

WebSocket Secure Chat Application Documentation

1. Project Overview

This project implements a WebSocket-based secure chat application using TypeScript. It facilitates real-time, secure communication between multiple users. The application consists of a WebSocket server that handles connections and message relay, and a client-side application that users interact with for messaging and other features. The frontend is hosted on InfinityFree, and the backend services are hosted on [Render.com](https://render.com) and Firebase.

2. Features

The application boasts a comprehensive set of features designed for a modern and secure chat experience:

- **Secure Communication:** Utilizes WebSocket Secure (wss://) over HTTPS for encrypted data transmission.
- **User Authentication:** Secure user login with username and password, including account creation with hashed credentials (using bcrypt) for enhanced security.
- **Real-time Messaging:** Instantaneous message delivery between connected clients. Supports both broadcast messages to all users ("All Chat") and direct private messages between individuals.
- **End-to-End Encryption Exploration:** Implements concepts of end-to-end encryption, using AES-256 for message content and RSA-4096 for cryptographic key exchange between clients.
- **Secure File Transfers:** Enables users to send and receive files securely within private chats. The transfer process includes encryption of the file content.
- **Rich Media and Formatting:** Supports sending emojis and basic text formatting such as bold, italics, strikethrough, and code blocks in messages.
- **Active User Tracking:** Displays a list of currently online users.
- **Typing Indicators:** Shows when other users are typing in the current chat context.
- **Auto-Reconnect:** Clients attempt to automatically reconnect to the server if the connection is lost.
- **Heartbeat Mechanism:** Maintains connection health and detects unresponsive clients through a ping/pong mechanism.
- **Logout Functionality:** Users can explicitly log out of the application.
- **Security Hardening:** Includes measures like brute-force protection for login.

attempts and robust logging.

3. Technology Stack

The application is built using a modern technology stack for both frontend and backend development.

3.1. Frontend

- **Framework/Library:** React
- **Language:** TypeScript
- **UI Components:** Shadcn/UI, providing a set of reusable and accessible components.
- **Styling:** Tailwind CSS for utility-first CSS styling, with PostCSS for processing.
- **Build Tool:** Webpack is used for bundling the frontend assets.
- **Key Libraries:**
 - emoji-picker-react: For integrating an emoji picker.
 - dompurify: For sanitizing HTML content to prevent XSS attacks.
 - marked: For parsing Markdown-like text formatting in messages.
 - react-textarea-autosize: For automatically adjusting the height of the message input area.
 - firebase: Client SDK for interacting with Firebase services (specifically Storage for file uploads).
 - lucide-react: For icons used throughout the user interface.

3.2. Backend

- **Runtime Environment:** Node.js
- **Language:** TypeScript
- **WebSockets:** The ws library is used for implementing the WebSocket server.
- **Database:** Firebase Firestore is utilized for persistent data storage, including user accounts and chat history.
- **Key Libraries:**
 - bcrypt: For hashing and comparing user passwords securely.
 - firebase-admin: Server-side SDK for interacting with Firebase services, particularly Firestore.

3.3. Security

- **Transport Layer Security:** Communication between the client and server is secured using wss:// (WebSocket Secure), which relies on HTTPS encryption.
- **Credential Security:** User passwords are not stored in plaintext; instead, they are

hashed using bcrypt.

- **Message Encryption:**

- Messages between the client and server are encrypted.
- For end-to-end encryption between clients, AES-256 is used for message content, and RSA-4096 is employed for the exchange of AES keys. The client generates an RSA key pair, shares the public key, and uses it to encrypt/decrypt shared AES keys for direct messages.

- **File Transfer Security:** Files are encrypted before transfer using AES-GCM, with the AES key itself being exchanged securely using RSA-OAEP between the involved clients.

- **Rate Limiting & Brute-Force Protection:** The server implements rate limiting to prevent abuse and brute-force protection mechanisms for login attempts.

- **Input Sanitization:** On the frontend, DOMPurify is used to sanitize user-generated content before rendering it as HTML, mitigating cross-site scripting (XSS) risks.

4. Hosting

- **Frontend:** The client-side React application is hosted on InfinityFree.com.
- **Backend:** The Node.js WebSocket server is hosted on Render.com and associated database and image hosting services are managed and hosted on Firebase.com.

5. Project Structure

The project is organized into a clear directory structure:

- **syEVERS-cpsc-455-project/:** Root directory.
 - **README.md:** Project overview, features, and setup instructions.
 - **CHANGELOG.md:** Log of changes and version history.
 - **components.json:** Configuration for Shadcn/UI.
 - **index.html:** Main HTML file for the frontend application.
 - **package.json:** Lists project dependencies and scripts.
 - **tailwind.config.js, postcss.config.js:** Configuration files for Tailwind CSS and PostCSS.
 - **tsconfig.json:** TypeScript compiler options.
 - **webpack.config.js:** Webpack bundler configuration.
 - **src/:** Contains the source code.
 - **App.tsx:** Main React component for the frontend UI and client-side logic.
 - **server.ts:** Backend WebSocket server implementation.

- **client.ts**: Original command-line client (likely superseded by the React UI).
- **globals.css**: Global styles for the application.
- **index.tsx**: Entry point for the React application.
- **components/ui/**: Contains Shadcn/UI components like button.tsx, card.tsx, input.tsx, etc..
- **lib/utils.ts**: Utility functions, including cn for merging Tailwind classes.
- **accounts.json**: Potentially an initial or backup storage for user accounts, though Firestore is the primary user data store.

6. Key Functionalities Detailed

6.1. Real-time Messaging

Users can send and receive messages in real-time. Messages can be sent to "All Chat" (broadcast to every connected user) or as direct messages to a specific user. The UI updates instantly as new messages arrive.

6.2. User Authentication and Session Management

Users register or log in with a username and password. The server validates credentials against data stored in Firebase Firestore. Upon successful login, the user's state is updated, and they can participate in chats. Logging out terminates the session and updates the user's status.

6.3. Secure File Transfers

In private chats, users can initiate file transfers. The process involves:

1. The sender selects a file.
2. The file is encrypted client-side using AES-GCM; the AES key is encrypted with the recipient's RSA public key.
3. A file transfer request containing metadata (filename, size, type, encrypted AES key, IV) is sent to the recipient via the server.
4. The recipient can accept or reject the transfer.
5. If accepted, the encrypted file content is sent in chunks.
6. The recipient decrypts the AES key using their RSA private key and then uses the AES key to decrypt the file chunks.
7. Uploaded files are also stored in Firebase Storage.

6.4. Rich Text Formatting and Emojis

Messages support Markdown-like syntax for **bold**, *italics*, ~strikethrough~, and code blocks. An emoji picker allows users to easily insert emojis into their messages. The

frontend uses marked to parse the formatting and DOMPurify to sanitize the output before rendering.

6.5. Typing Indicators

When a user is typing a message in a chat, other participants in that chat (either "All Chat" or a direct message) will see an indicator showing that the user is typing. This is managed by START_TYPING and STOP_TYPING messages sent between the client and server.

6.6. User and Conversation Management

The UI displays a list of online users and users with whom the logged-in user has had past conversations. Users can switch between "All Chat" and private conversations. The availability of a peer's public key (necessary for secure direct messaging and file transfer) is indicated in the UI.

7. Project Evolution (Key Milestones)

The project has evolved through several versions, adding features and refining functionality:

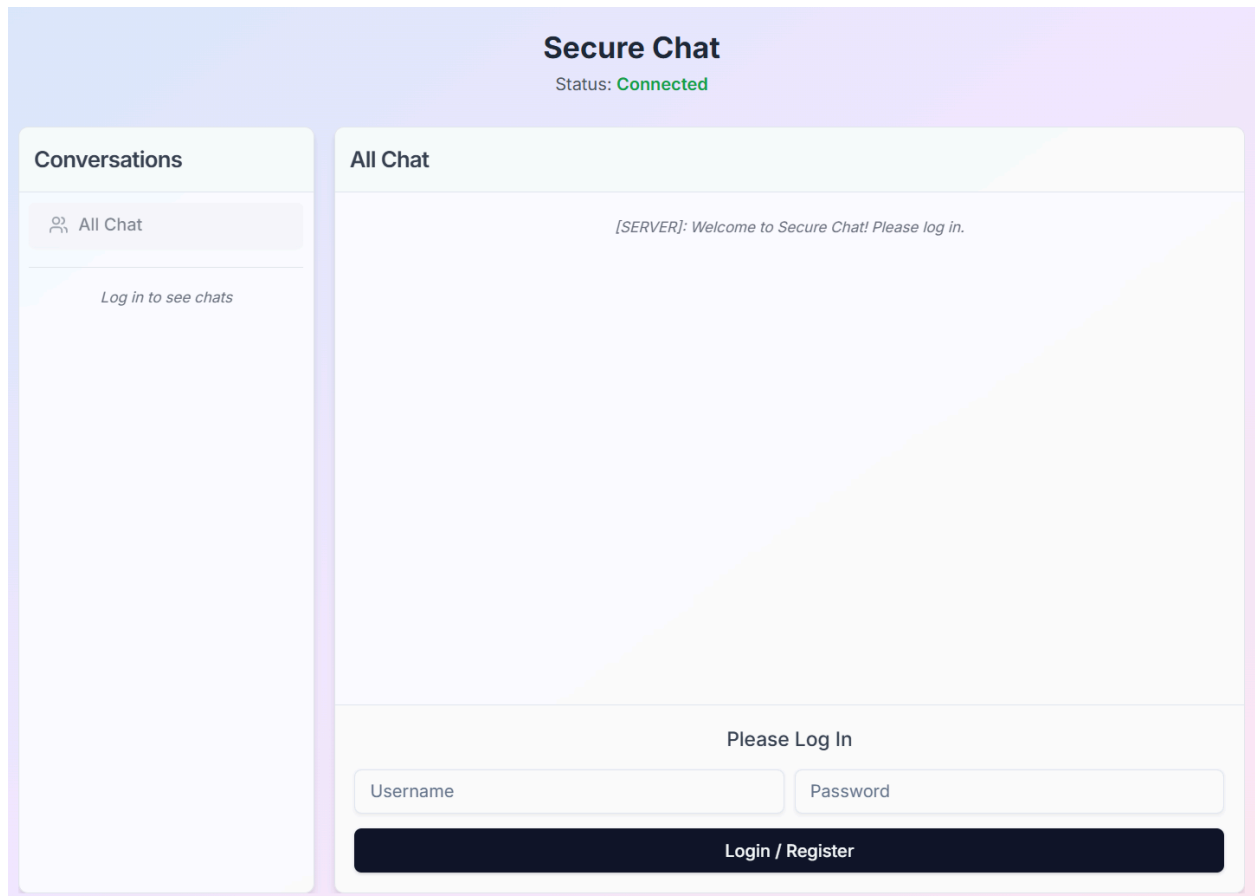
- **[0.1.0] - 2025-02-03:** Initial commit, client and server code work in progress.
- **[0.1.1] - 2025-02-08:** Implemented secure WebSocket (wss) connections and began authentication work.
- **[0.1.2] - 2025-02-12:** Fixed login issues and improved server connection information.
- **[0.1.3] - 2025-02-14:** Enabled client-to-client communication, logout functionality, file sending in chat, UI clarity for server, rate limiting, auto-reconnect, and fixed active user list.
- **[0.1.4] - 2025-02-14:** Program closes connection after logout, fixed account storage, added duplicate username detection, and updated README.
- **[0.2.1] - 2025-03-22:** Enhanced password security with hashing and salting.
- **[0.2.2] - 2025-03-23:** Added brute-force protection and secure file transfer.
- **[0.3.0] - 2025-04-10:** Developed a new client UI using React and Shadcn/UI components.
- **[0.3.1] - 2025-04-12:** Added direct messaging between users; broadcast and direct messages are encrypted.
- **[0.3.2] - 2025-04-17:** Implemented rich text formatting (italics, bold, strikethrough, code blocks) and emoji support in chat, along with file transfers in

the UI chat.

- **[0.3.3] - 2025-04-21:** Switched from MongoDB to Firebase Database and added typing indicators.
- **[0.3.4] - 2024-04-23:** Fully migrated the backend to be hosted on Render.com.

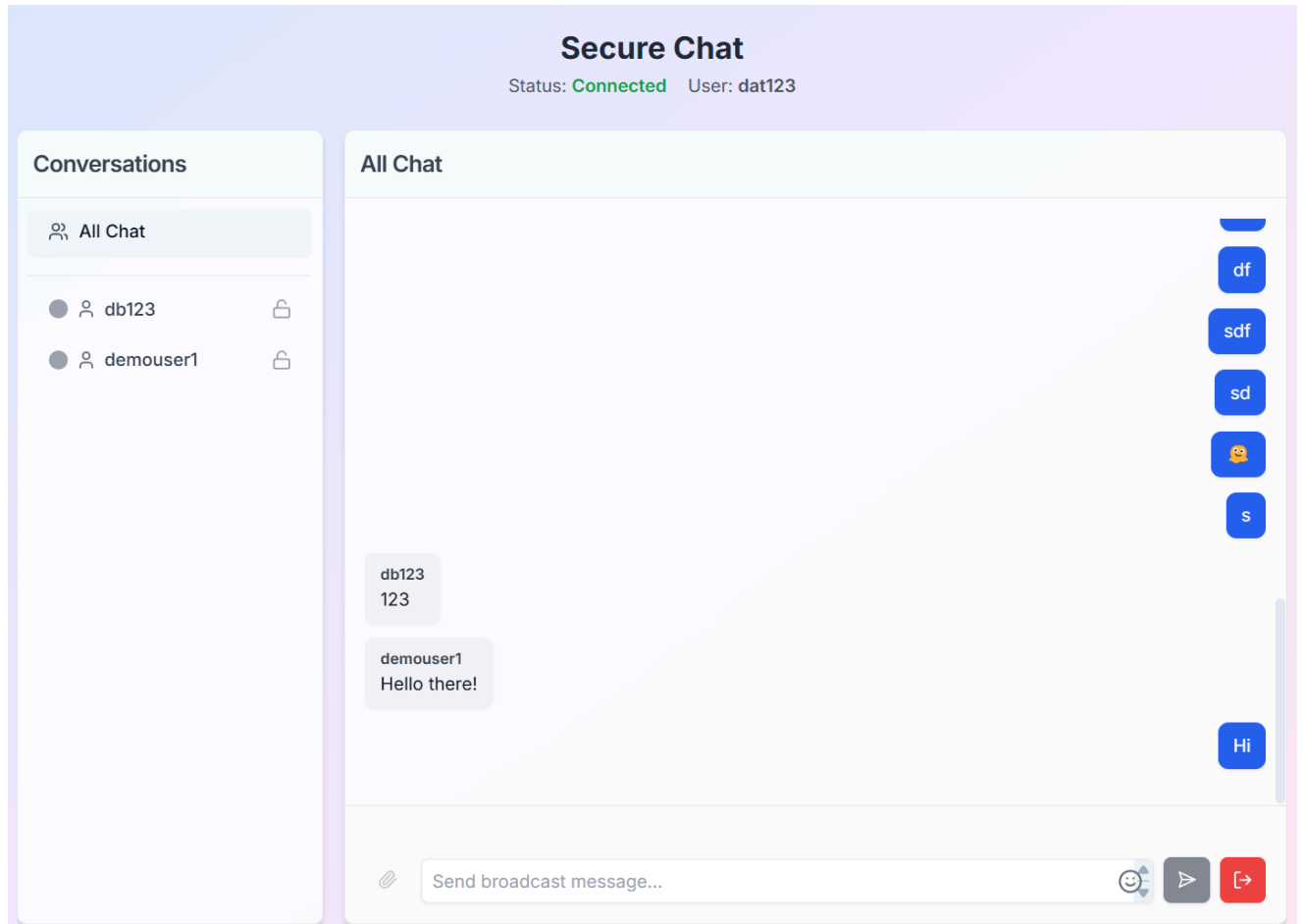
8. UI/UX

8.1. Homepage:

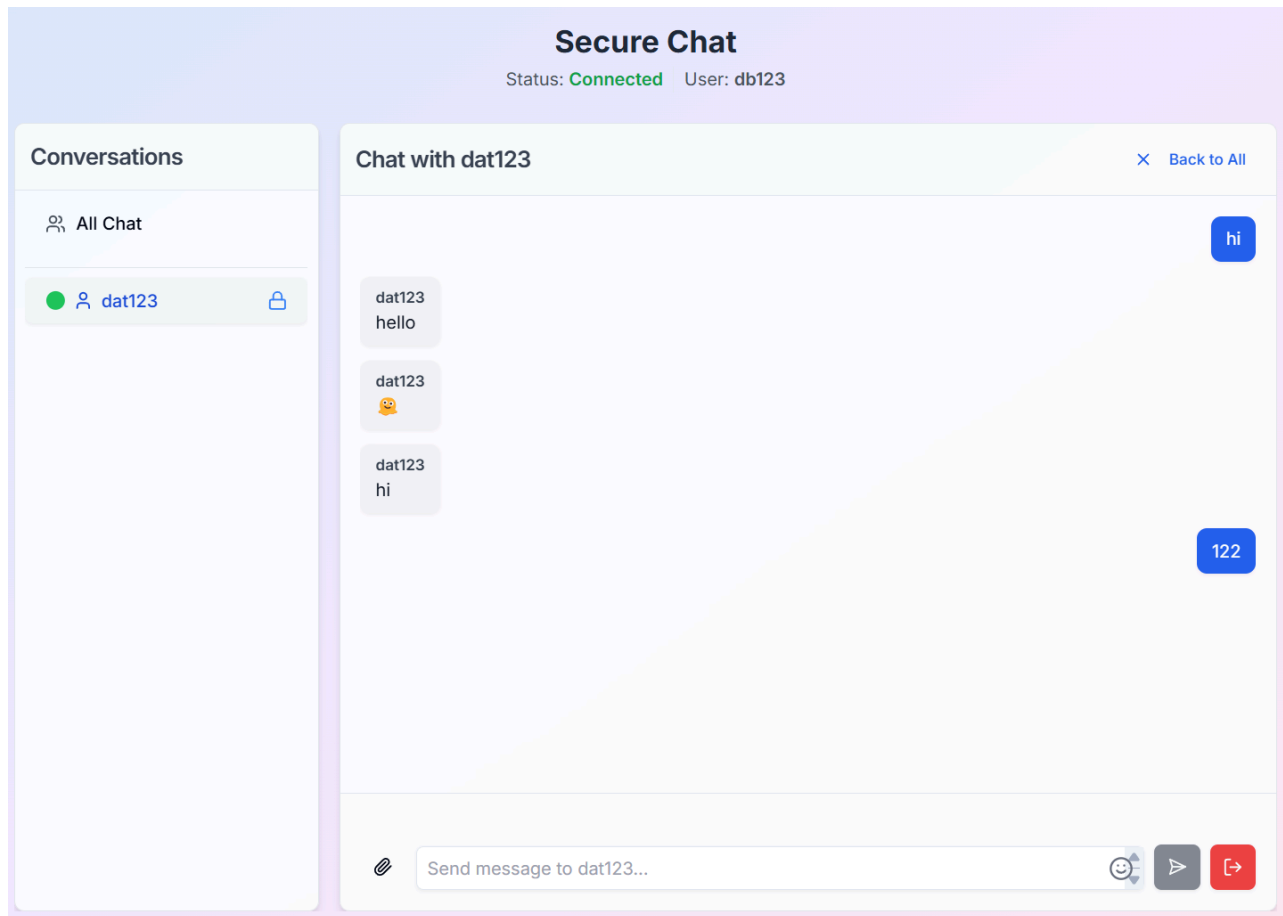


The image shows a web application interface for "Secure Chat". At the top, the title "Secure Chat" is displayed in bold, with a status indicator "Status: Connected" in green text below it. The interface is divided into two main sections. On the left, a sidebar titled "Conversations" contains a list of chat items, with "All Chat" selected and highlighted. Below the list, a message "Log in to see chats" is visible. The main area on the right is titled "All Chat" and contains a large message from the server: "[SERVER]: Welcome to Secure Chat! Please log in." At the bottom of the main area, there is a login section titled "Please Log In" which includes two input fields for "Username" and "Password", and a dark blue button labeled "Login / Register".

8.2. Public Chat Room:



8.2. Private Chat Room:



8.2. Send and Receive File in Private Chat Room:

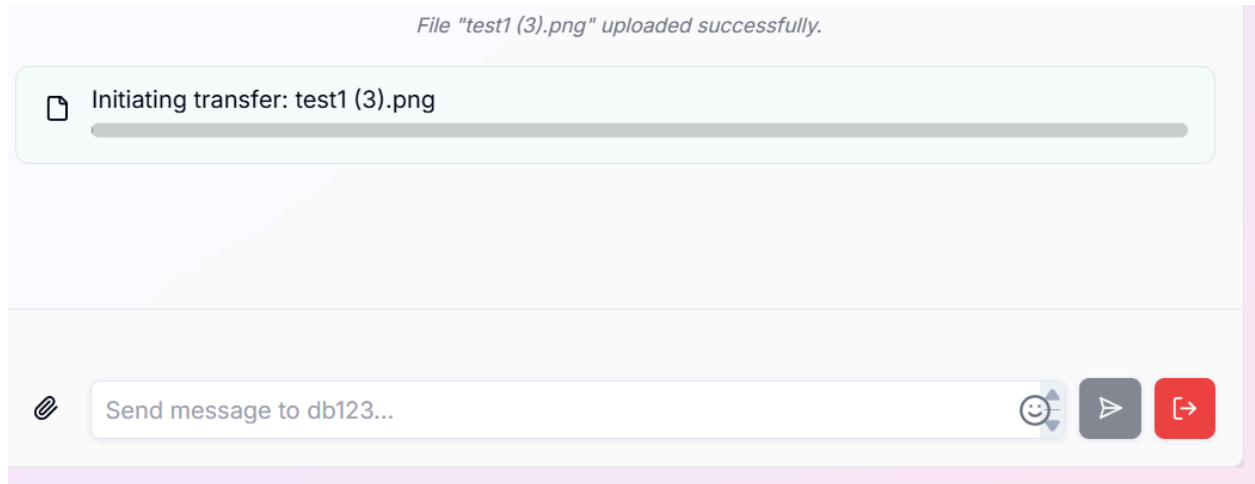


Figure: Sender is sending file.



Figure: Sender sent file successfully.

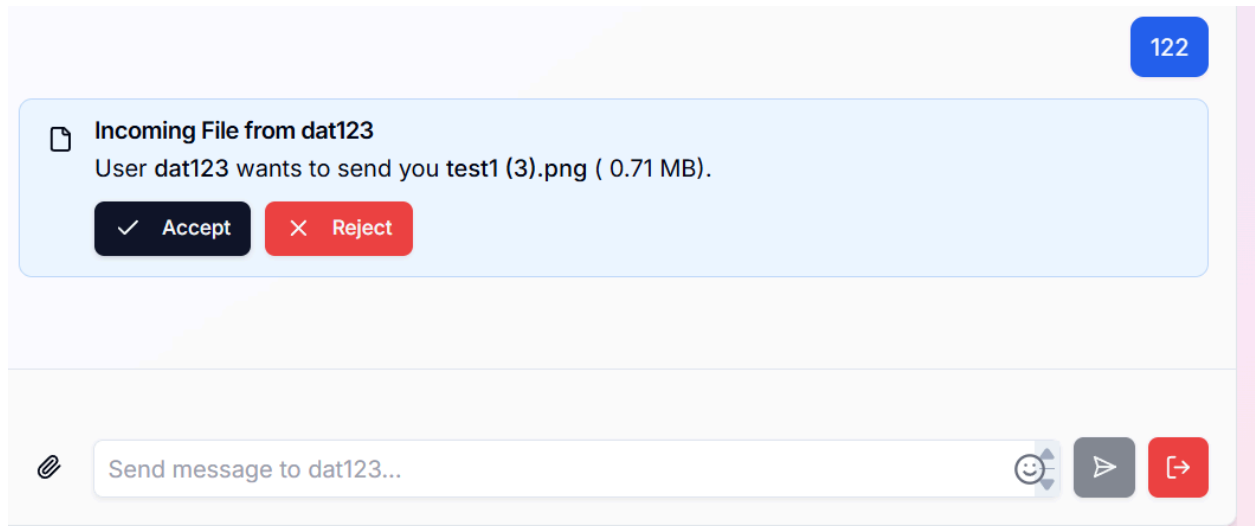


Figure: Receiver is getting a request from sender.

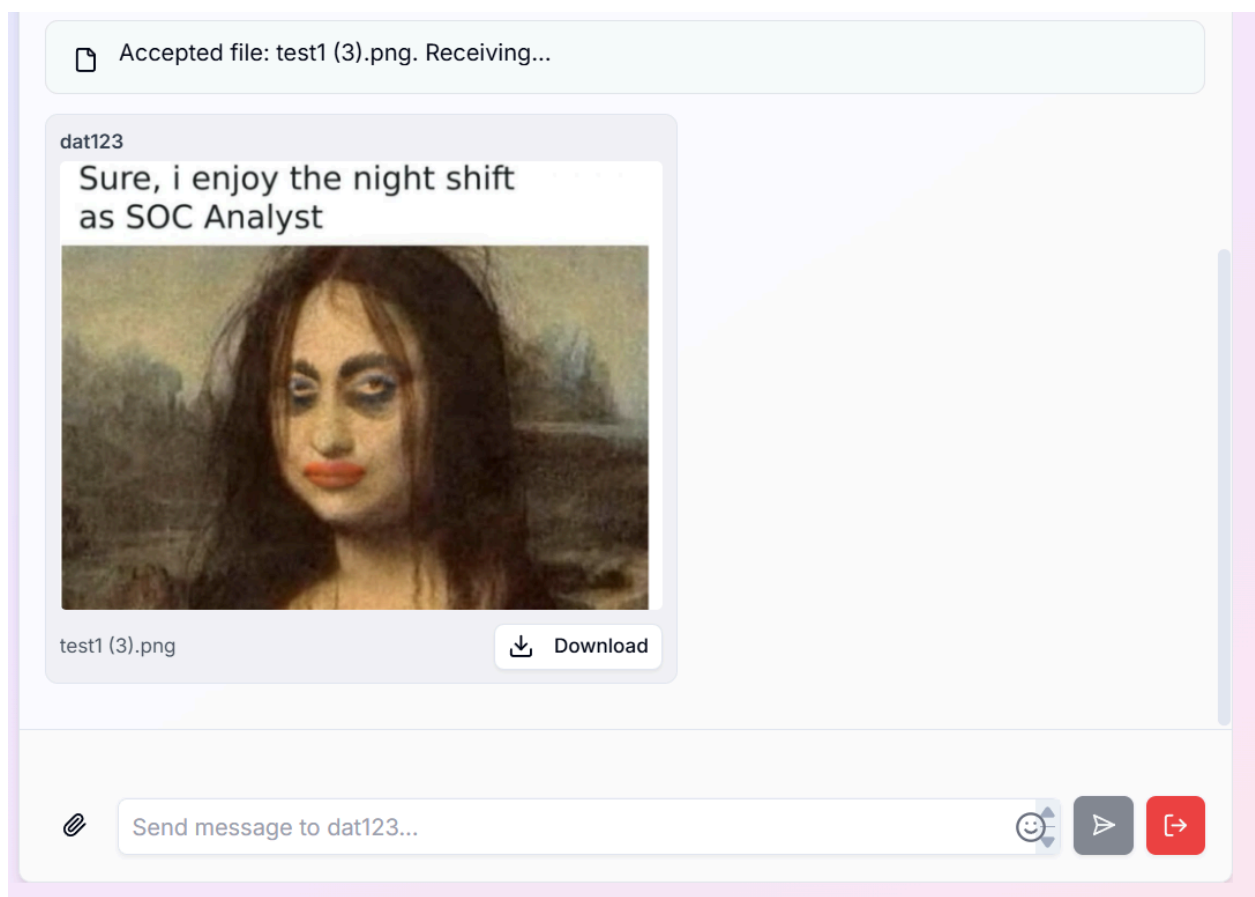


Figure: Receiver got the file successfully.

9. Reference:

AI helped to format some parts of this documentation.