Chatbot App Documentation

0. Introduction

ChatBot App is a user-friendly application that allows users to send and receive messages to the OpenAI API. The app includes functionalities for registration and login. Ensuring that users can store their chat history and either continue previous conversations or start new ones.

0.1 Purpose

The primary purpose of the ChatBot App is to allow users to communicate with an AI assistant. The app facilitates real-time communication between users and the OpenAI API. Utilizing Firebase for authentication and database management, the app ensures secure and reliable access to user accounts and chat histories.

The user interface is designed with intuitiveness in mind and allows users to engage in meaningful and continuous interactions either by resuming existing conversations or starting new ones.

0.2 Scope

The scope of the ChatBot App is development, deployment, and maintenance of an Android-based application designed to enable real-time communication with the OpenAI API.

The scope of the project does not include the development of versions for other platforms such as IOS or a web-based version. The focus remains on creating a reliable and efficient Android application for secure and real-time communication with the OpenAI API.

Application functions include; registration/login, send/receive messages to the AI, chat history management, password and account information management, intuitive user interface, compatibility with different android devices.

0.3 Audience

The primary audience for the ChatBot App includes; general users, students and educators.

0.4 Assumptions and Dependencies

Assumptions:

1. Internet Access: The application assumes that users will have a stable internet connection.

2. User Credentials: The application assumes that users will provide credentials such as a valid email and nickname at registration and login.

3. API Availability: The application assumes continuous availability of Firebase and the OpenAI API services for seamless operation.

4. User Knowledge: It is assumed that users have basic knowledge of operating Android applications and can follow on-screen prompts and instructions.

5. Device Compatibility: The application assumes users will have Android devices compatible with the minimum operating system requirements.

Dependencies:

1. Firebase Authentication Service: Is used for secure user login and registration.

2. Firebase Realtime Database Service: Is used for storing and retrieving chat messages and user data.

3. OpenAI API: The application depends on the OpenAI API for processing and responding to user messages.

4. Third-Party Libraries: Networking between the user and OpenAI API depends on retrofit library (HTTP client built on OkHttp).

5. Android SDK: The application depends on the Android SDK for development, ensuring compatibility with various Android devices.

1. Requirement Analysis

1.1 Functional Requirements

1.User Authentication

• The user should be able to;

• register a new account with their valid e mail address.

• login securely using their credentials.

• logout securely at any time.

• choose their nickname at registration.

• change their passwords and nicknames.

• view their account information.

2. Chat Functionality

• The user should be able to;

• send messages and receive responses from the AI assistant.

• view their chat histories.

• start a new chat.

• delete chats from their chat history.

• continue on their previous chats.

• Each chat should have a topic to be viewed in history.

• Topics should be the first ever message sent by the user on the corresponding chat instance.

• Topics should have a maximum length of 30 characters.

• The chat should be saved every time the user returns back from the chat.

3. User Interface and Views

• The user should be greeted by login screen.

• The login screen should have a way to navigate to register screen.

• The user should be able to decide their nickname at registration.

• User should be greeted with their nickname at main menu after logging in.

• User should be able to navigate to chat history or account screen from the main menu.

• User should have an option to log out at the menu.

• The account screen should show the users e-mail address.

• The account screen should have buttons to change password, change nickname or go back to main menu.

• The change password screen should have a field to input the current password and two fields for the new password.

• The change password screen should have a button to go back to main menu.

• The change nickname screen should display current nickname and have a field to input new nickname also a button to update the nickname and a button to go back to main menu.

• The chat history screen should display previous chat topics in a grid view.

• The user should be able to continue on their previous chats upon single touch input to the corresponding topic.

• The chat history screen should have a button to start a new chat and a button to go back to menu.

• The chat screen should display previous chat if the user selects a chat from their history.

• The chat screen should have a button to delete the chat currently on display and a button to go back to chat history screen.

• The chat screen should have a text field where users can input and a button to send the message to the AI assistant.

• The chat screen should adjust itself if the text view is full and scroll down.

• The chat screen should display both user input messages and messages received from AI.

• The chat screen should display the user input message with the user's nickname.

• The chat screen should have an indicator that AI message is being processed.

1.2 Non-Functional Requirements

1. Performance

• The app should be and feel responsive to the user input with minimal delays.

• The app should authenticate users and load their data quickly.

• The messages should be sent and received from the API with minimal delay.

• An individual chat history should be only fully loaded from the database if the user demands it otherwise only the first user inputs from the chats should be loaded to determine topics for chat history grid therefore saving resources.

• The app should optimize CPU, memory, and network usage to ensure smooth operation on a wide range of devices.

2. User Interface

• The user interface should be intuitive and should not take much time to get accustomed to.

• The user interface should not require more than basic application knowledge to be navigated.

• The user interface should be mobile friendly and feel responsive to the user.

• The app should have an intuitive and consistent user interface that is easy to navigate for users of all ages and technical backgrounds.

• The app should provide clear and concise feedback for user actions, such as successful logins, message sent/received notifications, and error messages.

3. Security

• User credentials must be secured using Firebase Authentication.

• User-specific data should not be accessible to other users and securely handled.

4. Maintainability

• Code must follow clean architecture principles.

• The app should be designed using modular components to facilitate easy updates, maintenance, and addition of new features.

5. Reliability

• The app should ensure data consistency across all users and devices, with real-time synchronization of messages and user information.

• The app must be available almost all the time, with minimal downtime.

• The app must gracefully handle errors and exceptions, providing appropriate error messages and logging for debugging purposes.

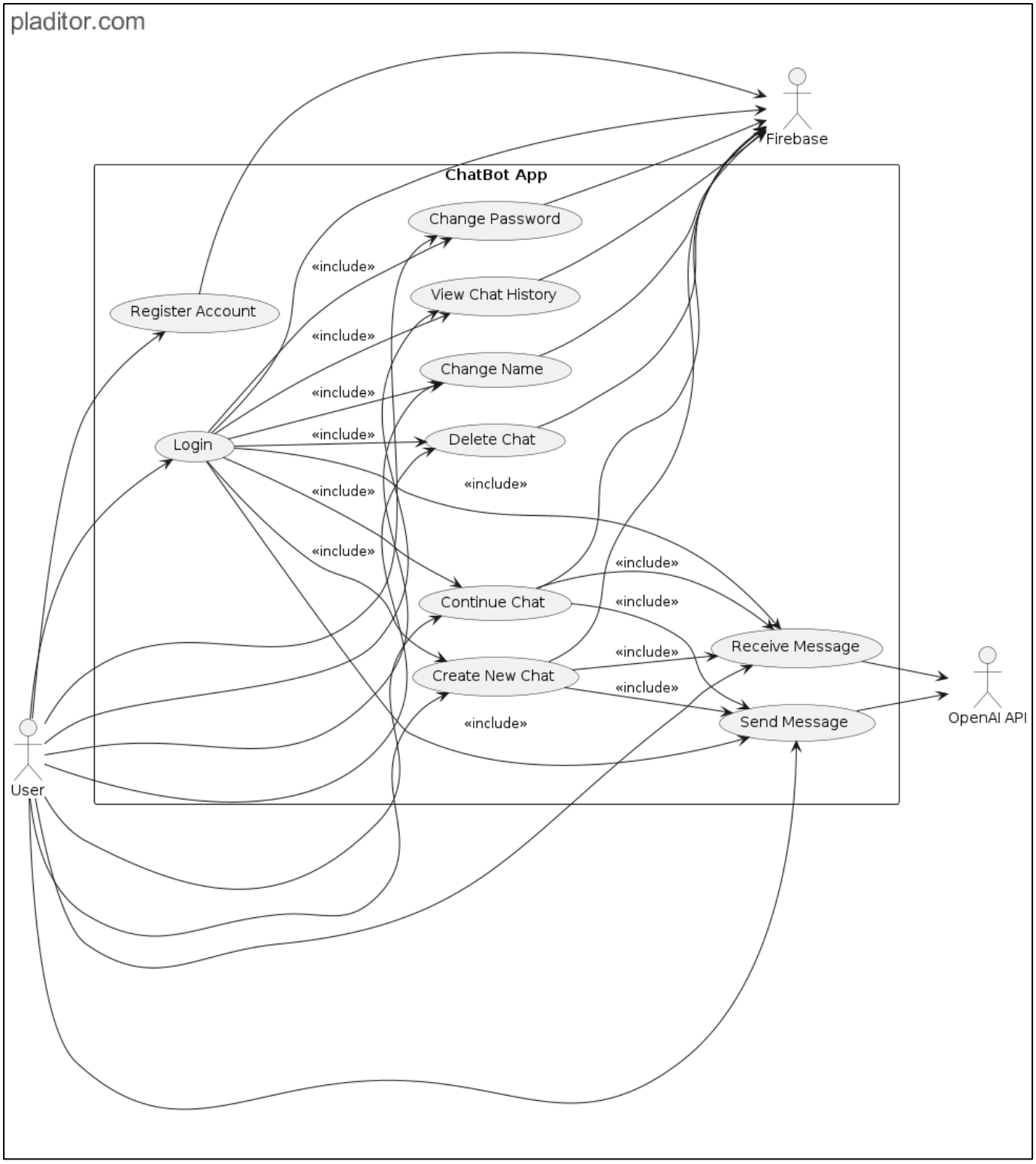
6. Compatibility

• The app must be compatible with a wide range of Android devices.

• The app must support different screen sizes and resolutions.

2. Use Case

2.1 Use Case Diagram



2.2 Use Cases

Use Case 1:

User Registration

Participating Actors:

Initiated by User

Communicates with Firebase Authentication

Flow of events:

1. User runs the app for the first time and greeted by login screen. User then press “Register Now”.

2. System opens up registration screen. User enters credentials email, password, and name and press “Register”.

3. Credentials along with a custom user class object is sent to Firebase.

4. Firebase creates a new user account under new user id.

5. User is granted authentication and redirected to main menu.

Use case 2:

User Login

Participating Actors:

Initiated by User

Communicates with Firebase Authentication

Flow of events:

1. User runs the app and authentication is timed out or user logged out and greeted by login screen.

2. User enters credentials email and password and press “Login”.

3. The app sends the credentials to Firebase Authentication.

4. Firebase validates the credentials. If credentials are valid, the user is logged in and user is redirected to the main menu.

Use case 3:

Create New Chat

Participating Actors:

Initiated by User

Communicates with OpenAI API

Communicates with Firebase Realtime Database

Flow of events:

1. User initiated a new chat trough the start new chat button.

2. A new chat id is created on the Firebase database and passed to the application.

3. User input new message and pressed send. Message is displayed on screen. Message is added in a custom message class with role user. Message object is then added to messages list and sent to the api.

4. Api responds back and contents are displayed on the screen and put into another message object. That object is also added to the messages list.

Use case 4:

Save Chat

Participating Actors:

Initiated by User

Communicates with OpenAI API

Communicates with Firebase Realtime Database

Flow of events:

1. User wants to save and quit the current chat and press the go back button.

2. Messages list is passed onto the Firebase Realtime Database under corresponding chat id.

3. User is redirected to the chat history screen and can view the previous chat in history.

Use case 5:

Delete Chat

Participating Actors:

Initiated by User

Communicates with Firebase Realtime Database

Flow of events:

1. User wants to delete the current chat and press delete button. User is asked to confirm this action by a pop up.

2. Chat id is already fetched and corresponding chat is deleted from the Firebase Realtime Database.

3. User is redirected to the chat history screen.

Use case 6:

Continue Chat

Participating Actors:

Initiated by User

Communicates with OpenAI API

Communicates with Firebase Realtime Database

Flow of events:

1. User wants to continue a previous chat and presses on a chat topic from the chat history.

2. Chat id is fetched from the chat info class object. Chat data is requested from the Firebase Realtime Database and loaded onto a new messages list consisting of message objects.

3. Messages list is passed to the text view and previous chat is displayed on the screen.

4. User enters new input and press the send button. Messages list is updated with the new user input and passed to the OpenAI API. API responds and response is displayed and added to the messages list.

Use case 7:

Change Password

Participating Actors:

Initiated by User

Communicates with Firebase Authentication

Flow of events:

1. User wants to change password and navigates to account from the main menu. From account screen user chooses the option change password.

2. User is redirected to password change screen and asked to enter current password and the new password latter two times.

3. User press change password button and data is sent to Firebase Authentication. Firebase responds with task complete and user is notified of the successful operation.

Use case 8:

Change Name

Participating Actors:

Initiated by User

Communicates with Firebase Realtime Database/Authentication

Flow of events:

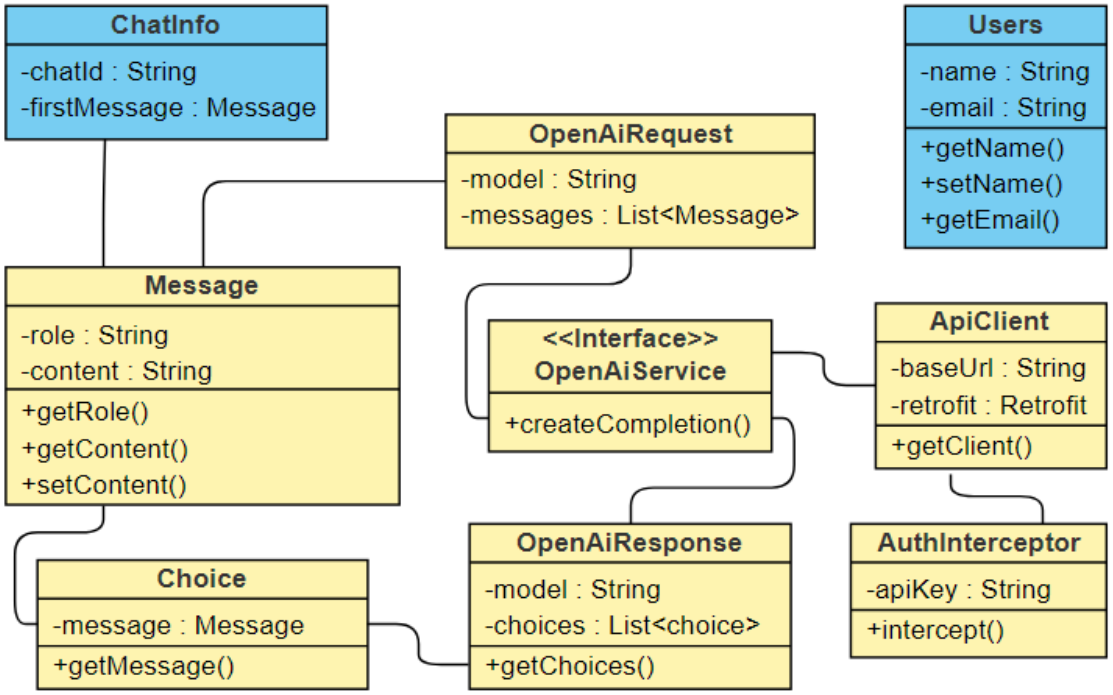
1. User wants to change nickname and navigates to account from the main menu. From account screen user chooses the option change name.

2. User is redirected to name change screen and presented with current name displayed and a text input field for new name. User input new name and press change name button.

3. A new reference to the firebase database is created and user's user object is fetched new name is inserted as the name and the new object is passed to the Firebase Realtime Database.

3. Class Diagram

3.1 Diagram



3.2 Relationships

Auth interceptor attaches the authentication header to all the packages sent to OpenAI API which includes the API key.

`ApiClient` class is designed to set up and return a `Retrofit` client instance configured with a base URL and the interceptor for authentication. For efficiency, this setup ensures to create the Retrofit instance once and reuse it for all API calls.

OpenAiService interface makes a post request to the OpenAI API through retrofit instance. Creates a completion which takes an OpenAiRequest object as body and returns the API response as OpenAiResponse object. Also handles the HTTP request and response asynchronously.

Message class is used by many classes throughout the application as a means to store chat entries from both the user and the AI assistant. A List of message objects basically define a chat. Choice class is to handle AI response style as the

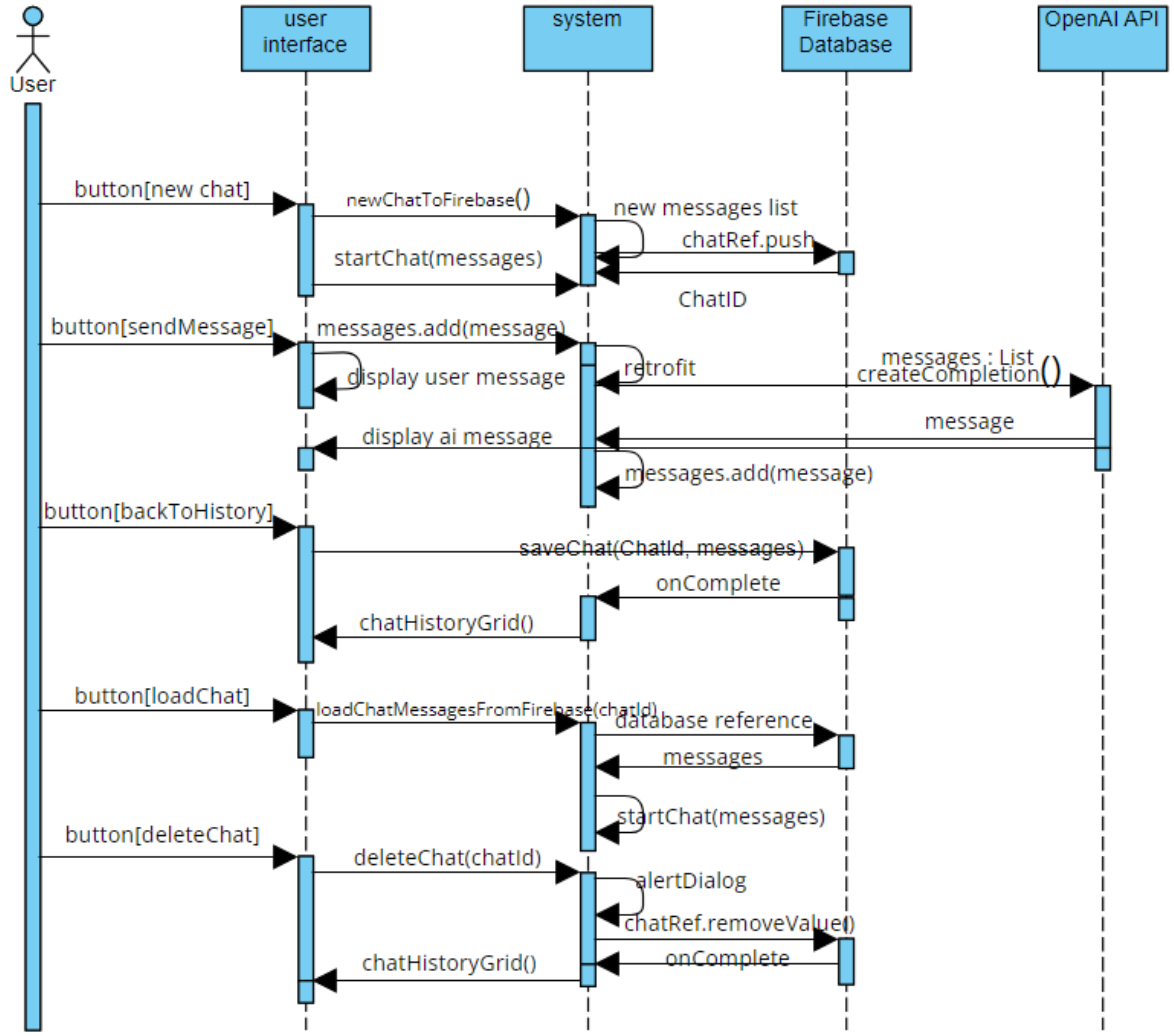
response for a single user input returns more than one choice system needs a way to categorize them.

Chat info class is used to determine which chat is currently in process and also determine topics to be shown on the chat history grid. For the grid there is a grid adapter class.

Users class holds the user information and the username is accessed by chat function every time user inputs a message to the chat.

4. Sequence Diagram

4.1 For Chat Management



Key process 1: New Chat

1. User presses new chat button. And system creates a new list of message objects.

2. A new database entry for the new chat is created and the chat id is retrieved from database.

3. System sets view to chat screen.

Key process 2: Send Message

1. User enters the desired message in the text field and press the send message button.

2. User message is added to the list of message objects with role user and displayed on the screen.

3. A new retrofit instance is initialized and a completion request is sent to the OpenAI API containing the list of message objects.

4.Upon receiving AI response, it is added to the list of message objects and displayed on the screen.

Key process 3: Back to History (save chat)

1. User presses go back button and system calls saveChat() function passes the chat id and the current list of message objects as parameter.

2. saveChat() function updates the Firebase Realtime Database at the given chat id with the given message list therefore chat history is saved.

3. System sets view to chat history.

Key process 4: Load Chat

1. User presses on the topic of desired previous chat. System fetches the chat id from the grid adapter and uses it to retrieve the desired chat history from Firebase Realtime Database.

2. Upon receiving complete chat history into a local message object list it is passed to the startChat() function.

3. System sets view to chat screen and displays previous chat entries.

Key process 5: Delete Chat

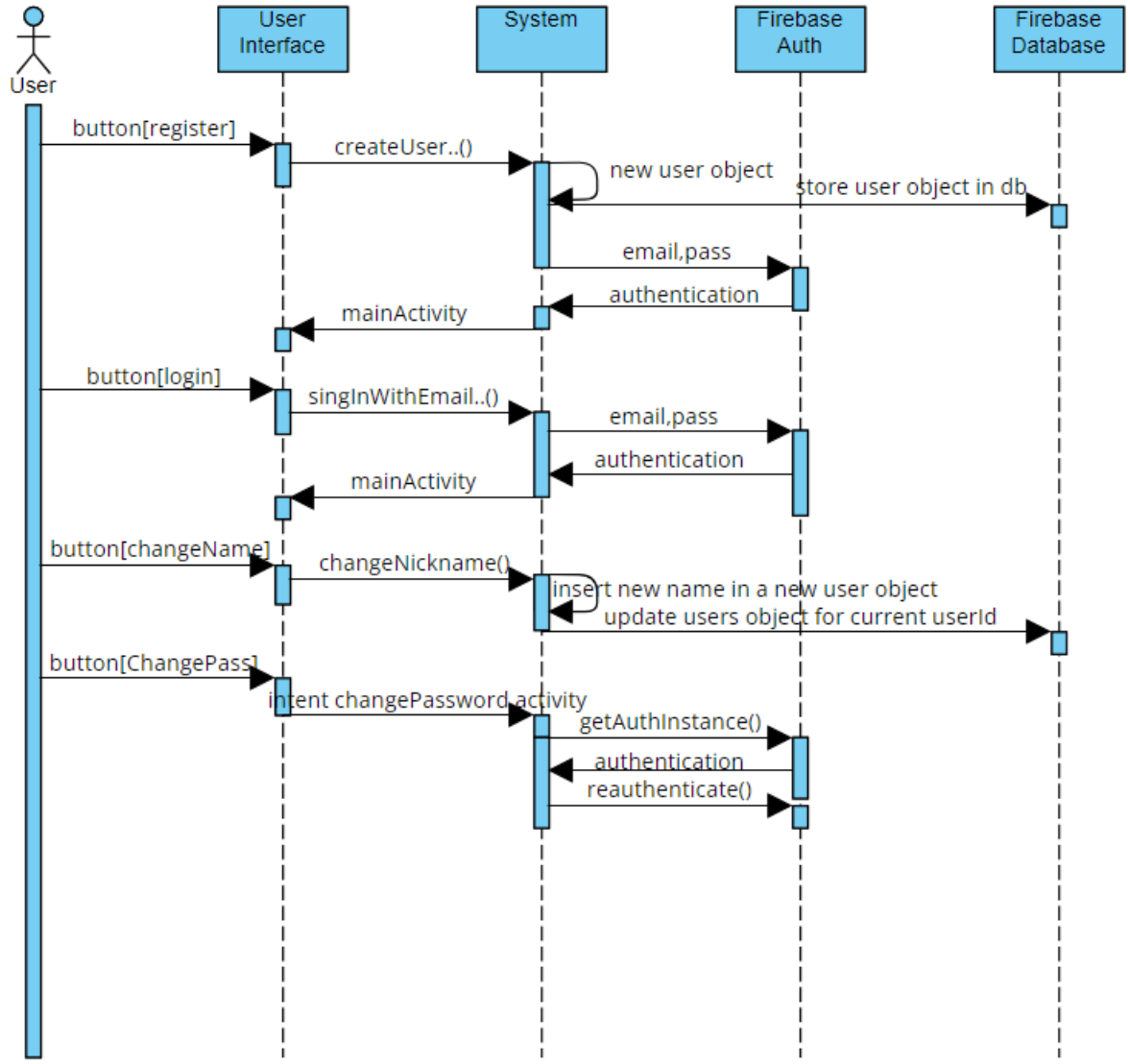
1. User presses delete chat button. And system calls deleteChat() function.

2. deleteChat() function prompts the user with a pop-up dialog if they are sure about the deletion of the chat.

3. With user confirmation node is deleted from the Firebase Realtime Database chat history.

4.System sets view to chat history grid view.

4.2 For Account Management



Key process 1: Register

1. User enters credentials and press the register button. Email and password is sent to the Firebase Authentication. If registration is a success the system creates a new user object with given credentials and stores the user in Firebase Realtime Database.

2. Authentication for newly created user is retrieved from Firebase Authentication.

3. System sets view to main menu.

Key process 2: Login

1. User enters credentials and press the login button. Email and password is sent to the Firebase Authentication.

2. If login is a success authentication for the user is retrieved from Firebase Authentication.

3. System sets view to main menu.

Key process 3: Change Name

1. User enters desired new nick name and press the change name button. A new user object is created by the system with the same credentials and updated username.

2. The newly created user object is then sent to the Firebase Realtime Database.

Key process 4: Change Password

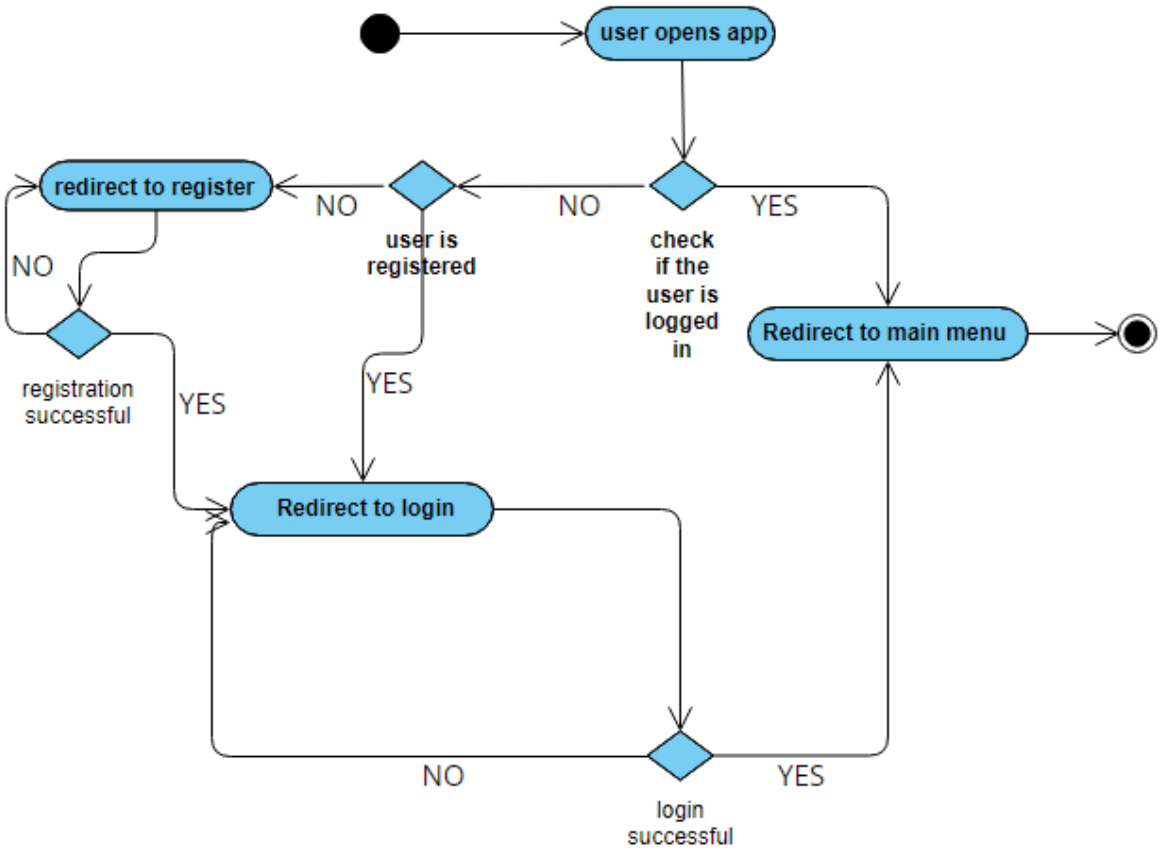
1. User enters their current password and the new password latter two times and press the change password button.

2. Authentication for current instance is fetched from the Firebase Authentication.

3. Using the built-in user.reauthenticate() function the password is updated and user is sent back to main menu.

5. Activity Diagrams

5.1 User Authentication Activity Diagram



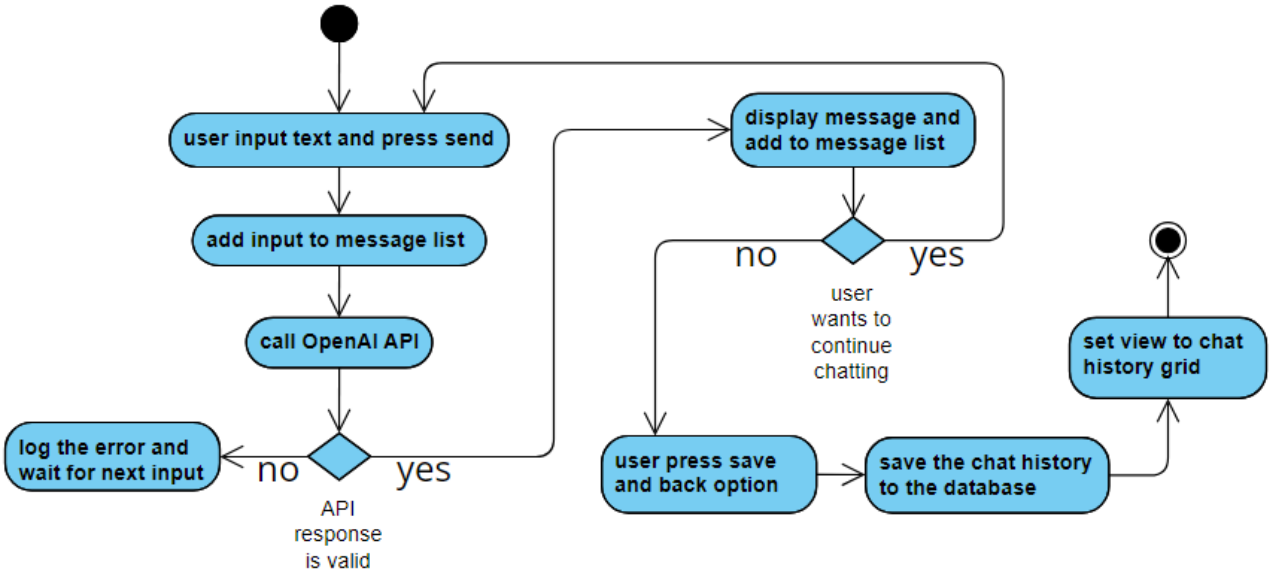
Authentication Workflow:

1. User opens the application and if the user authentication is still valid from the previous instance user is directly sent to the main menu.
2. If the user authentication is not valid user is greeted by the login screen. From this screen the user can choose to login using their credentials or register a new account.
3. If the user is registered and tries to log in with their credentials their credentials are sent to the Firebase Authentication. If the response is a valid authentication user is redirected to the main menu. If the credentials entered by the user is wrong user is prompted to try again.
4. If the user is not registered, the user can choose the option to register. User is then sent to the registration screen and asked to provide

credentials. Credentials are sent to the Firebase Authentication and depending on the response user is prompted.

5. If registration is a success user can then press the login now button and be redirected to the main menu without the need of entering their credentials again.

5.2 Send Messages to the AI Assistant Activity Diagram



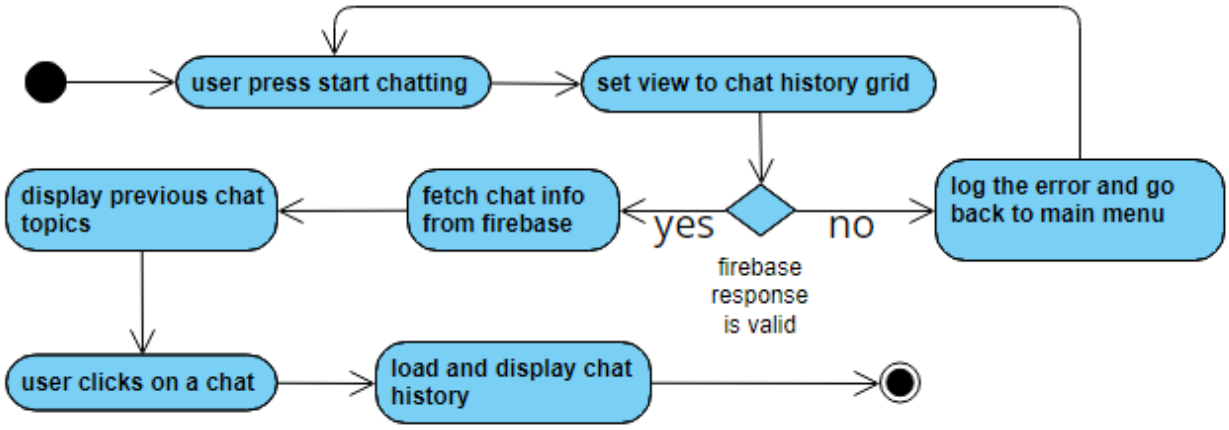
Communication With AI and Save Chat History Workflow:

1. User inputs their desired message and press the send message button. System prints the message on the screen and takes the message into a custom container and sends it to the OpenAI API. An indication progress bar is visible until receiving any response from the API.
2. Upon receiving an un-valid response from the API system logs the error and waits for the next user input.
3. Upon receiving a valid response from the API system displays the message and adds the received message into the message container (chat history).
4. User has the option to continue chatting.
5. When the user is done with the current chat user can choose to save go back to the chat history grid via go back button. Upon pressing go back

messages container is sent to the Firebase Realtime Database and saved.

6. User is redirected to chat history grid.

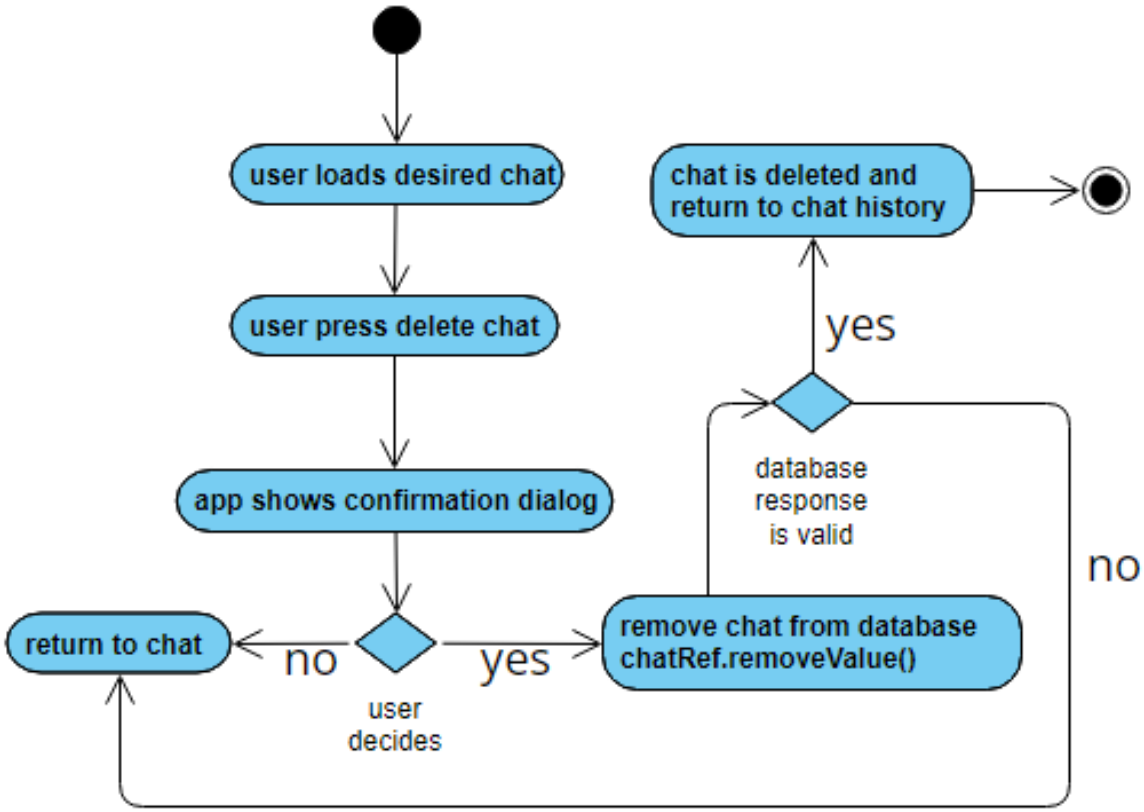
5.3 Display Chat History Activity Diagram



Display Chat History Workflow:

1. User choose the option to start chatting from the main menu and is redirected to the chat history grid.
2. System requests the previous chat topics and corresponding chat ids from the Firebase Realtime Database.
3. If the database response is not valid user is prompted and redirected to the main menu.
4. If the database response is valid data is fetched into a custom chat info class.
5. Previous chat topics are displayed on the screen as a grid.
6. User choose their desired chat topic to continue chatting.
7. System redirects user to the chat screen and loads the chat history on the view.

5.4 Delete Chat Activity Diagram

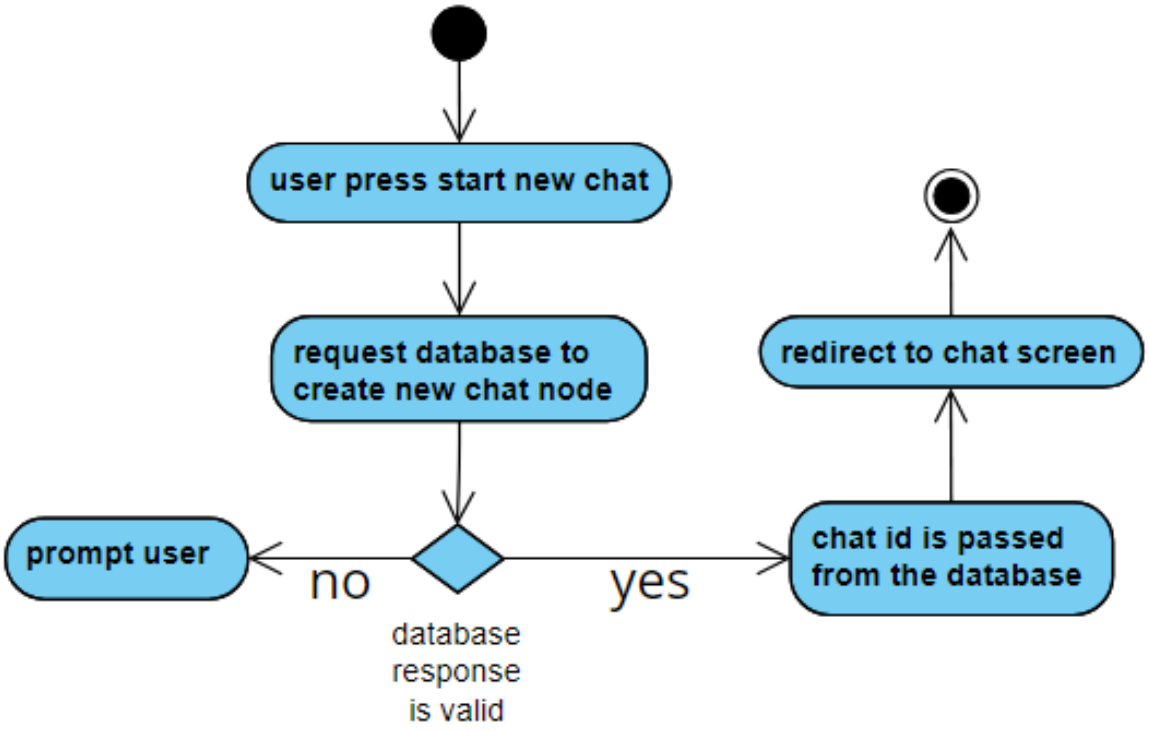


Delete Chat Workflow:

1. User loads desired chat from chat history grid.
2. User press delete chat button and is prompted with a confirmation pop up.
3. If the user doesn't confirm the action, pop up closes and nothing is done to the current chat.
4. If the user confirms the action a request to delete the node is sent to the Firebase Realtime Database.
5. If the response from the database is not valid deletion of chat is not complete and the user returned to the same chat.

6. If the response from the database is valid chat node is deleted from the database and user is redirected back to the chat history grid.

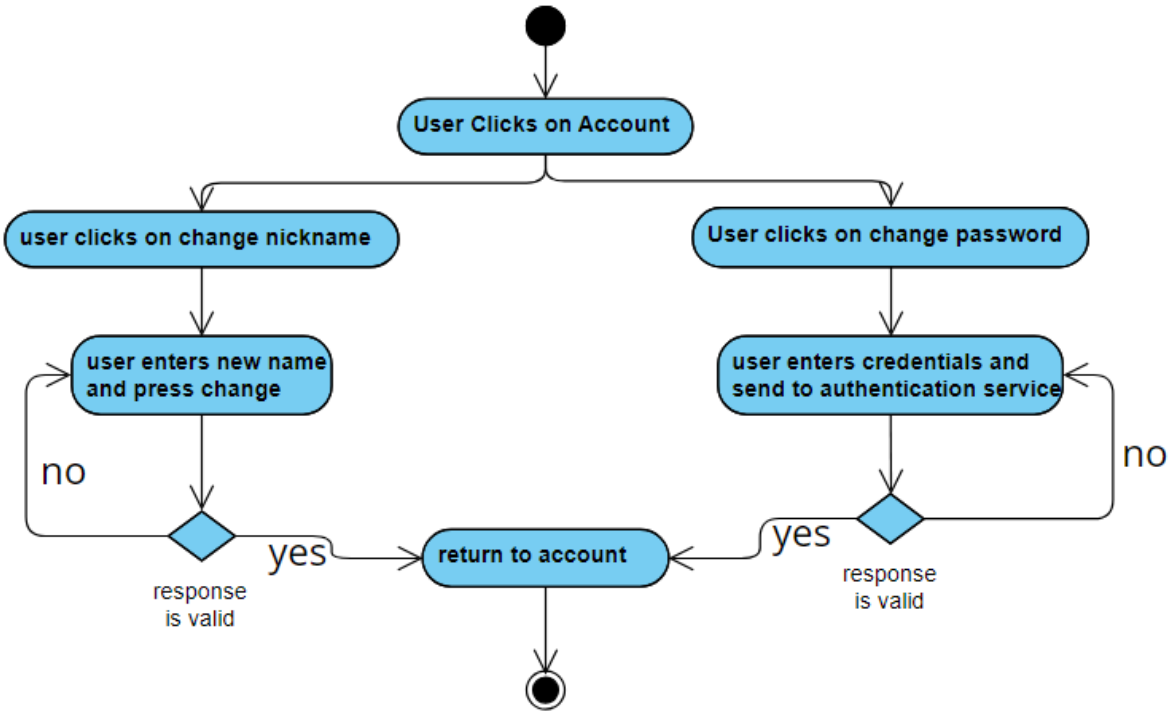
5.5 Start New Chat Activity Diagram



Start New Chat Workflow:

1. User press the start new chat button from the chat history grid.
2. A request to create a new chat node is sent to the Firebase Realtime Database.
3. Upon un-valid response user is prompted. Upon a valid response Chat id is fetched from the database and user is redirected to the chat screen.

5.6 Change Password/Nickname Activity Diagram



Change Password/Nickname Workflow:

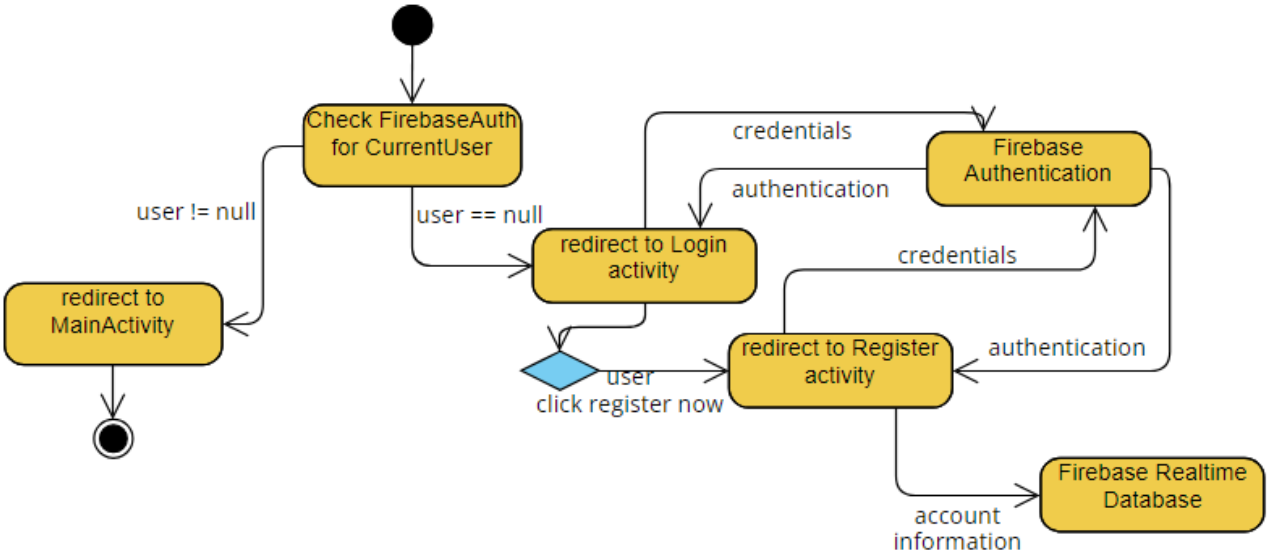
1. User clicks on Account from the main menu and is redirected to the account screen.
2. From the account screen user can choose to change their password. Upon clicking on change password user is redirected to the change password screen.
3. User then enters credentials and new password and press change password button. Request is sent to the Firebase Authentication. Upon un-valid response password is not changed and user is prompted. Upon valid response password is changed on the Firebase Authentication service and the user is redirected back to the main menu.
4. From the user account screen user choose to change their nickname and is redirected to the change nickname screen. User enters desired new

nickname and a request is sent to the Firebase Realtime Database to update the user object for the current user.

5. Upon un-valid response from the database username is not changed and the user is prompted
6. Upon a valid response from the database username is changed and the user is redirected back to the main menu.

6. State Diagrams

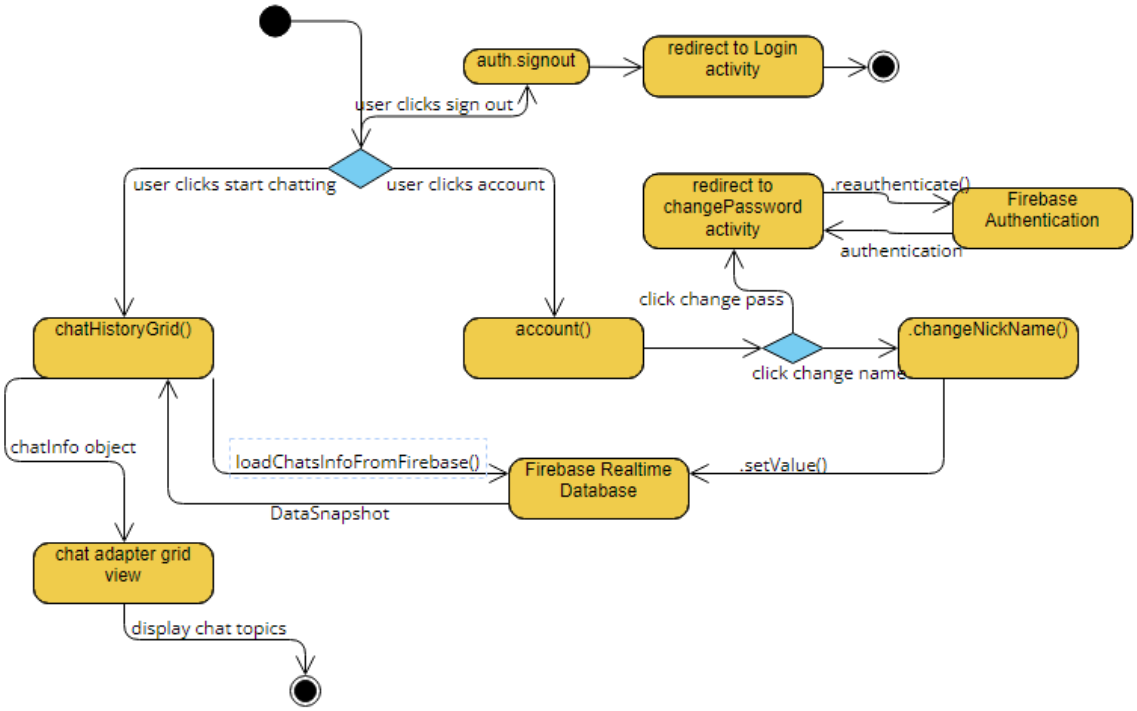
6.1 Initial State/User Authentication State Diagram



Description:

1. Upon app initialization system checks whether user authentication is valid. If user is already authenticated redirect user to the MainActivity.If the user is not authenticated redirect user to the Login activity.
2. User can login using their credentials in that case credentials are sent to Firebase Authentication. If the response includes a valid authentication user is logged in.
3. User can also choose to register a new account. Credentials are sent to the Firebase Authentication and if response is valid a new user object is created with provided account information and is saved on the Firebase Realtime Database.

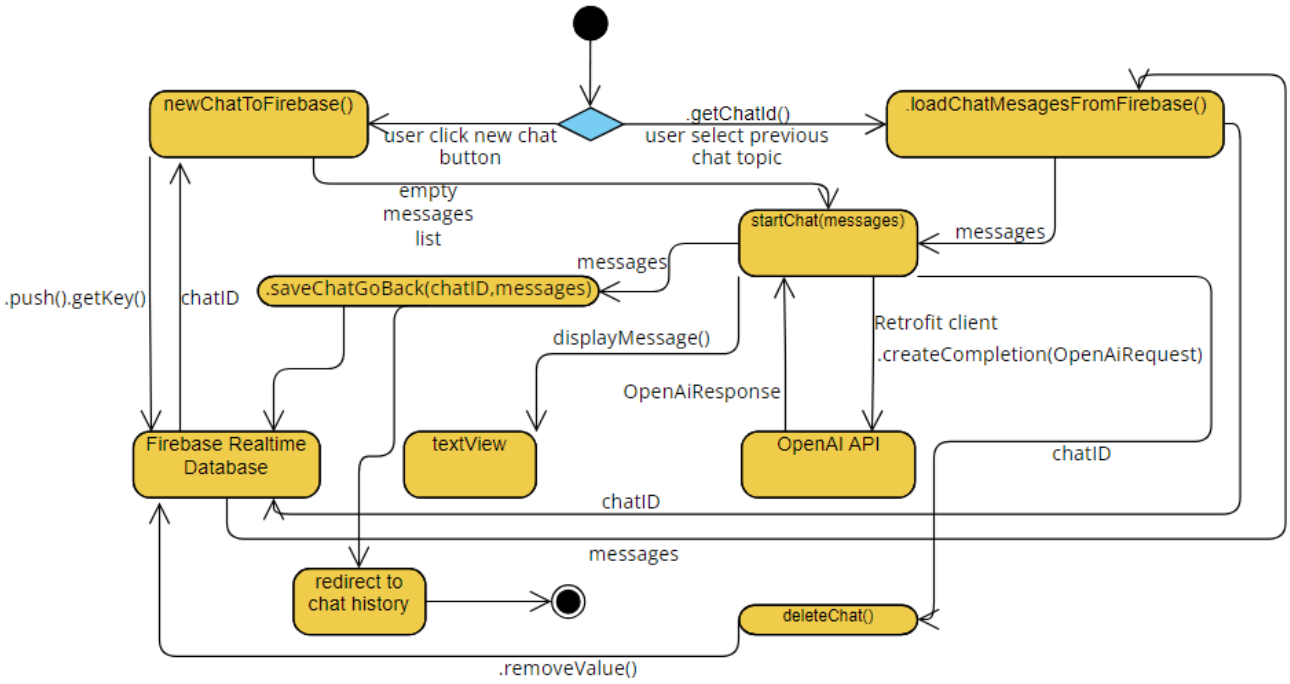
6.2 Main Menu and Account Management State Diagram



Description:

1. From the main menu user can choose to view their account information, start chatting or logout.
2. If the user logs out they are sent to the Login activity.
3. If the user starts chatting previous chat topics are fetched from Firebase Realtime Database and displayed on the grid.
4. If user want to alter account information, they can change their username and password.
5. Change password activity communicates with Firebase Authentication and changes the user password with the new password.
6. User can also update their username on the Firebase Realtime Database.

6.3 Chat Screen State Diagram



Description:

1. User can choose to start a new chat or continue on a previous chat by clicking on the desired chat topic.
2. Upon starting a new chat user is redirected to chat screen. A new messages list is created and a new chat entry is initialized in the Firebase Realtime Database and returned a chatID for the new chat. Messages list is then passed to the startChat() function.
3. Upon clicking a previous chat topic corresponding chatID is fetched from chatInfo class and chat history is fetched from Firebase Realtime Database. Chat history is then passed to the startChat() function.
4. If the user inputs a message and press send message button the message is displayed and added to the current messages list and passed to the OpenAI API. Response from the API is displayed and also cast into a message object and added to the messages list.

5. The user is done with the chat and wants to go back to the chat history. They press the go back and save button and the messages list is saved to the Firebase Realtime Database with the current chatID.
6. User also has the option to delete chat histories. Pressing the delete chat button at the chat screen will delete the chat node at the given chatID from the Firebase Realtime Database.

7. Project Plan

7.1 Project Objective

The aim of this project is to develop a chatbot application using OpenAI's API with Firebase Authentication for user authentication and Firebase Realtime Database for chat history management. The application focuses on an intuitive user interface, efficient API integration, and essential account management features.

7.2 Project Management Approach

We have adopted agile approach. This method allows us to break the project into manageable phases, prioritize key functionalities, and deliver a Minimum Viable Product quickly. Through iterative development cycles, we have progressively added additional features and refined the application based on our own feedback and applications evolving requirements.

7.3 Project Timeline

Planning

• Requirements Analysis. (Day 1)

• Define scope and features to implement. (Day 1)

• Break down features. (Day 1)

• Agree on architecture design for minimum viable product. (Day 2)

Development

• Design and implement interface for chat screen and implement OpenAI API into the chat. (Days 3-6)

• Implement login and registration using Firebase Authentication. Minimum Viable Product is complete. (Day 7)

• Add main menu for better navigation between the chat and account parts. (Day 8)

• Add account tab to view account information. (Day 8)

• Implement Firebase Realtime Database to store chat history and user account information. (Day 8-10)

• Implement chat deletion function for previous chats. (Day 11)

• Implement change password and change name function. (Day 12)

User Interface Enhancements

• Refine the user interface and views for ease of use and better visuals. (Day 13)

Testing

• Refine error handling throughout the whole project. (Day 14)

8. Architecture and Design

8.1 Overview

The app consists of three main components; client side, server side, database.

• Client Side:

Responsible for user interface, user input, and displaying chat messages.

Communicates with the server using Retrofit for API requests.

• Server Side:

OpenAI API, processes user queries and generates AI responses.

OpenAI API acts as the core logic provider, ensuring accurate responses are delivered back to the client.

• Database:

Firebase Authentication manages user authentication.

Firebase Realtime Storage manages user account information and user chat histories, ensuring persistence across sessions.

Client-Server Architecture ensures that the client only handles user interface while OpenAI API focuses on business logic.

Firebase Authentication provides a robust authentication and management while Firebase Realtime Storage provides a realisable database to store and retrieve users chat histories.

Retrofit along with functions such as interceptors from OkHTTP simplifies handling HTTP requests between the client and the OpenAI API. Also enables us to write clean and modular networking code.

8.2 Design Patterns Used

• Model-View-Controller:

•**Model:** Components like OpenAiRequest and OpenAiResponse handles data logic and communication with the OpenAI API. Components like Users and chatInfo handles data logic with Firebase.

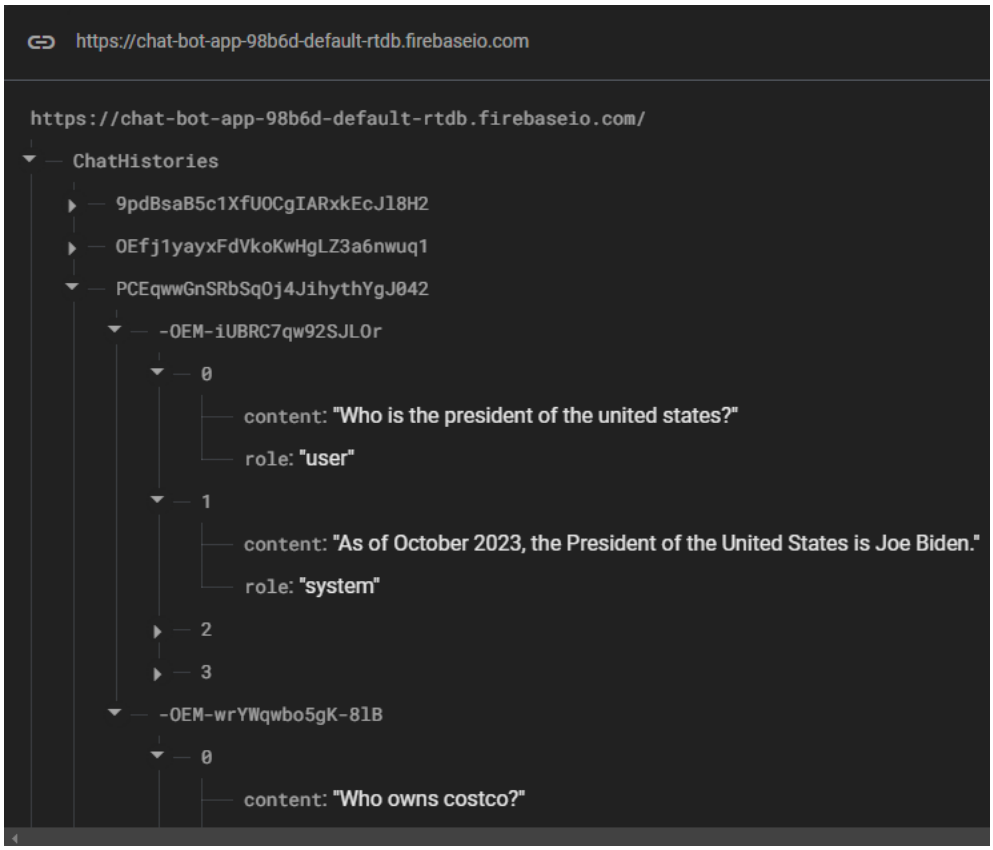
•**View:** Layouts and user interface components handles the user's interaction with the application such as input/output.

•**Controller:** Methods inside MainActivity handles the application logic and calls the model methods to interact with Firebase and OpenAI API. Also updates user interface with the data gathered from model.

•Singleton Pattern:

• Retrofit API client is implemented with singleton pattern. We ensure only a single instance of retrofit client is created and manages all network communications.

8.3 Database Design



•Above is the database design of how the chat histories are stored. Under each user id is a unique chat id. Every chat node contains message objects passed onto them by the application. Every chat entry has their content and the author which is indicated as role.

•Above is the general view of the both classes the database houses for the application. Under users every user is denoted by their unique user id and every user node holds the users account information, to name them e mail and name.

9. Testing

Overview:

All testing was done manually. Tests cases and test scenarios were conducted to test both functional and non-functional requirements.

9.1 Test Scenarios

• Check Whether Chat Histories Are Displayed Properly

Assumption:

User is logged in.

Stable Internet Connection

Steps:

Navigate to the main menu.

Press “Start Chatting” button.

Expected Result:

Chat history is displayed with chat topics in a grid with minimal delay.

• Check Whether the User Can Continue Chat Properly

Assumption:

User is logged in.

User has had at least one previous chat with the ai and saved it.

Stable Internet Connection

Steps:

Navigate to the main menu.

Press “Start Chatting” button.

Choose a random chat topic from the history.

Observe chat screen and type a related question and send.

Expected Result:

Every message from the chat history is displayed with indications of their author (user or ai).

AI does respond in the context of the previous chat thus, a true continuation of the chat.

AI doesn't take too long to respond.

While waiting for AI response a progress bar indicator is visible.

•Check Whether the Save and Go Back Works Properly

Assumption:

User is logged in.

Stable Internet Connection

Steps:

Navigate to the main menu.

Press “Start Chatting” button.

Choose to start a new chat or continue one.

Enter a new message.

Go back using save and go back.

Load the same chat again.

Expected Result:

Chat history is updated and shown on the final screen.

• Check Account Information functionality:

Assumption:

User is logged in.

Stable Internet Connection

Steps:

Navigate to the main menu.

Observe if the username is displayed and greeted.

Press “Account” button. Observe if the user is shown his e mail associated with the account.

Press “change name” and change name. Observe if the name has changed.

Press “change password”. Logout using “Logout” button try to log back in with new password.

Expected Result:

User name and e mail is displayed properly.

User can successfully logout and log back in with the new password.

User can successfully update their username.

- **Check Whether User Authentication Works Properly:**

Assumption:

Stable Internet Connection

Steps:

Start the app and try to register.

Logout using “logout” button and try to log back in using credentials.

Logout and try to register with the already registered email.

Try to login with invalid credentials.

Expected Result:

User can register with valid credentials.

User can login with valid credentials.

- **Check Whether the Delete Chat Works Properly**

Assumption:

User is logged in.

User has had at least one previous chat with the ai and saved it.

Stable Internet Connection

Steps:

Navigate to the main menu.

Press “Start Chatting” button.

Choose a random chat topic from the history.

Press “delete chat button”.

Expected Result:

Chat entry is deleted from the database and not shown in chat history grid.

User is redirected to chat history.

- **Check Internet Connection Cut-off During Chat**

Assumption:

User is logged in.

User is actively having a chat.

Internet connection is cut-off mid chat.

Steps:

Navigate to the main menu.

Press “Start Chatting” button.

Choose a random chat topic from the history or start a new chat.

Cut internet connection-off.

Restart the app.

Expected Result:

Chat entry is not saved on the database and is lost.

• Stress Test with Rapid Chat Inputs

Assumption:

User is logged in.

User is actively having a chat.

Stable internet connection.

Steps:

Navigate to the main menu.

Press “Start Chatting” button.

Choose a random chat topic from the history or start a new chat.

Rapidly input messages to the app and send to AI.

Expected Result:

App remains responsive, and all messages are processed in order.

• Retrieve a Chat with More Than One Hundred Messages

Assumption:

User is logged in.

Stable internet connection.

Steps:

Navigate to the main menu.

Press “Start Chatting” button.

Choose a random chat with at least one hundred messages from the history.

Expected Result:

App loads and displays chat history without noticeable delay.

• Memory Usage During Prolonged Use Is Stable

Assumption:

User is logged in.

Stable internet connection.

App is running for at least 30 minutes.

Steps:

Navigate to the main menu.

Press “Start Chatting” button.

Choose a random chat topic from the history or start a new chat.

Continuously chat with the bot for 30+ minutes.

Expected Result:

Memory usage remains stable, and the app does not crash or show degraded performance.

• Test the App User Interface on Different Devices

Assumption:

User is logged in.

Stable internet connection.

More than one device.

Steps:

Navigate and stress test the user interface on different devices.

Expected Result:

User interface remains properly proportioned and provide similar experiences on all devices.

10. Feasibility

This application is feasible to develop. The technologies, costs, and timeline align with the project's scope. The completed project demonstrates that it can be successfully developed and deployed.

10.1 Technical Feasibility

•The application utilizes widely available technologies such as Android Studio, Firebase, and OpenAI's API.

• Android Studio provides a robust platform for building the app, while Firebase offers reliable backend services for user authentication and chat history storage.

• The OpenAI API integrates seamlessly with Android apps through HTTP calls using Retrofit and OkHttp libraries.

• OpenAI API and Firebase are very well-documented, ensuring ease of development and debugging.

10.2 Economic Feasibility

• This is a school project, so the development costs are primarily the time investment.

• OpenAI’s API offers a free trial with sufficient tokens. We can also opt for a low-cost API subscription if needed. Prices are as low as 5 USD.

• Firebase’s free services are more than enough for storing user and chat data for a small-scale application.

10.3 Operational Feasibility

• The application can be implemented within a realistic timeline.

• Features such as user authentication, chat functionality, and data storage/retrieval are pretty straight forward and easy to implement.

• The user interface is simple. Great for non-technical users.

10.4 Social Feasibility

• The application aligns with societal trends toward AI adoption and digital transformation.

• The application contributes positively to education.

Risk Assessment

Risk	Likelihood	Impact	Mitigation Strategy
API Rate Limit Exceeded	Medium	High	Implement throttling mechanism.
OpenAI API Downtime	Low	High	Display an error message to notify users about the downtime.
Integration Bugs with Firebase	Medium	Medium	Perform integration testing.
Scalability	Medium	High	Implement efficient data retrieval methods.
Data Corruption in Firebase	Low	High	Enable Firebase backups and data validation.
App Crashes Due to Heavy Load	Low	High	Perform stress testing, optimize code, and implement proper exception handling.
Unstable Network Connectivity	Medium	Medium	Cache messages locally until network connectivity is restored.
API Key Leak	Medium to High	High	Store API keys securely using encrypted storage.
Loss of Saved Chats by Users	Low	Medium	Implement a "Restore Chat" option to retrieve deleted chats.