¥apper - System Architecture Document

Team Number: 28

The University of Kansas

EECS 581 - Software Engineering II

Team Members: Nabeel Ahmad, Aniketh Aatipamula, Omar Mohammed,

Shero Baig, Humza Qureshi, Yaeesh Mukadam

Project Name: ¥apper

System Architecture Document

Document Revision History

Date	Version	Comment
10/24/2025	1.0	File is created, Outline is built for readability.
10/25/2025	1.1	Updated Synopsis, Architecture Description, and UML Diagrams

Table of Contents

System Architecture Document

Section 1 - Synopsis

This project is a real-time chat application that features WebSocket-based messaging. It emphasizes responsive interfaces, performance testing, and post-launch maintenance for scalability, reliability, and engagement.

Section 2 - Architecture Description

This project will utilize JavaScript and HTML/CSS for the frontend development. The role of the frontend is to handle user-interface (UI) rendering of elements such as: message list, input, participant list, and any accessibility settings that may exist. The frontend will also handle local client logic such as: scrolling and message input validations. WebSockets will also be used to send and receive messages between the clients and the server. From these aspects, the frontend will be able to handle any incoming events. The frontend will also handle the first initial interaction that users have with the project, with aspects like username and overall customization being a key area of interaction. Users will be able to set their own username, allowing for a more user-centered approach. Overall, the frontend exists to be a single page vanilla HTML/JS/CSS application with WebSockets to handle message interactivity.

From the backend point of view, Python will be used in this area of development. One responsibility of this area of the project will be to authenticate WebSocket upgrades, therefore allowing real-time communication between connections. The backend will also handle WebSocket connections and per-connection state (an example of this would be any current room(s) or user identifications). It will also handle moderation policies with backend algorithms to check for words containing bad language. There will also be a database to store messages, rooms, and users. And, when storing users, the backend will also have a hand in user name creation by determining if a user name has already been taken. If the user name has been taken, the user will be notified of such and they can then input a new name to be used. FastAPI and Socket.IO will also be utilized on the backend to implement REST/websockets respectively. Where REST API will handle username changes and other non-message/live update interactions. The storing of messages will also be handled by the backend to make sure message history can be retrieved and displayed as necessary.

The flow of messages will begin with the initial load using browser HTML GETs to then call for REST API for current rooms and initial history. The frontend will establish WebSocket to backend and authenticate using REST. The next step of message flow will be to send a message where the client will send their desired text over WebSocket. The backend will verify the user and begin applying filters for aspects like bad language. The backend will finally publish the event to the connected clients. For message retrieval, connected clients receive the message event and append it to the UI.

To observe the overall performance of the project, the logging of connection events, errors, and moderation actions can be used along with correlating identifications to determine any improvements or actions that need to be made for reconciliation. And with thorough testing

of the project, we aim to provide as clean of a sprint release as possible. This will be done using unit tests for core message handling events, where integration tests for WebSocket flow can be diagnosed and managed.

2.1 Architecture

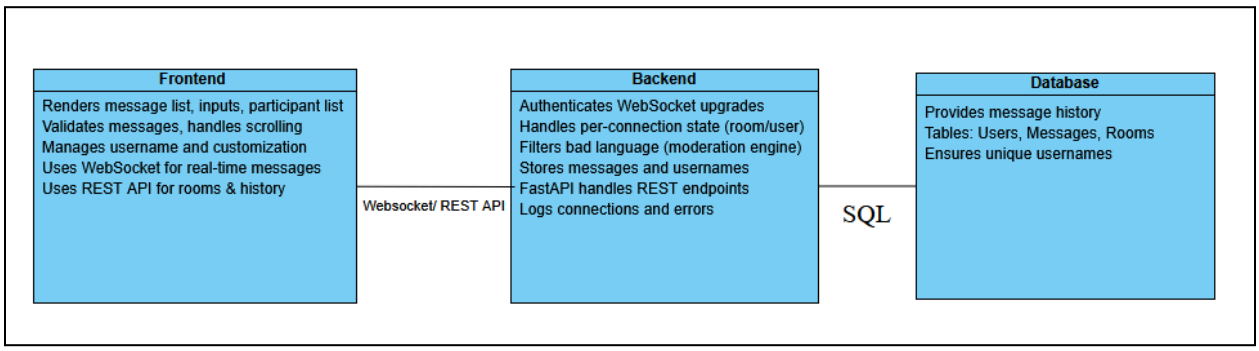

Figure 1: UML Diagram showcasing the architecture for the project

2.1 Data Flow

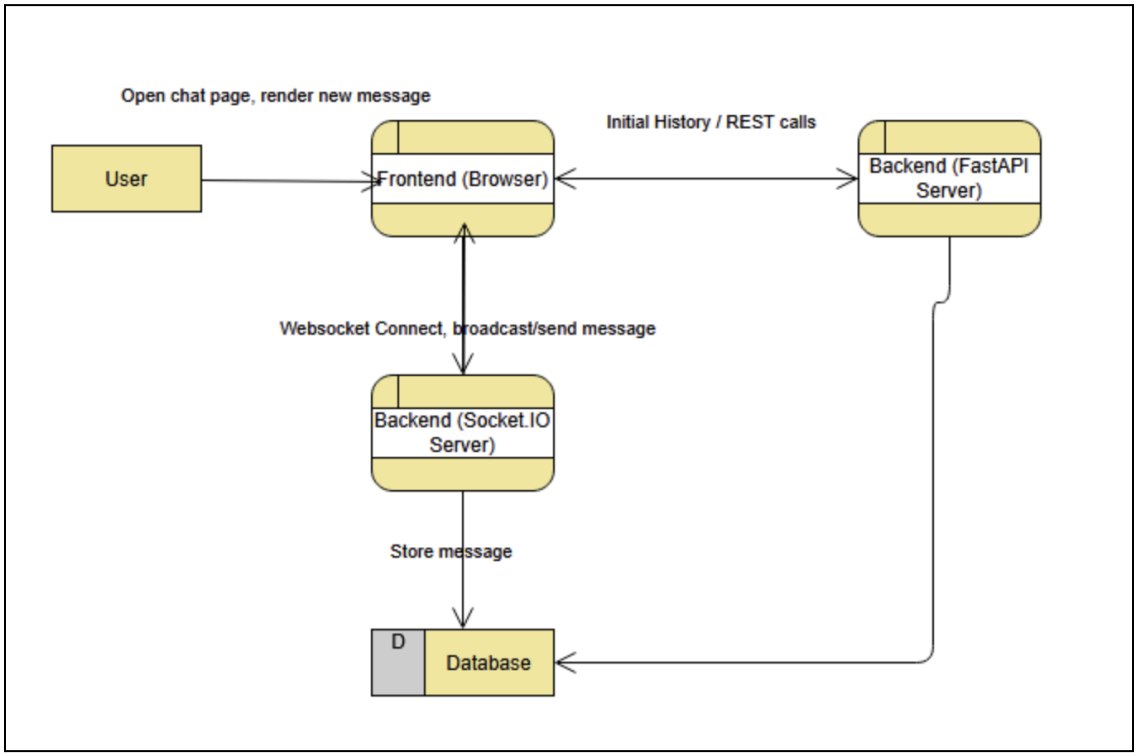

Figure 2: UML Diagram showcasing the Data Flow for the project

System Architecture Document

2.1 Components

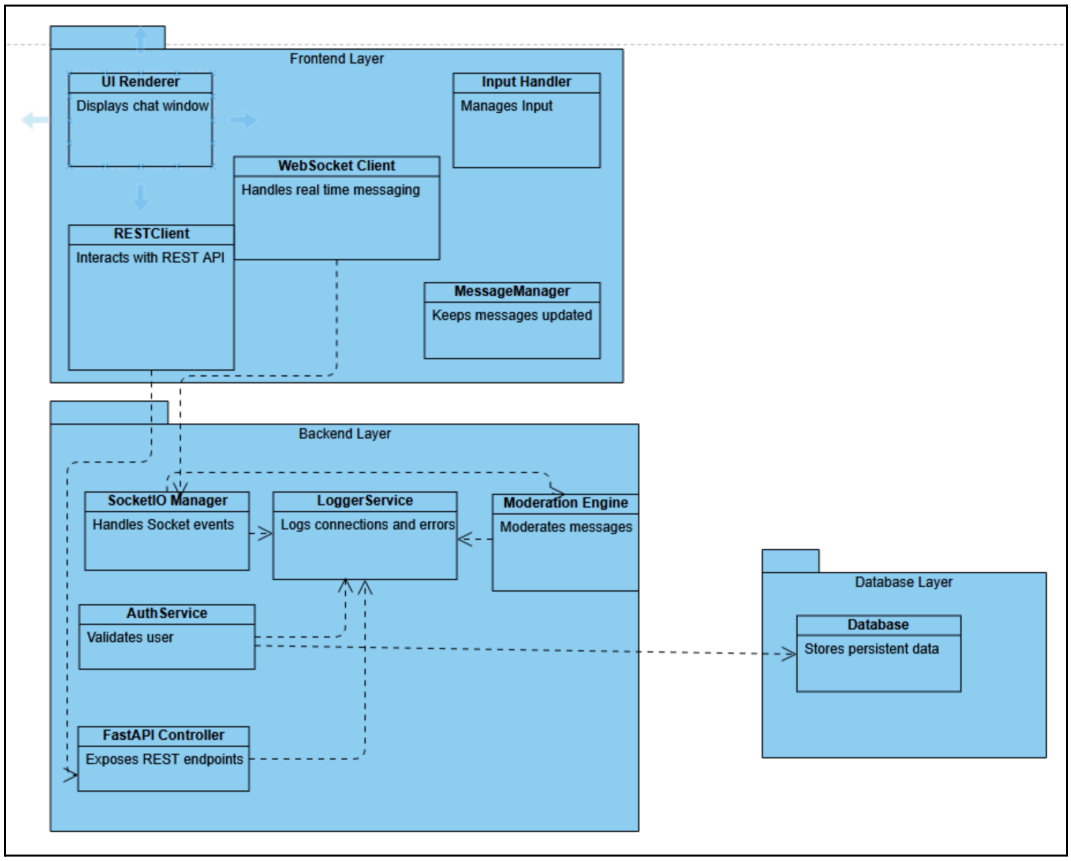

Figure 3: UML Diagram showcasing the components for the project