 MIN=3	77.04%	88.56%	82.80%
MAX=16 MIN=4	83.04%	74.08%	78.56%
MAX=17 MIN=2	-	-	-
MAX=17 MIN=3	60.00%	97.52%	78.76%
MAX=17 MIN=4	-	-	-

Figure 33 Measured values of sensitivity, specificity and accuracy for different values of the thresholds.

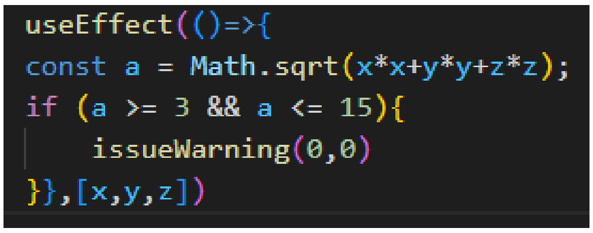
The implementation of fall detection algorithms is based on the research article titled "Multi-Sensor Fall Detection for Smartphones". In this article, various falling scenarios were simulated and different sensor reading thresholds were applied to detect falling events.

The Signal Magnitude Vector (SMV) is calculated using the formula shown in Figure 32. This value represents the square root of the sum of squares of the x, y, and z readings of the accelerometer. The SMV is then used to identify a probable fall event by deriving acceleration peak thresholds.

Figure 33 contains the measured values of sensitivity, specificity, and accuracy for different acceleration peak thresholds. Among all the thresholds, MAX=15 and MIN=3 were chosen due to their acceptable performance, with performance metrics exceeding 85% in all cases.[21]



```
await Accelerometer.getPermissionsAsync();
Accelerometer.setUpdateInterval(100);
Accelerometer.addListener(setAcc)
```

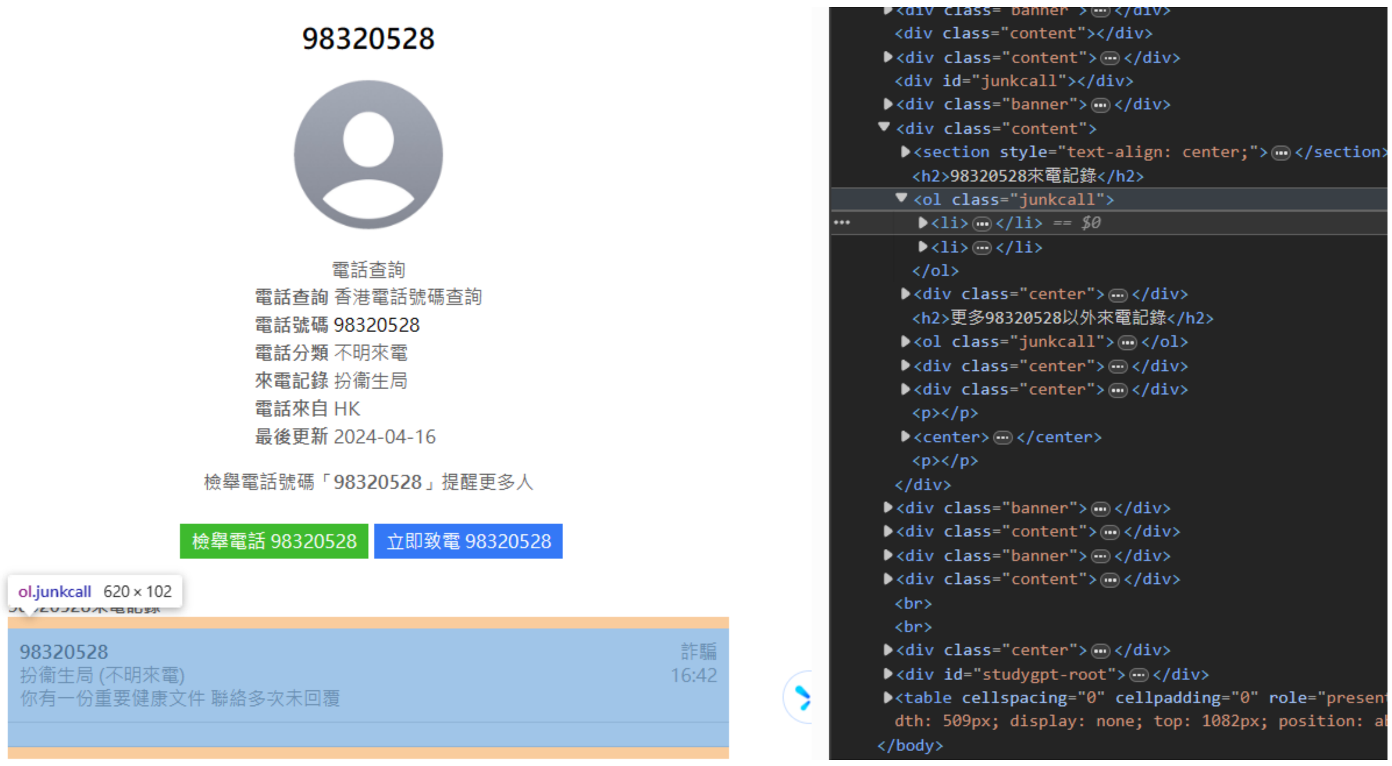


```
useEffect(()=>{
const a = Math.sqrt(x*x+y*y+z*z);
if (a >= 3 && a <= 15){
    issueWarning(0,0)
}},[x,y,z])
```

Figure 34 & 35 code to track the accelerometer value and detect falling

Above is the code that uses the `Accelerometer` class from the 'expo-sensors' library to request permission from the user to access the accelerometer, set up a time interval for each update, and attach the updated readings to variables. Whenever there is an update on the variables, the Signal Magnitude Vector (SMV) is computed using the formula from Figure 32. If the SMV falls between the predefined interval, a warning is issued to notify the user of a potential fall event. [22]

3.6.5 Deceptive Calls Detection



98320528

電話查詢
電話查詢 香港電話號碼查詢
電話號碼 98320528
電話分類 不明來電
來電記錄 扮衛生局
電話來自 HK
最後更新 2024-04-16

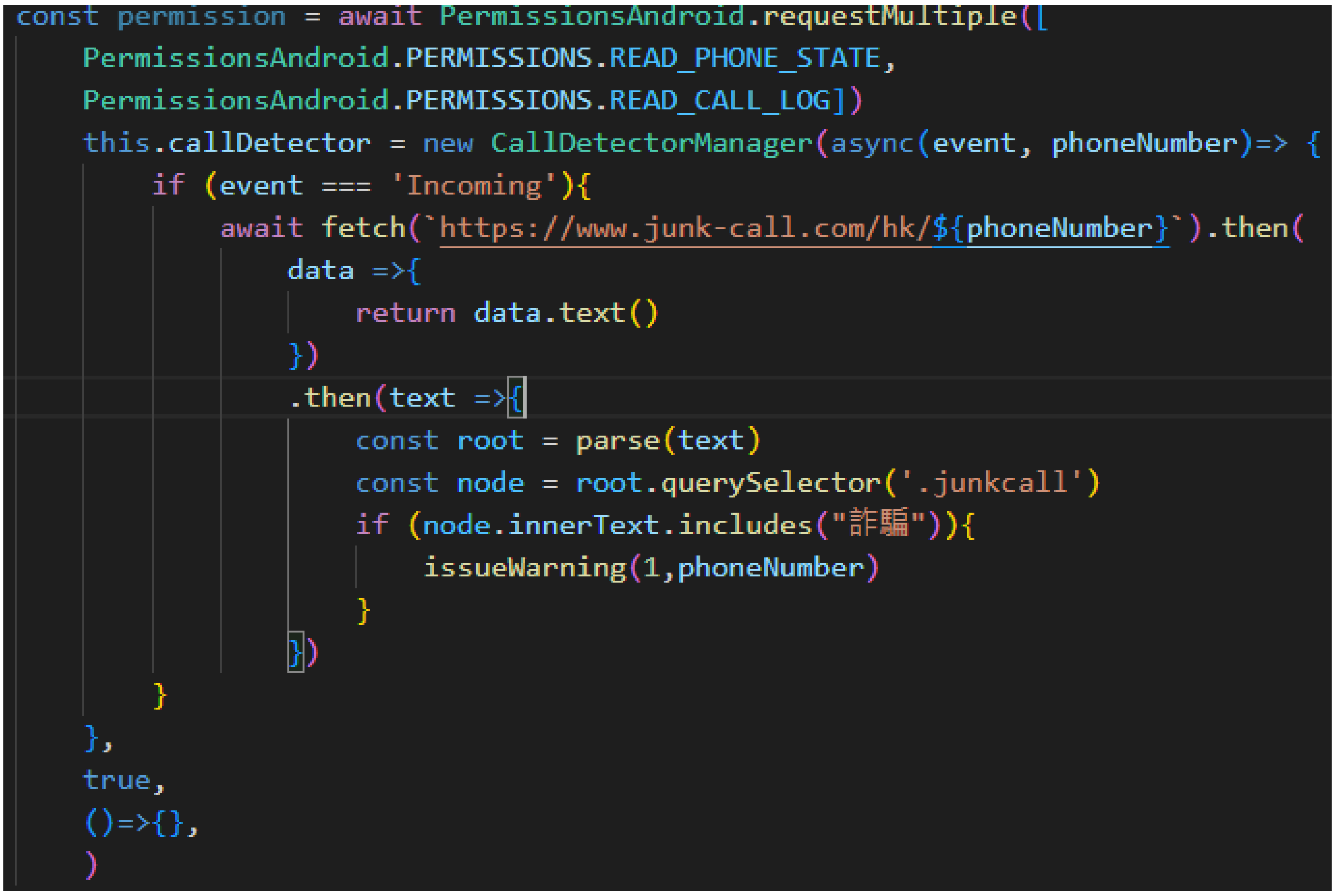
檢舉電話號碼「98320528」提醒更多人

98320528 詐騙
扮衛生局 (不明來電) 16:42
你有一份重要健康文件 聯絡多次未回覆

```
▶<div class="banner">…</div>
 <div class="content"></div>
▶<div class="content">…</div>
 <div id="junkcall"></div>
▶<div class="banner">…</div>
▼<div class="content">
  ▶<section style="text-align: center;">…</section>
   <h2>98320528來電記錄</h2>
  ▼<ol class="junkcall">
    ▶<li>…</li> == $0
    ▶<li>…</li>
   </ol>
  ▶<div class="center">…</div>
   <h2>更多98320528以外來電記錄</h2>
  ▶<ol class="junkcall">…</ol>
  ▶<div class="center">…</div>
  ▶<div class="center">…</div>
   <p></p>
  ▶<center>…</center>
   <p></p>
 </div>
▶<div class="banner">…</div>
▶<div class="content">…</div>
▶<div class="banner">…</div>
▶<div class="content">…</div>
 <br>
 <br>
▶<div class="center">…</div>
▶<div id="studygpt-root">…</div>
▶<table cellspacing="0" cellpadding="0" role="present
 dth: 509px; display: none; top: 1082px; position: a
</body>
```

Figure 35 Screenshot of JunkCall HK website and its html document [23]

To identify whether a phone number belongs to a deceptive call using the JunkCall HK database, the relevant information can be extracted from the HTML document. Based on the above screenshot, the targeted information, i.e., whether the number is reported as deceptive or not, is contained within an HTML ordered list element with the class name "junkcall".



```
const permission = await PermissionsAndroid.requestMultiple([
    PermissionsAndroid.PERMISSIONS.READ_PHONE_STATE,
    PermissionsAndroid.PERMISSIONS.READ_CALL_LOG])
    this.callDetector = new CallDetectorManager(async(event, phoneNumber)=> {
        if (event === 'Incoming'){
            await fetch(`https://www.junk-call.com/hk/${phoneNumber}`).then(
                data =>{
                    return data.text()
                })
                .then(text =>{
                    const root = parse(text)
                    const node = root.querySelector('.junkcall')
                    if (node.innerText.includes("詐騙")){
                        issueWarning(1,phoneNumber)
                    }
                })
        }
    },
    true,
    ()=>{},
    )
```

Figure 36 Implementation of scam calls detection

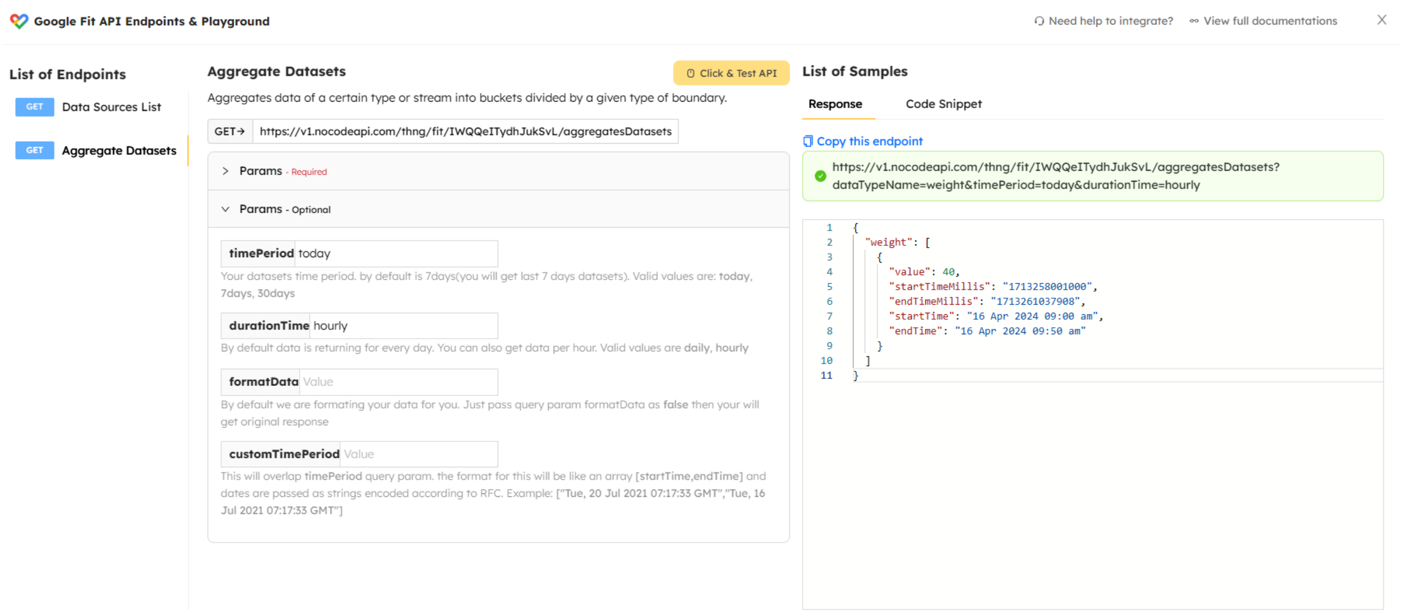
To access the incoming call number, permissions for `READ_PHONE_STATE` and `READ_CALL_LOG` must first be granted. Then, the `CallDetectionManager` from the 'react-native-call-detection' library is used to asynchronously detect incoming call numbers. [24]

Once the number is obtained, a GET request with the phone number appended to the URL is sent to the JunkCallHK website using the JavaScript fetch API. The response from the website is a text string.

The obtained text string is parsed into Document Object Model (DOM) elements, from which the specific element with the class name "junkcall" is extracted.

If the inner text of the extracted element contains words that indicate a scam call, a warning is issued.

3.6.6 Body Status Monitoring



Aggregate Datasets

Aggregates data of a certain type or stream into buckets divided by a given type of boundary.

GET→ https://v1.nocodeapi.com/thng/fit/IWQQeITydhJukSvL/aggregatesDatasets

Params - Required

Params - Optional

timePeriod: today

Your datasets time period. by default is 7days(you will get last 7 days datasets). Valid values are: **today, 7days, 30days**

durationTime: hourly

By default data is returning for every day. You can also get data per hour. Valid values are **daily, hourly**

formatData: Value

By default we are formating your data for you. Just pass query param formatData as **false** then your will get original response

customTimePeriod: Value

This will overlap **timePeriod** query param. the format for this will be like an array **[startTime,endTime]** and dates are passed as strings encoded according to RFC. Example: **["Tue, 20 Jul 2021 07:17:33 GMT","Tue, 16 Jul 2021 07:17:33 GMT"]**

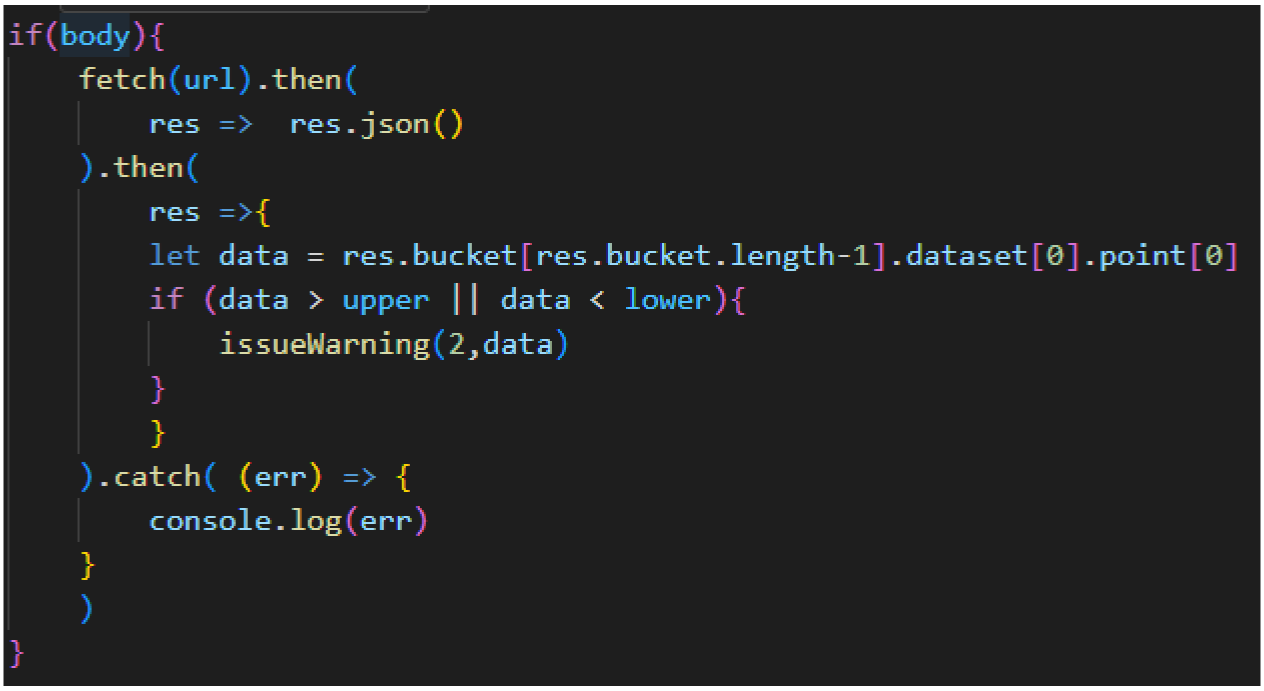
https://v1.nocodeapi.com/thng/fit/IWQQeITydhJukSvL/aggregatesDatasets?dataTypeName=weight&timePeriod=today&durationTime=hourly

```
{
  "weight": [
    {
      "value": 40,
      "startTimeMillis": "1713258001000",
      "endTimeMillis": "1713261037908",
      "startTime": "16 Apr 2024 09:00 am",
      "endTime": "16 Apr 2024 09:50 am"
    }
  ]
}
```

Figure 37 Screenshot of NoCodeAPI service for Google Fit API [25]

There are two ways to read health data from a user's smartwatch. The first method is to develop a customized smartwatch application that can pair with the application and synchronize the health data. However, this approach has a drawback in that different smartwatches use different operating systems, and a respective smartwatch application has to be developed for each operating system to support the specific smartwatch. This can result in a lot of redundant development work, which is not preferred.

The second solution is to rely on third-party applications that support different smartwatches and retrieve data from the third-party application. In this case, Google Fit is the chosen application since it is a popular health application in the market, and Google provides an API to retrieve the health data, which is convenient for developers. However, attempts to link the application to Google Fit using the suggested “react-native-google-fit” library have failed due to a version conflict with other libraries used. An alternative solution is to use an API service provided by a company called NoCodeAPI. This service allows for the retrieval of Google Fit data directly from a GET request without any setup or library requirements. [26]



```
if(body){
    fetch(url).then(
        res =>  res.json()
    ).then(
        res =>{
        let data = res.bucket[res.bucket.length-1].dataset[0].point[0]
        if (data > upper || data < lower){
            issueWarning(2,data)
        }
        }
    ).catch( (err) => {
        console.log(err)
    }
    )
}
```

Figure 38 Implementation of Body status monitoring

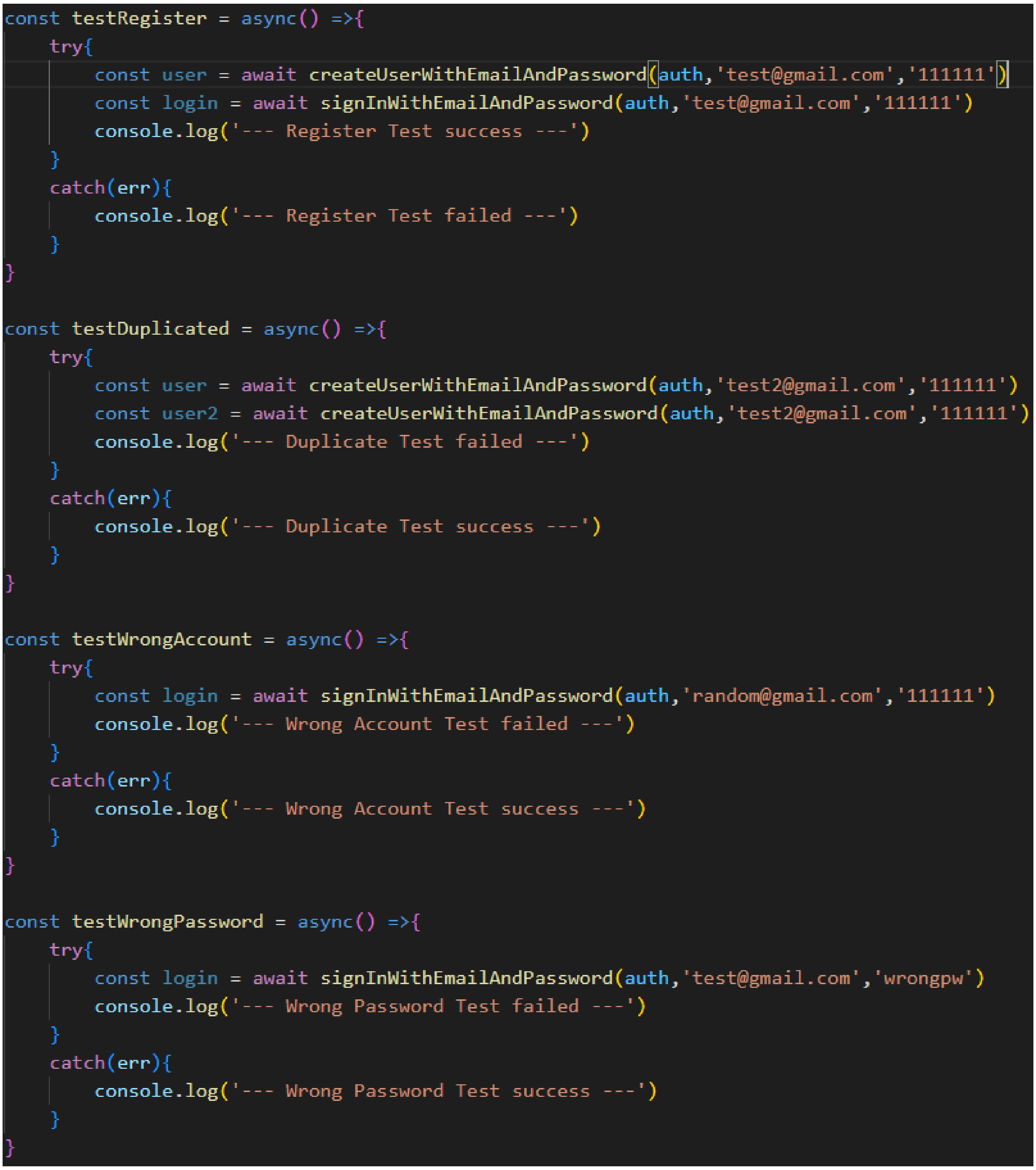
To track a user's body status using NoCodeAPI, users will need to sign up for the Google Fit API service within NoCodeAPI and provide their personal API during registration. The API will then be fetched constantly within the application using the fetch API.

If the retrieved data falls outside of the predefined limit, a warning will be issued to the user.

3.7 Testing

This section will focus on the set of tests used to validate the application. Section 3.7.1 – 3.7.4 will focus on the testing of registration and authentication, settings, fall detection, and deceptive call detection respectively. Section 3.7.5 will describe the user acceptance test performed.

3.7.1 Registration and authentication

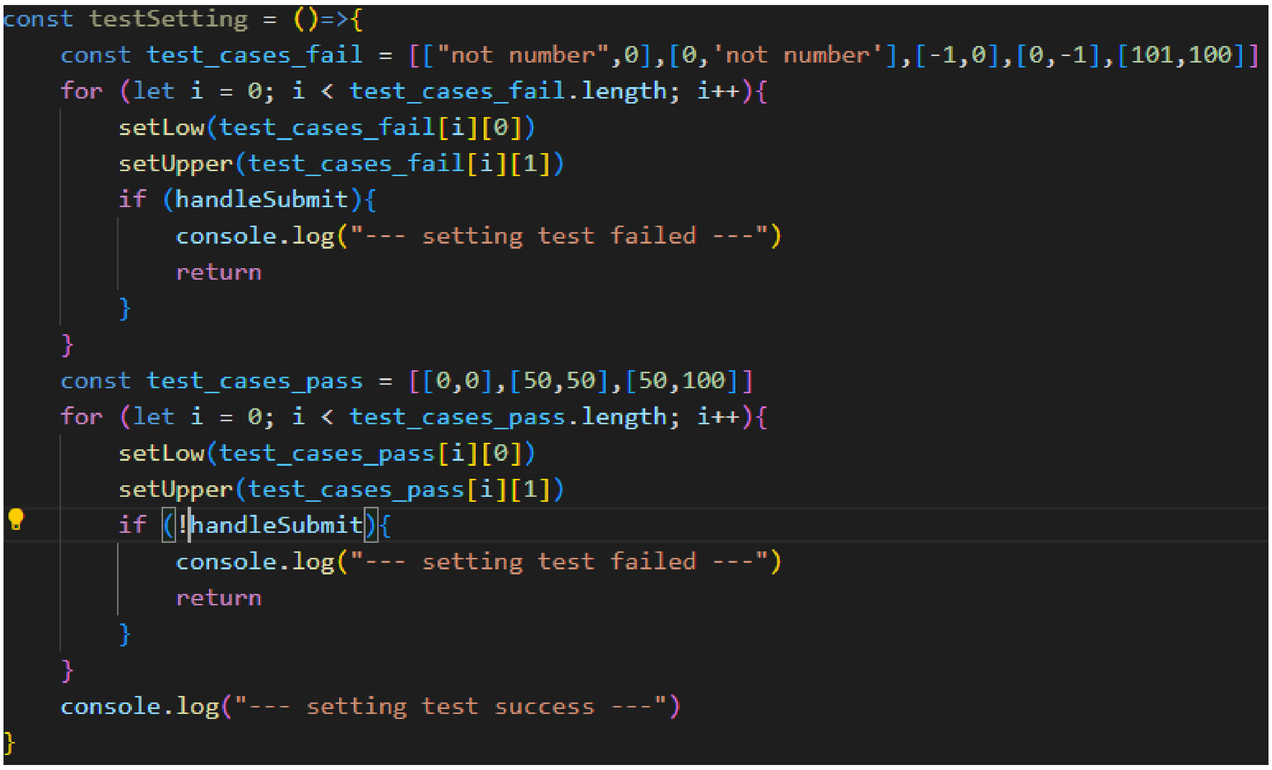


```
const testRegister = async() =>{
    try{
        const user = await createUserWithEmailAndPassword(auth,'test@gmail.com','111111')
        const login = await signInWithEmailAndPassword(auth,'test@gmail.com','111111')
        console.log('--- Register Test success ---')
    }
    catch(err){
        console.log('--- Register Test failed ---')
    }
}

const testDuplicated = async() =>{
    try{
        const user = await createUserWithEmailAndPassword(auth,'test2@gmail.com','111111')
        const user2 = await createUserWithEmailAndPassword(auth,'test2@gmail.com','111111')
        console.log('--- Duplicate Test failed ---')
    }
    catch(err){
        console.log('--- Duplicate Test success ---')
    }
}

const testWrongAccount = async() =>{
    try{
        const login = await signInWithEmailAndPassword(auth,'random@gmail.com','111111')
        console.log('--- Wrong Account Test failed ---')
    }
    catch(err){
        console.log('--- Wrong Account Test success ---')
    }
}

const testWrongPassword = async() =>{
    try{
        const login = await signInWithEmailAndPassword(auth,'test@gmail.com','wrongpw')
        console.log('--- Wrong Password Test failed ---')
    }
    catch(err){
        console.log('--- Wrong Password Test success ---')
    }
}
```

Figure 39 Testing of registration and authentication

As the application consists of users' sensitive and personal information, it is crucial to ensure that the authentication is functional and secure. There are a total of four tests on registration and authentication. The first test aims to validate that normal registration and authentication will not cause an error. The second test validates that it is impossible to create another account with the same email address. The third and fourth tests validate that using a wrong email address or password cannot bypass the authentication.

3.7.2 Settings



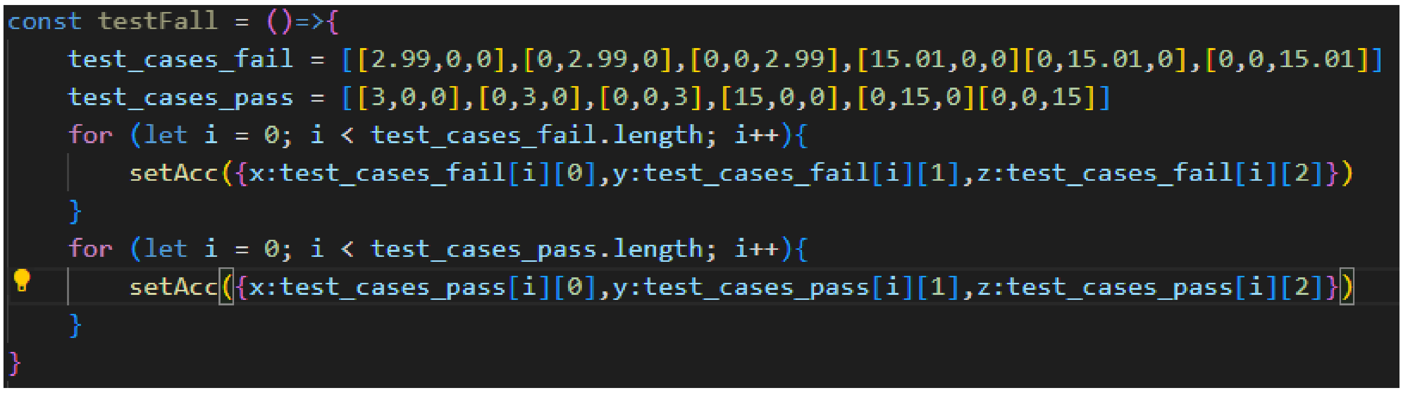
```
const testSetting = ()=>{
    const test_cases_fail = [["not number",0],[0,'not number'],[-1,0],[0,-1],[101,100]]
    for (let i = 0; i < test_cases_fail.length; i++){
        setLow(test_cases_fail[i][0])
        setUpper(test_cases_fail[i][1])
        if (handleSubmit){
            console.log("--- setting test failed ---")
            return
        }
    }
    const test_cases_pass = [[0,0],[50,50],[50,100]]
    for (let i = 0; i < test_cases_pass.length; i++){
        setLow(test_cases_pass[i][0])
        setUpper(test_cases_pass[i][1])
        if (!handleSubmit){
            console.log("--- setting test failed ---")
            return
        }
    }
    console.log("--- setting test success ---")
}
```

Figure 40 Testing for settings

It is crucial to ensure that the users' entered threshold of body status is valid because otherwise, it may trigger a bug and render the feature non-functional. The "test_cases_fail" test cases validate this by testing non-numeric inputs, negative inputs, and lower bound values larger than upper bound values. These inputs are expected to fail and help ensure that the application correctly handles invalid inputs.

The "test_cases_pass" input test cases validate the feature's proper functionality by testing valid input values. These include zero as input, equal lower bound and upper bound as input, and normal input values that are expected to pass. By conducting these tests, the application can ensure that the users' entered threshold values are properly validated, preventing bugs and maintaining the functionality of the feature.

3.7.3 Fall Detection



```
const testFall = ()=>{
    test_cases_fail = [[2.99,0,0],[0,2.99,0],[0,0,2.99],[15.01,0,0][0,15.01,0],[0,0,15.01]]
    test_cases_pass = [[3,0,0],[0,3,0],[0,0,3],[15,0,0],[0,15,0][0,0,15]]
    for (let i = 0; i < test_cases_fail.length; i++){
        setAcc({x:test_cases_fail[i][0],y:test_cases_fail[i][1],z:test_cases_fail[i][2]})
    }
    for (let i = 0; i < test_cases_pass.length; i++){
        setAcc({x:test_cases_pass[i][0],y:test_cases_pass[i][1],z:test_cases_pass[i][2]})
    }
}
```

Figure 41 Testing of fall detection algorithm

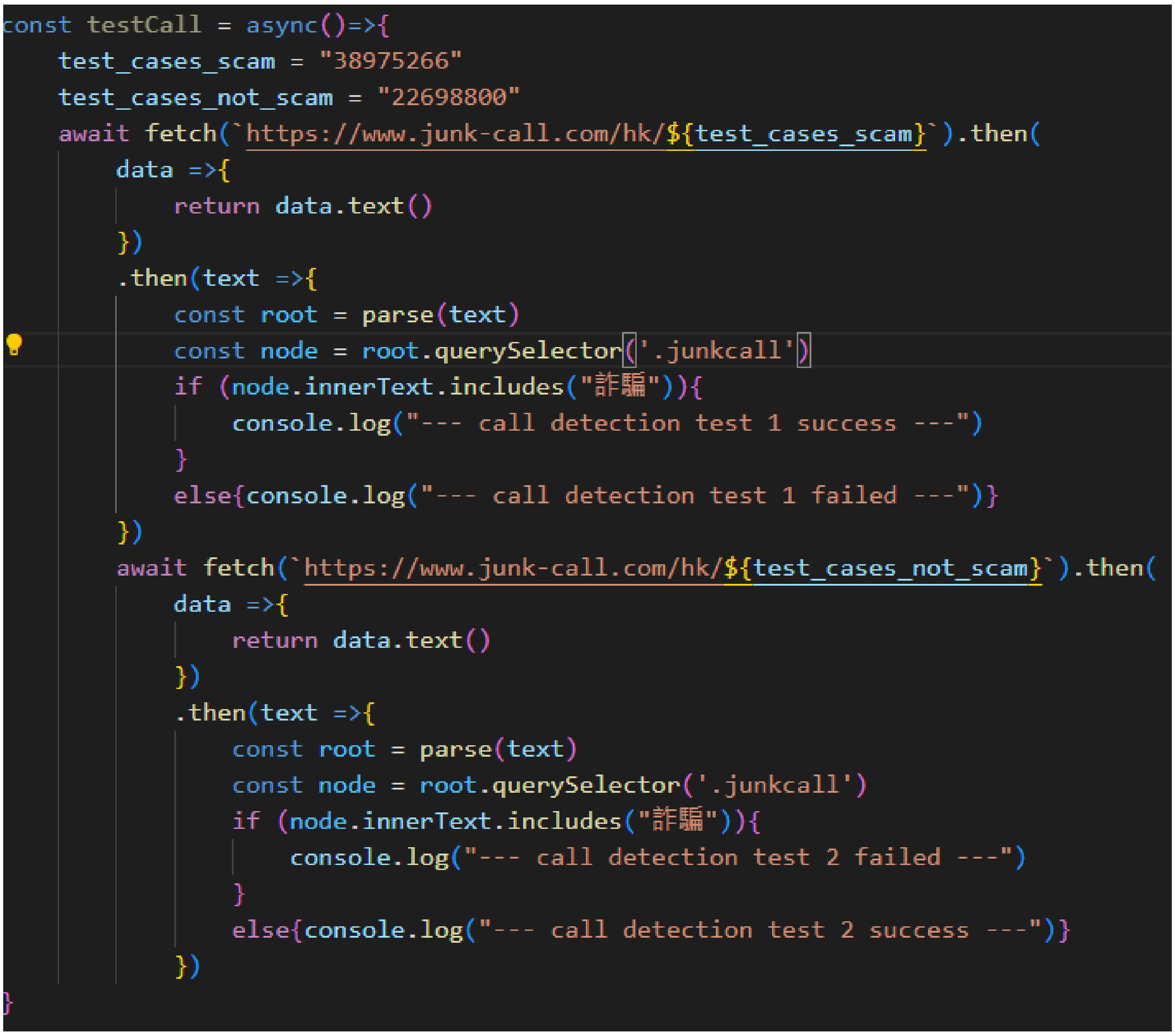
Boundary value analysis is an important testing technique used to ensure that components of acceleration function normally within a predefined range. In this case, the range is set between 3 to 15, and two test cases are performed to validate this range.

The first test case tests values of 2.99 and 15.01 in each component of acceleration. These values are expected to fall out of the predefined range, and no warning should be triggered. This test case validates that the application correctly handles values outside of the range and does not produce false alarms.

The second test case tests values of 3 and 15 in each component of acceleration. These values are within the predefined range and are expected to trigger a warning. This test case validates that the application correctly identifies values within the range and produces the appropriate warning.

By performing these tests, the application can ensure that each component of acceleration functions normally within the predefined range and produces the correct warning when values fall within or outside of the range.

3.7.4 Deceptive Call Detection



```
const testCall = async()=>{
    test_cases_scam = "38975266"
    test_cases_not_scam = "22698800"
    await fetch(`https://www.junk-call.com/hk/${test_cases_scam}`).then(
        data =>{
            return data.text()
        })
        .then(text =>{
            const root = parse(text)
            const node = root.querySelector('.junkcall')
            if (node.innerText.includes("詐騙")){
                console.log("--- call detection test 1 success ---")
            }
            else{console.log("--- call detection test 1 failed ---")}
        })
        await fetch(`https://www.junk-call.com/hk/${test_cases_not_scam}`).then(
            data =>{
                return data.text()
            })
            .then(text =>{
                const root = parse(text)
                const node = root.querySelector('.junkcall')
                if (node.innerText.includes("詐騙")){
                    console.log("--- call detection test 2 failed ---")
                }
                else{console.log("--- call detection test 2 success ---")}
            })
}
```

Figure 42 Testing of call detection

To validate the accuracy of the call detection algorithm, two phone numbers are manually obtained from the JunkCall HK website. One of these numbers is identified as a deceptive phone number, while the other is identified as a safe phone number. The algorithm is then applied to both numbers to determine if it correctly identifies them.

The first number, which is known to be deceptive, is expected to trigger a warning from the algorithm. This validates that the algorithm can accurately detect deceptive phone numbers and warn users about potential scams or fraudulent calls.

On the other hand, the second number, known to be safe, should not trigger any warning from the algorithm. This further validates the algorithm's accuracy by ensuring that it correctly identifies safe phone numbers and does not produce false alarms.

By conducting these tests, the application can verify the correctness of the call detection algorithm and ensure that it effectively distinguishes between deceptive and safe phone numbers.

3.7.5 User Acceptance Test

For features that are difficult to test with test cases, manual testing was conducted, and the results are as follows:

1. Location tracking: A random position was simulated using an emulator, and another linked device was used to observe whether the location is updated. The result was successful, and the location was correctly reflected. This validates that the location tracking feature is functioning correctly.

2. Calls filtering: Incoming calls were simulated on the emulator using both identified deceptive and safe phone numbers. The application successfully issued a warning for the deceptive phone numbers and did not issue a warning for the safe phone number. This confirms that the calls filtering feature is working as expected.

3. LLM integration: A simulated chat was conducted, asking common daily-life questions. The application provided a smooth chat experience and appropriately answered the questions. This demonstrates that the LLM integration feature is functioning correctly.

4. Fall detection: Two types of tests were performed for fall detection. Free falling of the phone on a soft surface consistently triggered the warning due to the bounce back. Falling simulations while holding the phone resulted in the warning being triggered in about 7 out of 10 tests. This indicates that the fall detection feature is working, albeit with some variability.

5. Body status monitoring: A smartwatch emulator was used to generate abnormal body status data. However, it was observed that the Google Fit application was not correctly uploading the health data to the cloud and synchronizing among devices. This resulted in fetching empty data from the API, rendering the body status monitoring features unable to be successfully implemented.

Based on these manual tests, it can be concluded that most of the features are functioning correctly. However, there is an issue with the body status monitoring feature due to the limitations of the Google Fit application.

3.8 Limitations, possible improvements, and future directions

This section will focus on listing the limitations of the project and suggesting possible actions to improve it. In sections 3.7.1 to 3.7.4, the discussion will focus on limitations and potential improvements of current features: LLM integration, deceptive calls filtering, fall detection, and body status monitoring, respectively. In section 3.7.5, future directions and recommendations will be discussed for further development or research.

3.8.1 LLM integration

As the Gemini API requires an IP address from an authorized region, users from restricted regions may face difficulty accessing the service. To overcome this barrier, using a VPN can help bypass the regional restrictions.

Alternatively, another solution is to utilize other LLM API services that do not impose any region restrictions. This would ensure that users from all regions can access the service without the need for a VPN.

It is also possible to host a server with an LLM loaded, but it is essential to ensure that the server has sufficient GPU resources. Adequate GPU resources will guarantee reasonable inference time and performance, providing a smooth user experience.

3.8.2 Deceptive Calls Filtering

The current deceptive calls filtering method heavily relies on the JunkCall HK database, which only records Hong Kong phone numbers. As a result, the list of deceptive calls may not be comprehensive, and users may still receive unwanted calls from other regions.

To improve the filtering system, one possible solution is to find other services or databases that provide information on deceptive calls in other regions. This would help expand the database and provide more comprehensive coverage, allowing users to avoid unwanted calls from a wider range of regions.

Another possible improvement is to train a speech recognition model to identify suspicious keywords, such as "money" or "bank," during phone conversations. This approach would enable the system to detect potential deception and flag the call as suspicious. By combining both methods, the filtering system could become more accurate and reliable, providing users with better protection against unwanted calls.

3.8.3 Fall Detection

The current fall detection algorithm implemented in the project is relatively simple, and there is potential for further improvement. However, due to the limitations of React Native as a cross-platform development tool, the algorithm can only provide a rough estimate of fall detection since it cannot fully utilize all the available sensors in smartphones.

To improve the fall detection algorithm, native development using Android Studio or Xcode is preferred as it can fully utilize the built-in sensors in smartphones. By combining different sensor readings, a more accurate and reliable fall detection algorithm can be developed. Additionally, building a custom fall detection algorithm using machine learning techniques may further improve the performance.

To develop a machine learning-based fall detection algorithm, sensor data can be retrieved from falling and non-falling scenarios performed by elderly volunteers or obtained from hospitals. This data can be used to train a machine learning model to find the optimal set of parameters for different sensor readings. A more complex, multi-step classification model can also be proposed to improve accuracy. By continuously collecting more data and refining the algorithm, the fall detection feature can become more robust and reliable, providing valuable assistance to elderly users.

3.8.4 Body Status Monitoring

As mentioned previously, relying on the third-party Google Fit app to retrieve body status data may not provide up-to-date information. To address this issue, other third-party applications such as Apple Health can be considered. Additionally, it is possible to develop a customized smartwatch application that pairs with the chosen app and constantly synchronizes the health data. This approach would ensure that the body status data is accurate and up-to-date, providing users with a reliable monitoring system.

Another potential improvement to the body status feature is to customize the thresholds based on each user's body status statistics. The current threshold is set by users themselves, which may not be appropriate for everyone. By using the user's recent health data, a set of thresholds can be generated based on a certain algorithm, such as a fixed margin between the mean. This approach would provide a more personalized monitoring system, ensuring that each user receives appropriate alerts based on their unique body status.

3.8.5 Future Direction

This project has the potential to serve as a prototype for a truly general-purpose, all-in-one application that includes all necessary functions for elderly users. In addition to the existing functionalities, as discussed above, new features that are useful to elderly users can be added. For example, the application could cooperate with the government or NGOs to assist in providing welfare, organize events, or announce important news.

By aggregating these new features into the application, it would become a more comprehensive and valuable tool for elderly users. This would help improve their quality of life, provide them with access to important resources and information, and help them stay connected with their community. As the elderly population continues to grow, the need for applications like this will only increase, making this project an important step towards providing elderly users with the tools they need to live healthy, happy lives.

4 Conclusion

In today's society, elderly individuals often face challenges such as telephone deception, missing persons, accidents, and a generation gap. To address these issues, this paper proposes the development of a customized mobile application that harnesses the power of technology.

The application will encompass several key functions that cater specifically to the needs of the elderly. These functions include location tracking, body status monitoring, deceptive calls detection, fall detection, and integration with LLM technology. Preliminary designs, including user interface, system, and database design, are created to guide the implementation and testing processes.

While there is room for improvement, such as utilizing machine learning to enhance the fall detection algorithm, the application is essentially capable of supporting most of the targeted features. It serves as a promising prototype for a general-purpose, elderly-focused application.

Moving forward, future steps involve further development of the application, focusing on improving the performance of existing features. Additionally, exploring new features like collaborating with different parties to offer more services to the elderly can be considered. The ultimate goal is for this application to comprehensively meet the needs of the elderly, providing them with the necessary support, safety, and assistance in their daily lives.

5 Reference

[1] Anti-Deception Coordination Centre, *Scam Statistics*

https://www.adcc.gov.hk/en-hk/statistic.html (accessed: Oct 1, 2023)

[2] Security Bureau, Hong Kong Police Force, *Legislative Council Panel on Security Initiatives for Preventing and Combatting Deception Cases,* May 2021

https://www.legco.gov.hk/yr20-21/english/panels/se/papers/se20210601cb2-1110-3-e.pdf (accessed: Oct 1, 2023)

[3] Sarah Perez, *Google's Family Link updates reflect the pandemic's impact on how parents view screen time,* 16 March, 2021

https://techcrunch.com/2021/03/16/googles-family-link-updates-reflect-the-pandemics-impact-on-how-parents-view-screen-time/ (accessed: Nov 24, 2023)

[4] Whoscall official website

https://whoscall.com/en/about (accessed: Nov 24, 2023)

[5] Google Family Link official website

https://families.google/intl/en/familylink/ (accessed: Nov 24, 2023)

[6] Apple Official Website

https://www.apple.com/hk/en/ios/health/ (accessed: Nov 24, 2023)

[7] React Native official website

https://reactnative.dev/ (accessed: Oct 1, 2023)

[8] Visual Studio Code official website

https://code.visualstudio.com/ (accessed: Oct 1, 2023)

[9] Firebase official website

https://firebase.google.com/ (accessed: Oct 1, 2023)

[10] Lucidchart official website

https://www.lucidchart.com/ (accessed: Oct 1, 2023)

[11] Lucidchart , *Introducing Types of UML Diagrams | Lucidchart Blog*
https://www.lucidchart.com/blog/types-of-UML-diagrams (accessed: Oct 1, 2023)

[12] Lucidchart , *UML Class Diagram Tutorial*

https://www.lucidchart.com/pages/uml-class-diagram (accessed: Oct 1, 2023)

[13] Figma official website

https://www.figma.com/ (accessed: Oct 1, 2023)

[14] Kinsta, *What Is GitHub? Kinsta A Beginner's Introduction to GitHub*, Dec 13, 2022

https://kinsta.com/knowledgebase/what-is-github/ (accessed: Oct 1, 2023)

[15] GeeksforGeeks, *Unit Testing | Software Testing,* 06 Feb, 2023

https://www.geeksforgeeks.org/unit-testing-software-testing/ (accessed: Oct 1, 2023)

[16] GeeksforGeeks, *Software Engineering | Integration Testing,* 06 Apr, 2023

https://www.geeksforgeeks.org/software-engineering-integration-testing/ (accessed: Oct 1, 2023)

[17] GeeksforGeeks, *Software Testing – UAT (User Acceptance Testing),* 29 Mar, 2023

https://www.geeksforgeeks.org/software-testing-uat-user-acceptance-testing/ (accessed: Oct 1, 2023)

[18] React Native Firebase website

https://rnfirebase.io/ (accessed: Apr 1, 2024)

[19] Google, Get started with the Gemini API in Node.js applications

https://ai.google.dev/gemini-api/docs/get-started/node (accessed: Apr 1, 2024)

[20] NPM, expo-location library

https://www.npmjs.com/package/expo-location (accessed: Apr 1, 2024)

[21] Noemi Biancone, Chiara Bicchielli*, Fernando Ferri and Patrizia Grifoni, *Multi-Sensor Fall Detection for Smartphones,* Published: March 23, 2021

[22] Expo documentation, Expo Sensors

https://docs.expo.dev/versions/latest/sdk/sensors/ (accessed: Apr 1, 2024)

[23] JunkCall HK website

https://www.junk-call.com/hk (accessed: Apr 1, 2024)

[24] NPM, React Native Call Detection

https://www.npmjs.com/package/react-native-call-detection (accessed: Apr 1, 2024)

[25] NoCodeAPI website

https://nocodeapi.com/ (accessed: Apr 1, 2024)

[26] Google Fit, Rest API

https://developers.google.com/fit/rest (accessed: Apr 1, 2024)