Suggested Project Structure

```
secure-chat-app/
├── app/app.py
│   ├── static/chat.js
│   ├── templates/chat.html
│   └── crypto.py
├── report/
│   └── Secure_Chat_App_Report.pdf  ← Place your report here
├── requirements.txt
└── README.md
```

- `app/`: Your main application logic and modules.
- `report/`: A dedicated folder for documentation and project reports.
- `Secure_Chat_App_Report.pdf`: Save the PDF you generate with the content I gave you inside this folder.
- `requirements.txt`: Lists the dependencies.
- `README.md`: A general overview of the project and instructions

Internship Project Report

Project Title:

Secure Chat App with End-to-End Encryption

Objective:

Create a private chat application with End-to-End Encryption (E2EE) using public-key cryptography.

Tools & Technologies Used:

- Python
- Flask-SocketIO
- RSA / AES (using `cryptography` library)

Mini Guide:

a. Generate RSA keys per user and share public keys.
b. Encrypt messages with AES; share AES keys using RSA.

c. Establish real-time communication using Flask-SocketIO.
d. Optionally store chat logs encrypted on the server.
e. Decrypt messages only on the client side for true E2EE.

Deliverables:

- Secure Chat App with:
    - User Login and Registration
    - Group Chat
    - End-to-End Encrypted Messaging
    - Encrypted Message Logs

Objective:

To develop a secure, real-time chat application that ensures **End-to-End Encryption (E2EE)** using public-key cryptography. The project is inspired by real-world applications chat app .

Tools & Technologies Used:

- Python 3
- Flask (Web Framework)
- Flask-SocketIO (Real-time WebSocket Communication)
- JavaScript, HTML, CSS
- Cryptography Library (for RSA and AES encryption)

Mini Guide (Core Functionality):

a. Generate RSA keys per user and share public keys.
b. Encrypt messages using AES. AES key is exchanged securely using RSA.
c. Use Flask-SocketIO for real-time communication between users.
d. Store chat logs encrypted on the server (optional feature).

e. Ensure messages are decrypted **only** on the client side, making the server blind to message contents.

Modules Developed:

1. **User Registration & Login**
    - Credentials are encrypted using RSA during registration and login.
    - Each user gets a unique RSA key pair (private and public).
2. **Chat Interface (Frontend)**
    - Built using HTML, CSS, and JavaScript.
    - Users can send and receive messages in real-time.
3. **Real-Time Chat (Backend)**
    - Flask and Socket.IO manage message broadcasting to groups or individuals.
    - The server only transmits encrypted content and never decrypts it.
4. **Encryption Logic**
    - RSA used for secure key exchange.
    - AES used for encrypting actual message content.
    - Only the receiving user can decrypt the message with the AES key received securely.
5. **Group Chat Functionality**
    - Users can select or create groups.
    - Messages sent in groups are encrypted and sent to all online members.

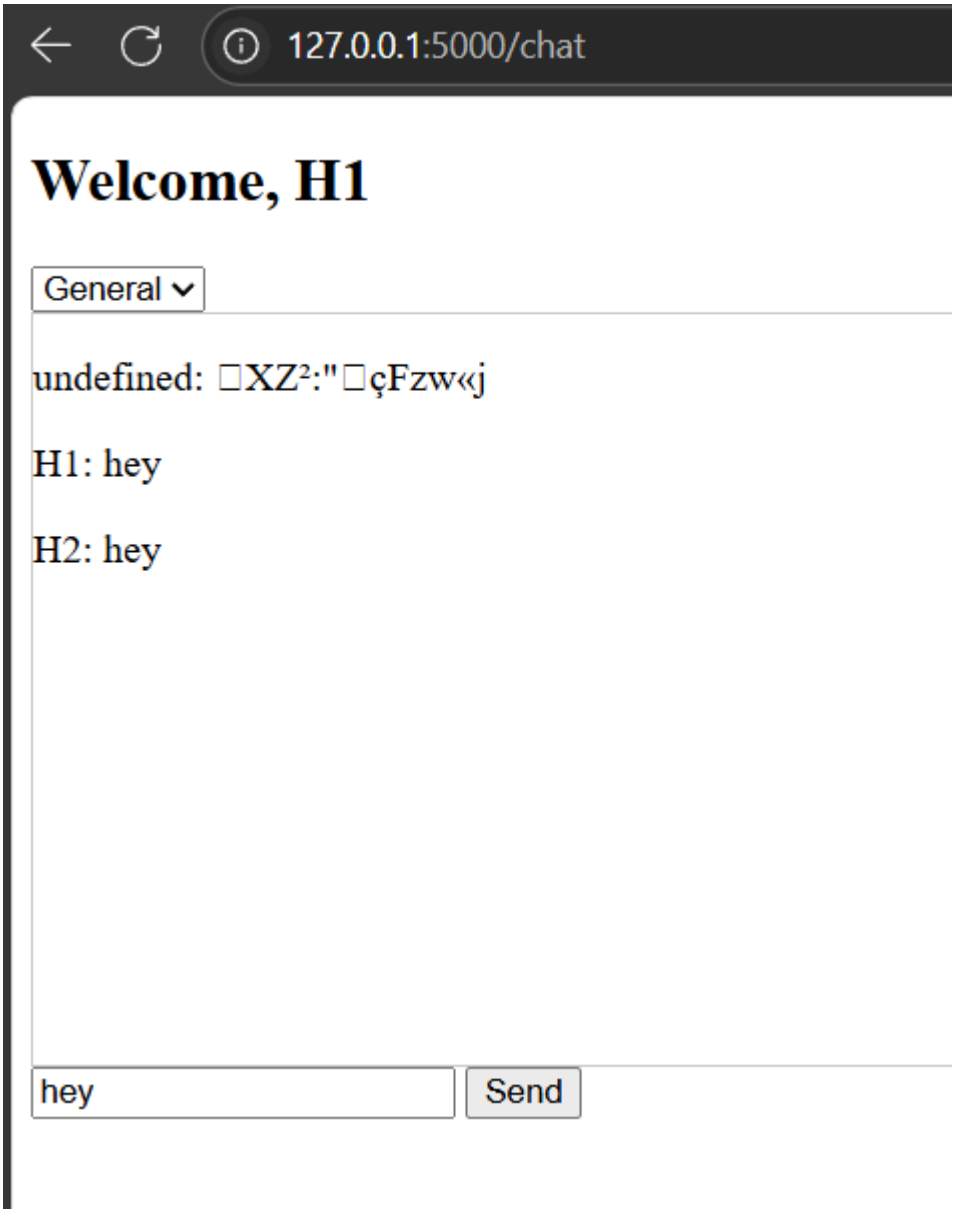
Deliverables:

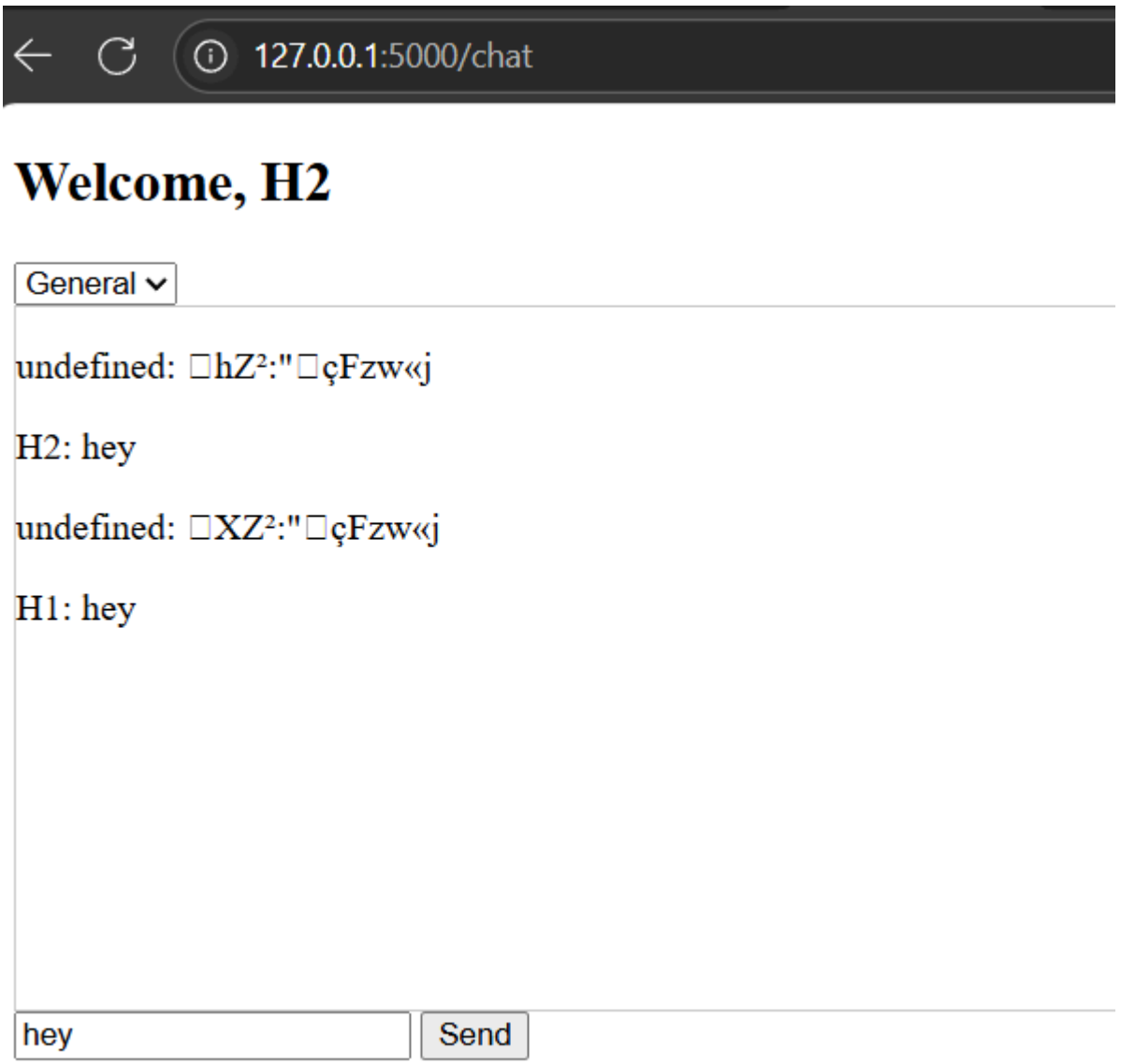
- A fully functional secure chat application with:

    - User registration and login
    - Real-time encrypted chat system
    - Group chat support
    - Client-side decryption logic
    - Basic persistent user storage
- Final submission includes:
    - Full source code in a `.zip` file
    - Setup guide / README
    - Internship report document

Learning Outcomes:

- Practical implementation of public-key cryptography (RSA) and symmetric encryption (AES).
- Understanding and use of Flask for backend services.
- Real-time communication using Flask-SocketIO.
- Building secure communication platforms and managing encrypted data transmission.
- Gained experience in web development and cryptographic protocol integration.


127.0.0.1:5000/chat
Welcome, H1
General
undefined: □XZ²:"□çFzw«j
H1: hey
H2: hey
hey
Send


127.0.0.1:5000/chat
Welcome, H2
General
undefined: □hZ²:"□çFzw«j
H2: hey
undefined: □XZ²:"□çFzw«j
H1: hey
hey
Send