Omni

A Distributed Messaging System

Team 6 Proposal

Table of Contents

Team Profile

Micah Havens

Micah is a full-stack developer with five years of experience writing website frontends in HTML/CSS/JavaScript (as well as other various web frameworks) and writing backends in Node.JS. Micah also has experience using the MongoDB and Redis database systems, and has good all-around design and programming skills, as well as good organizational skills. Aside from the languages listed above, they are proficient in Java, C#, C/C++, and Lua.

Scott Devere

Scott has experience in writing C/C++, assembly, Python, and Matlab. They also have written or created system requirements specifications, system design documents, and design artifacts.

C.J. Commodore

C.J. is a hobbyist game developer, working with Unity and C# to create 2D games. He is an independent developer, designing, coding, and testing all aspects of his games. Working as an intern has given him experience being on a team and working in an Agile environment. This job gives him experience with languages C/C++ and Ada, as well as Linux systems and Atlassian tools, such as JIRA and BitBucket.

Adnaan Dasoo

Adnaan has experience with full stack development particularly in the web development space. Using languages and frameworks such as HTML, CSS, Java, JavaScript, Node.js, Python, and C++. He has developed web pages and applications for companies in the local DMV area.

Josh Martin

Josh has past coding experience in C/C++, Python, and with software like Matlab. They have experience with the setup and design of GUI interfaces. Josh is familiar with using software like GitHub, Jupyter, and IDEs like CLion and Pycharm. They are familiar with picking up new languages and learning how to effectively implement them.

Project Description

Overview

Omni is a distributed internet messaging platform inspired by IRC. It will consist primarily of a module called the Omni server. The Omni server can be linked to an existing web server, where it will then handle incoming messages from users on the website and distribute them to the other users. However, this alone would not make Omni very special. What will differentiate Omni from other chat systems will be its ability to “pair” with other Omni servers running on other websites. When two websites running Omni are “paired,” the users of one paired site will be able to chat across the web with the users of the other site. Paired Omni servers will be called “peers.” Some Omni servers will also be “hubs,” which will not directly accept chat messages from users, but will instead act as a directory of servers, all of whom will allow free communication amongst their peers connected to the same hub. All servers connected to the same hub will be able to talk to one another.

An Omni server will consist of “channels,” much like platforms such as IRC, Discord, and Slack. Channels may be either “shared” or “private.” A private channel cannot be reached from an Omni peer, but shared channels will be available to peers.

The core of the chat system is Omni. The frontend and web server that will be implemented in this project should be taken as an example - others running their own Omni servers may elect to write their own frontend and/or web server implementations, as long as their Omni server behaves as ours does. They may even choose to write their own implementation of the Omni server, and as long as it is compliant with the protocol this project will develop, it will be a valid Omni server that can participate in the Omni ecosystem, including pairing with other servers, accepting connections from Omni-compatible clients, et cetera.

The Omni messaging system’s target user base are those conscious about using free and open-source software for their everyday needs, those interested in decentralized services that are not blockchain-based, those who are privacy-conscious, and any combination thereof. Our project’s frontend will provide a good example implementation of Omni which will also be friendly for new users who are non-technical but who are interested in alternatives to popular messaging platforms, a demographic of people who are commonly told to use IRC, despite IRC’s lack of resemblance to many messaging platforms.

Technologies

The Omni server will be written in Node.js, as will the web server backend. The frontend will be written in plain HTML, CSS, and JavaScript. Real-time communication between the client and web server will be accomplished using Socket.IO. Communication between paired Omni servers will likely be accomplished using TCP sockets, which are implemented in the Net module within Node.

Functional Features

Below is a summary of the functional features of the system Team 6 will implement. It is not a complete list of requirements, but rather a general and cursory overview of the primary features each component of the system will have.

Frontend

- User shall be able to create an account with the website
- User shall be able to log in, and afterwards log out
- User shall be able to select a chat channel to view from a list of available channels
- User shall be able to send and receive messages within the currently selected channel
- On switching channels, the user shall be able to view any messages they missed in the channel they are switching to
- User shall be able to see a list of other users who are online in the channel they currently have selected

Web Server

- Shall host a secure account database, preferably using an existing library/database system that has been proven secure
- Shall pass messages received from the user to the Omni server module
- Shall pass messages from the Omni server to the correct users as specified by the Omni server

Omni Server

- Shall provide an API that:
    - Allows the web server to notify the Omni server of a new message from a connected user
    - Allows the Omni server to notify the web server to send a message to a specified user/set of users
    - Allows the web server to get the list of channels in the Omni server
    - Allows the web server to get the list of online users
    - Allows the web server to get the list of peers
    - Allows the web server to get previously sent messages inside a channel
    - Allows administration of the Omni server
- Administration features of the Omni server shall include:
    - Banning a user from interacting with the server
    - “Timing out” a user, which will disallow them from sending messages for a specified period of time
    - Creating a new channel
    - Deleting an existing channel
    - Wiping the message backlog of a channel
    - Sending a pair request to another Omni server

    - Accepting a pair request from another Omni server
    - Unpairing from a paired Omni server
- Shall forward messages intended for a peer server to the correct peer
- Shall allow incoming messages from peers and notify the web server to direct them to the appropriate users
- Shall notify peers when a channel is deleted

Plan of Work

Leadership, Roles, and Tasks

Every member of the team will contribute to as many parts of the system as possible; however, because some members have more experience in some areas, primary roles and tasks will be assigned as follows:

Member	Roles	Tasks
Micah Havens	Team Lead Design Lead Backend Programmer	Coordinate development, lay out API specifications, implement networking code (mainly implementing the Omni server functionality, but also some of the web server functionality)
Scott Devere	Backend Lead Backend Programmer	Aid in system design, assist in implementing database and account management
C.J. Commodore	Requirements Lead Frontend Programmer Graphic Design Test Design	Aid in frontend design: design mockups, webpage markup Test design: ensure tests correspond with requirements
Adnaan Dasoo	Frontend Lead Frontend Programmer Graphic Design	Design and implement frontend, assist in web server implementation, ensure an ergonomic and responsive frontend for user-facing chat features
Josh Martin	Integration Lead Backend Programmer Test Design	Aid in backend implementation: implement database management, implement user account management, ensure security of database Aid in test design: ensure continuous integration and ensure that tests include integration validation

Every member will be responsible for writing tests that validate their own code, but C.J. and Josh in particular will oversee test development and code validation. They will be responsible for filling in any gaps they see in test coverage. As stated, tasks and roles are flexible; some members will have more expertise than others in certain areas, but all members will gain experience with every part of the system.

Development Phases

Development will follow an incremental cycle of design and specification followed by development and testing. One phase will last approximately two weeks, and within one phase there will be two to four cycles, or increments. Before a cycle, team members will meet and decide what tasks to work on and what features to implement. During a cycle, team members will work either individually or in pairs on their tasks. Members that are writing code will write in a test-first or test-driven manner. After a cycle, team members will again meet to discuss progress, ensure all finished changes are well-integrated and pushed to the main branch on GitHub, and discuss any changes that may need to be made to the overall specification and design. The following are the deliverable product goals for each phase:

Weeks 1 and 2 - Prototype

The prototype will allow users to chat on one website. It will not allow pairing servers. There will be a list of channels to choose from, which all users will be able to chat in. There will be no account system. Users will be able to type and send chat messages using a simple chat pane.

The team will meet to discuss a range of design decisions, including a general layout of the website and a general design for the backend. This phase will require heavy programming effort from Micah and Adnaan, who will implement initial versions of both the frontend and backend, with design input from C.J., Scott, and Josh. Josh will ensure coordination between the frontend and backend, so that the browser communicates well with the web server, and so that the web server correctly uses the Omni server’s API to communicate with Omni.

Weeks 3 and 4 - Minimum Viable Product

The minimum viable product will include all features of the prototype, but it will allow pairing and unpairing of Omni servers, and it will allow users to chat across websites (i.e. across paired Omni servers). It will feature some form of user account creation/management. If the final product goal cannot be reached, the minimum viable product should be complete enough for submission.

Week 5 and Onwards - Final Product

The final product will fulfill all requirements specified in the System Requirements Specification, which includes all features of the Minimum Viable Product. It will allow channels to be designated private or public, and it will distinguish between users who are server administrators and normal users. The interface will allow for server administrators to create and delete channels.