**Project Documentation:**

**“Messaging Web Application System”**

**Submitted to:**

Ms. Maria Gul

**Group Members:**

* Muhammad Mubashir
* Sharyar Naveed

**Program:**

SE-22-Red

**Subject:**

Database System

**Course Code:**

COMP-231

**Developed by:**

Muhammad Mubashir & Sharyar Naveed

**Table of Contents:**

1. **Introduction**

1.1 Project Overview

1.2 Project Objectives

1.3 Technologies Used

1. **System Analysis**

2.1 Functional Requirements

2.2 Non-Functional Requirements

1. **Database Design**

3.1 Entity-Relationship Diagram (ERD)

3.2 Schema Definition

1. **Development Process**

4.1 Environment Setup

4.2 User Interface Design

4.3 Messaging Functionality

4.4 Additional Features

4.5 Security Measures

4.6 Testing and Debugging

1. **Deployment**

5.1 App Deployment

1. **Conclusion**

6.1 Summary of Achievements

**1. Introduction**

**1.1 Project Overview**

In the evolving landscape of digital communication, the need for a secure and efficient messaging platform is paramount. This project focuses on developing a Messaging Web Application System using Vue.js, Node.js, Express.js, and MySQL. The application aims to provide a seamless communication tool with features like user authentication, chat history, and file sharing.

**1.2 Project Objectives**

The primary objectives of this project are:

* To develop a functional and secure messaging web application.
* To implement robust authentication mechanisms.
* To ensure data security and integrity.
* To provide a user-friendly interface with essential messaging features.

**1.3 Technologies Used**

* **Frontend**: Vue.js, HTML, CSS, JavaScript
* **Backend**: Node.js, Express.js
* **Database**: MySQL
* **Security**: Fingerprint Authentication, Input Validation
* **Real-time Communication**: AJAX, Socket.io

**2. System Analysis**

**2.1 Functional Requirements**

These are the specific functionalities that the messaging web application must have to meet user needs:

* **User Registration and Authentication**: The system must allow users to register by providing personal data and an avatar. It must include robust mechanisms to authenticate users securely during login, ensuring that unauthorized access is prevented.
* **Chat History Access**: The application should display the 25 most recent messages by default in the chat box. Users must have the ability to scroll back to view older messages, providing continuous access to chat history.
* **Message Deletion Visibility**: When a message is deleted by the sender, both the sender and the receiver should be able to see that the message has been deleted. The deleted messages will be marked as partially erased, indicating that a message once existed but has been removed.
* **File Sending Features**: Users should be able to send various types of files (documents, images, etc.) through the chat, enhancing the utility and functionality of the messaging platform.
* **Fingerprint Authentication**: For added security, the application should integrate fingerprint authentication, ensuring that only authorized users can access their accounts and sensitive information.

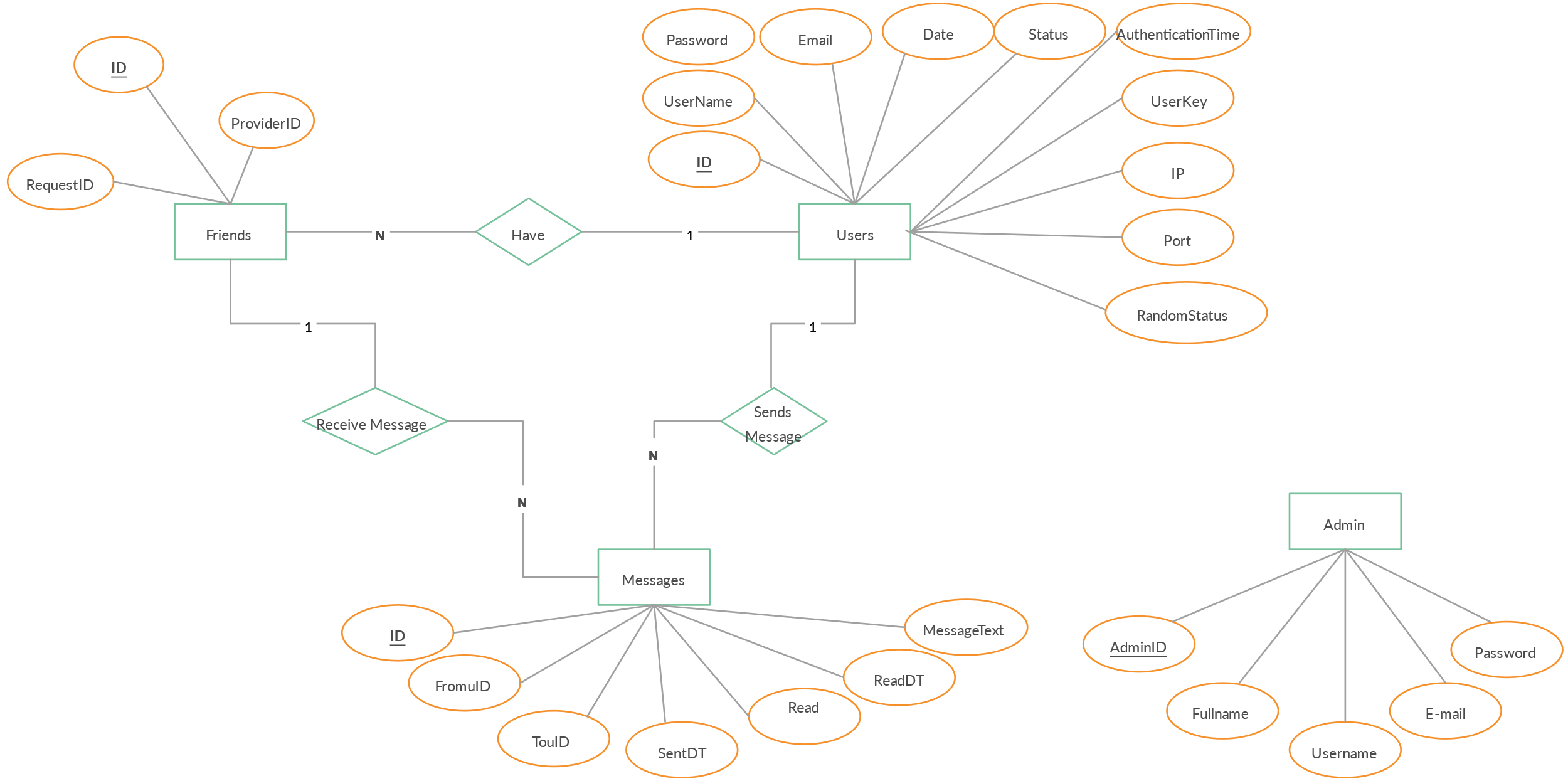
**2.2 Non-Functional Requirements**

These requirements define the overall qualities and attributes of the system, ensuring it meets user expectations beyond functional aspects:

* **Performance**: The application should be responsive, providing quick load times and smooth interactions. It must handle multiple concurrent users efficiently without significant slowdowns or delays.
* **Scalability**: The system should be designed to accommodate an increasing number of users and messages over time. This ensures the application can grow without performance degradation.
* **Security**: All user data, including personal information and chat content, must be encrypted and securely stored to protect against unauthorized access and breaches.
* **User Experience**: The interface should be intuitive and easy to use, providing a seamless and enjoyable experience for all users, regardless of their technical proficiency.
* **Reliability**: The application should have robust error handling to manage unexpected issues gracefully. Additionally, it must include data backup mechanisms to prevent data loss and ensure continuity of service.

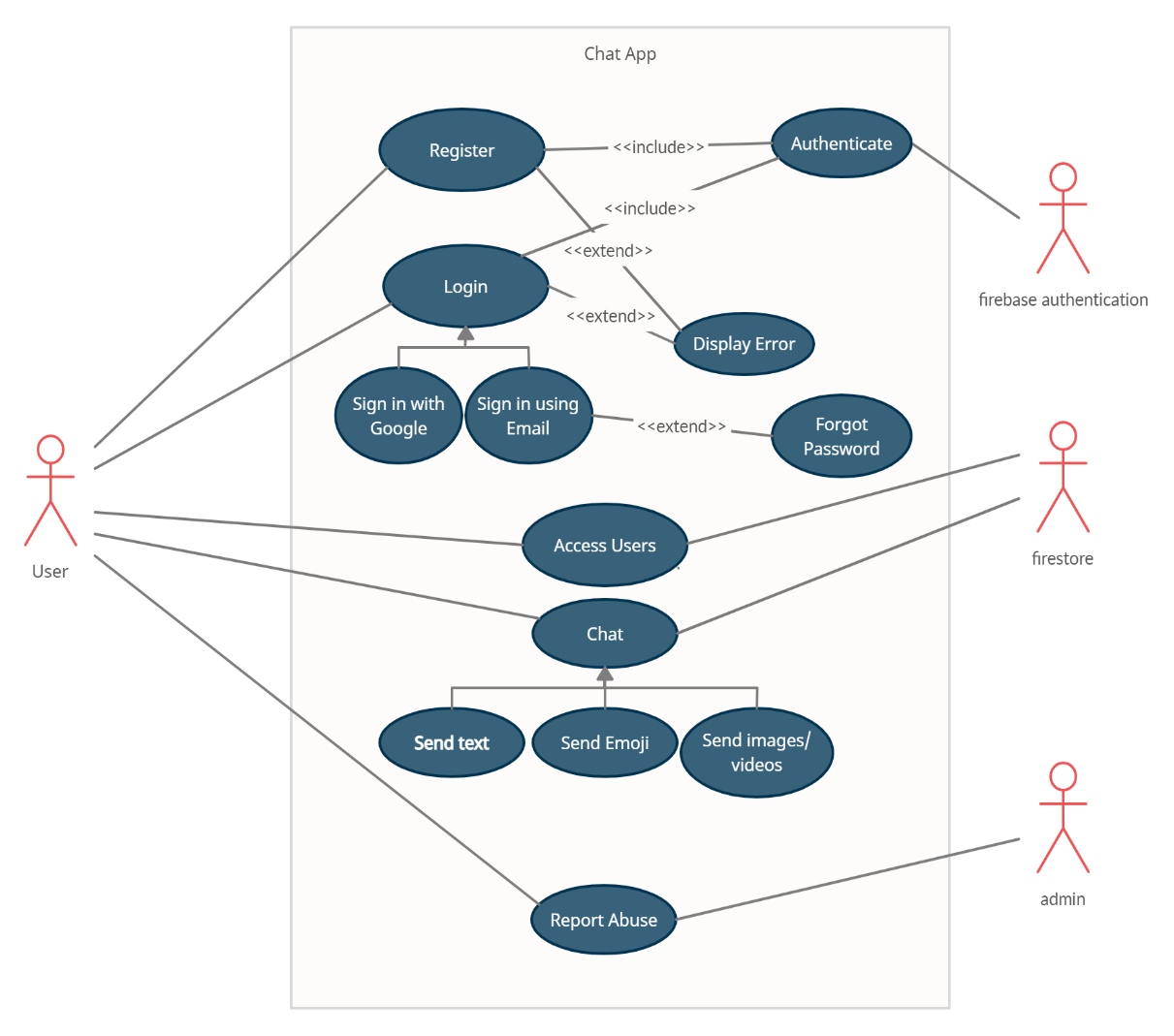
**3. Database Design**

**3.1 Entity-Relationship Diagram (ERD)**

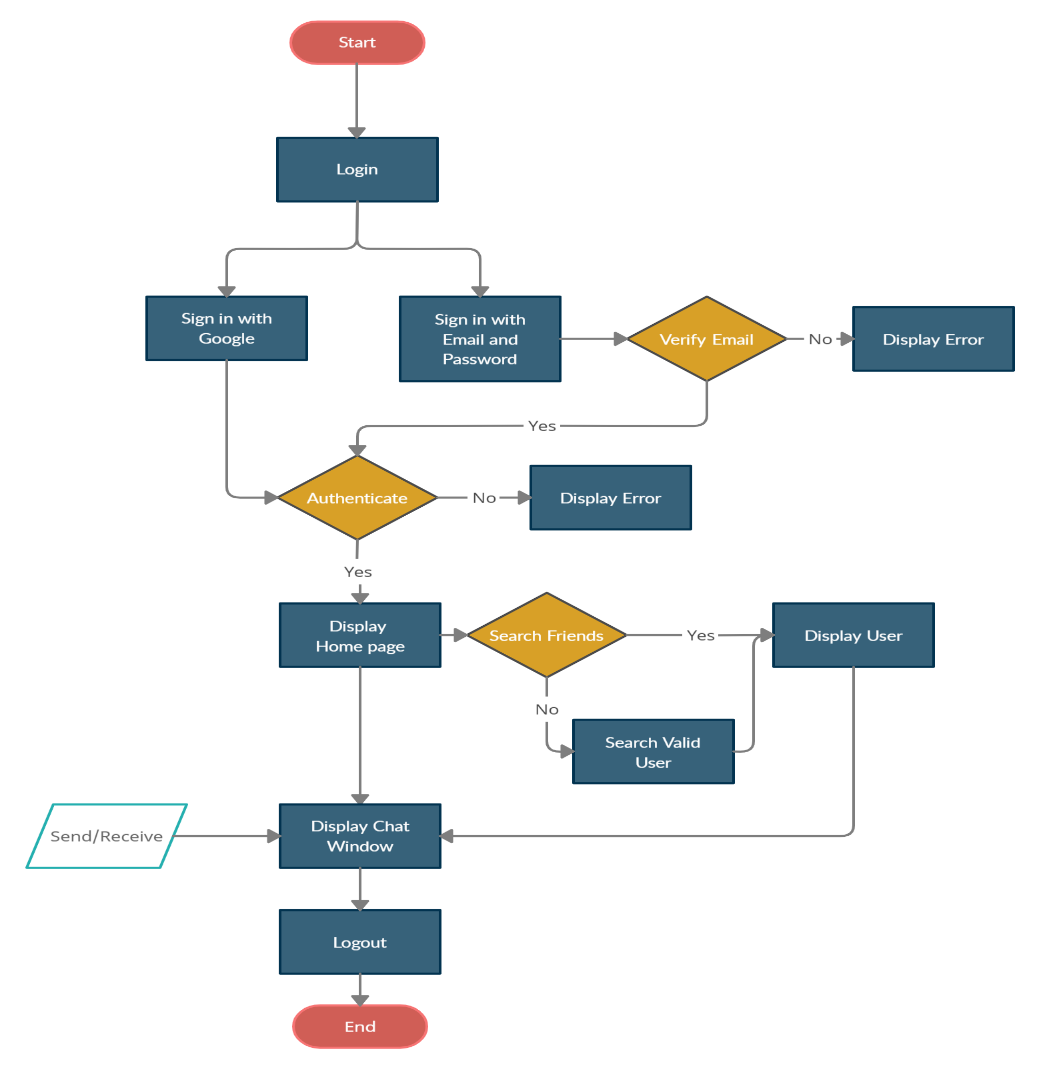
****

**3.2 Schema Definition**

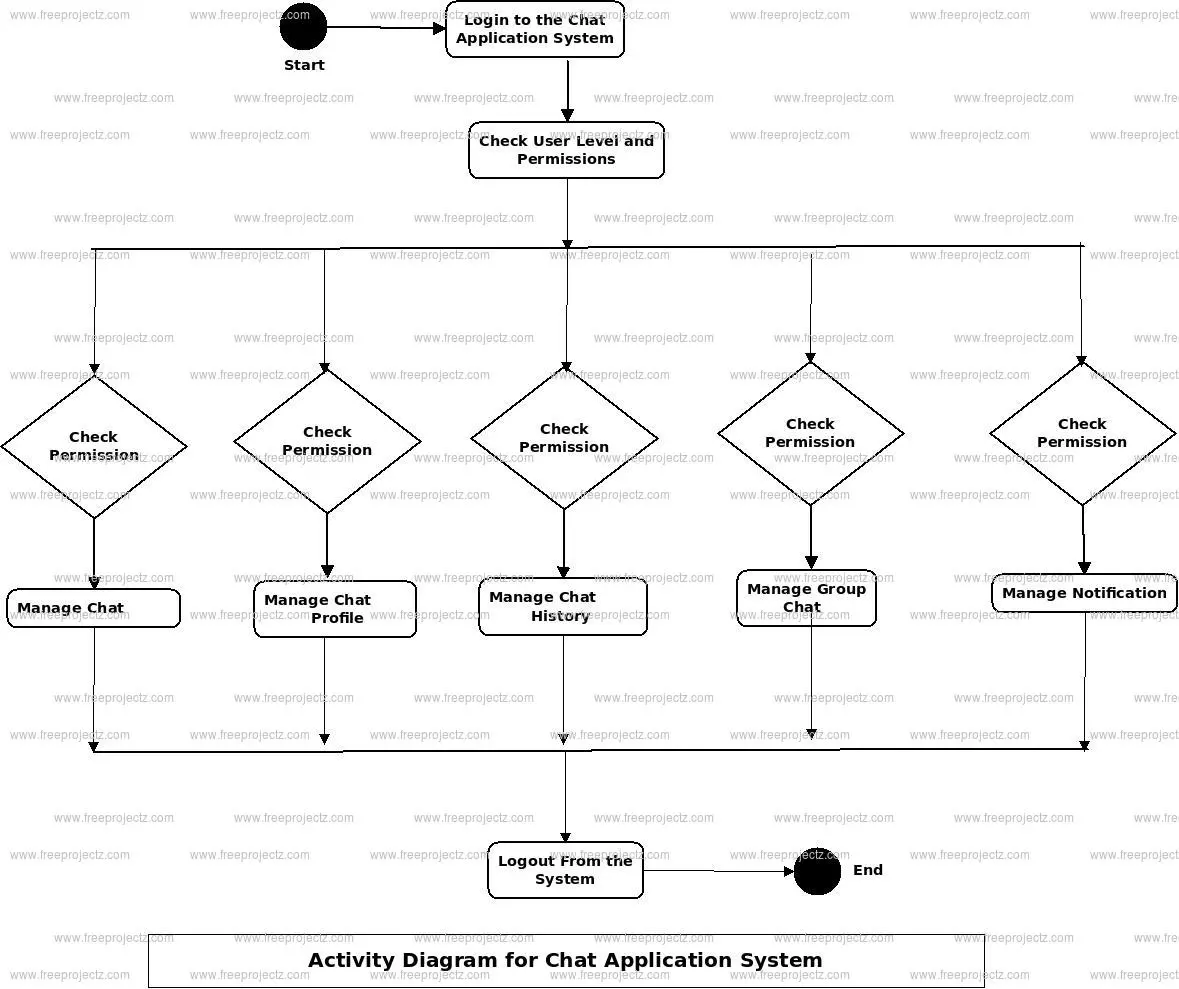
* **Use Case Diagram:**

****

* **Flow chart:**

****

* **UML Diagram:**

****

**Users Table**

* + **id**: A unique identifier for each user (Primary Key).
  + **username**: The user's chosen name for login and display purposes.
  + **password**: The user's encrypted password for secure authentication.
  + **avatar**: The user's profile picture, stored as a URL or binary data.
  + **fingerprint data**: Biometric data used for additional security during authentication.

This table is crucial for managing user information, ensuring secure access, and personalizing user experiences.

* **Conversations Table**
  + **id**: A unique identifier for each conversation (Primary Key).
  + **participant ids**: Identifiers for the users participating in the conversation (could be stored as a JSON array or in a separate join table for many-to-many relationships).
  + **timestamps**: Metadata capturing the creation and last update times of the conversation.

This table helps in managing and organizing conversation threads, allowing users to engage in multiple distinct chats.

* **Messages Table**
  + **id**: A unique identifier for each message (Primary Key).
  + **sender id**: The identifier of the user who sent the message.
  + **conversation id**: The identifier of the conversation to which the message belongs.
  + **content**: The actual message content, which can be text, file paths, or other media types.
  + **status**: Indicates the message status, such as read, unread, or deleted.

This table stores all the messages sent within conversations, providing essential details to retrieve and display messages in the correct context.

**4. Development Process**

**4.1 Environment Setup**

* **Install and configure Express.js and MySQL:** Set up the necessary software to run the server (Express.js) and manage the database (MySQL).

**4.2 User Interface Design**

* **Use HTML, CSS, and Vue.js to create a user-friendly interface:** Design the front-end of the application to ensure it is easy to use.

**4.3 Messaging Functionality**

* **Implement message creation, deletion, and retrieval with real-time updates using PHP and AJAX or Socket.io:** Develop the core messaging features, including sending, deleting, and viewing messages, with real-time updates.

**4.4 Additional Features**

* **Add search functionality, notifications, file attachments, emoji support, and user profiles:** Enhance the app with extra features for better user experience.

**4.5 Security Measures**

* **Implement input validation, authentication, and authorization checks:** Ensure the application is secure by verifying user input and controlling access to the system.

**4.6 Testing and Debugging**

* **Conduct thorough testing and debugging, considering automated testing and security scanning tools:** Test the application extensively to find and fix bugs, and use tools to automate some of these tests and check for security issues.

**5. Deployment**

**5.1 App Deployment**

* Deploy the application on a chosen hosting provider and set up performance and security monitoring: Launch the app on a hosting service and monitor its performance and security.

**6. Conclusion**

**6.1 Summary of Achievements**

* The Messaging Web Application System aims to meet the growing demand for secure and feature-rich communication platforms. By following a structured development process and incorporating user feedback, this project seeks to deliver a reliable and user-friendly messaging solution.