**SECURE CHAT APPLICATION**

**PROBLEM STATEMENT :**

Secure chat application with end-to-end encryption

**PROBLEM DESCRIPTION:**

Real-time communication applications are vulnerable to security threats such as data

interception, unauthorized access, and identity spoofing. This hackathon challenges you to

create a fully secure messaging platform where user messages remain private and

accessible only to the intended recipient. The project will involve cryptographic techniques

and secure API development to ensure confidentiality and integrity.

### **Tech Stack Used**

The project utilizes a combination of web technologies and frameworks to ensure secure and efficient communication:

* **Frontend:**

**HTML5, CSS3, and JavaScript** — For building the user interface, ensuring responsiveness, and handling real-time interactions.

**Socket.IO** — For real-time, bidirectional communication between the client and server.

* **Backend:**

**Flask** — A lightweight Python web framework used to handle HTTP requests, manage user authentication, and manage chat logic.

**Flask-SQLAlchemy** — Used for database management to store user credentials and chat messages securely.

**Flask-JWT-Extended** — Used for secure user authentication via JSON Web Tokens (JWT).

* **Database:**

**SQLite** — Chosen for its simplicity and lightweight nature, ideal for this project.

### **Approach Followed**

The project was developed in a structured manner to ensure modularity, security, and user-friendliness:

1. **Frontend Development:**

Designed a clean and organized UI for a chat interface using HTML, CSS, and JavaScript.

Integrated Socket.IO to enable real-time message exchange.

Added a file upload feature with proper UI controls for seamless file sharing.

1. **Backend Development:**

Utilized Flask to handle user registration, login, and secure session management.

Implemented message handling with Socket.IO to broadcast messages and files between clients.

Designed endpoints to manage file uploads securely using the uploads/ directory for file storage.

**3.Database Management:**

Designed two key tables:

**User Table:** To manage user authentication and registration.

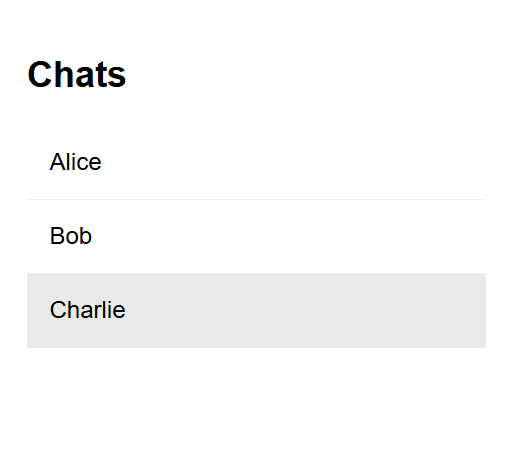
**Message Table:** To store chat data securely with timestamps.

Ensured efficient data retrieval and storage for better scalability.

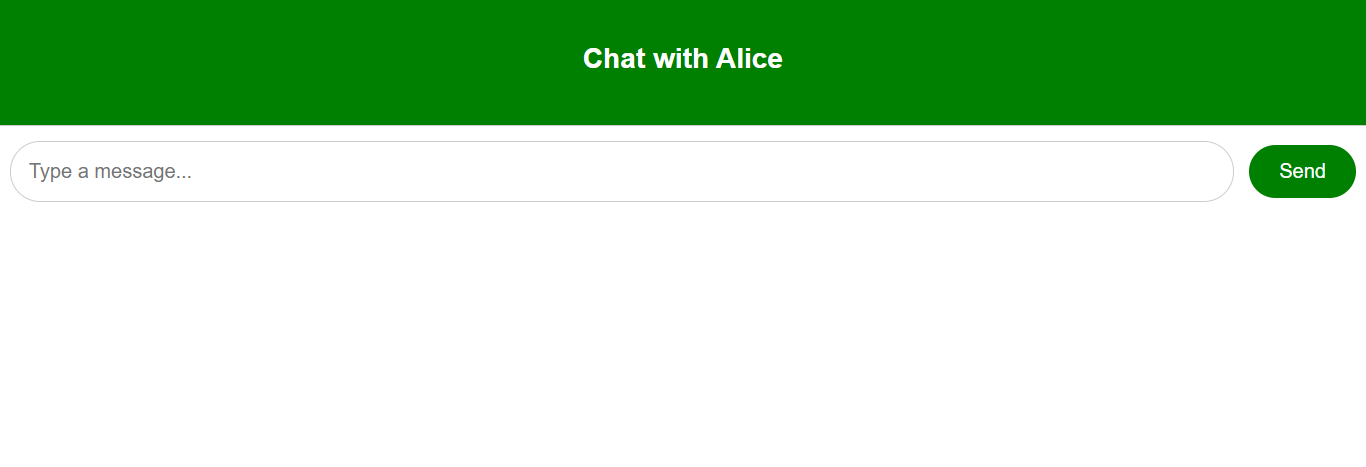
Conducted unit tests to verify message delivery, file upload handling, and database operations.

Ensured the user interface responded smoothly with real-time updates via Socket.IO.

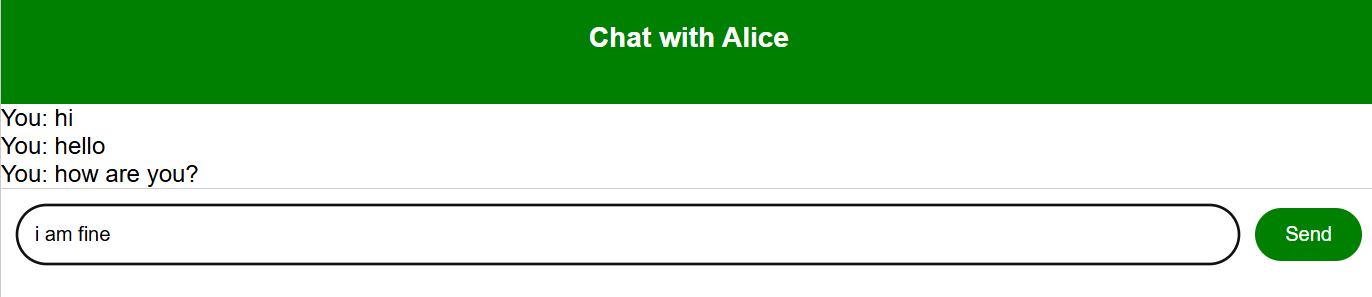
**Arrived Output:**



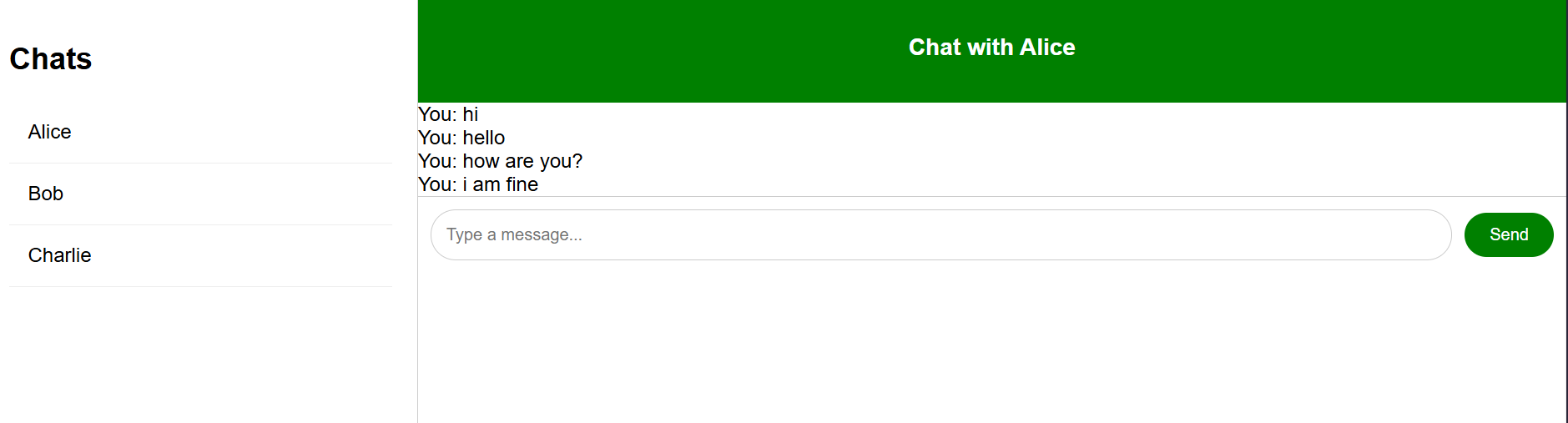
The registered users will be visible in the chat screen.



Each user will have a separate chatbox.



The messages sent will be visible in the screen.



This is how the overall screen looks.