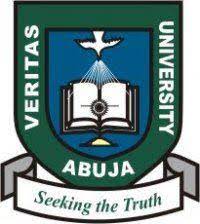
****

**NWANKWOALA KELECHI EMMANUEL**

**VUG/CSC/22/6834**

**CSC 302**

**DISTIBUTED COMPUTER SYSTEM**

**CAPSTONE PROJECT**

**Table of Contents:**

1. Introduction

2. Project Overview

* Main Application (main.py)
* HTML Templates
* CSS Stylesheets
* Client-side JavaScript

3. Installation Guide

4. Requirements

* User Requirements
* Functional Requirement
* NON-functional Requirement

5. Diagram

* USE CASE Diagram
* Flowchart Diagram

6. Code Explanation

* main.py
* HTML Templates
* CSS Stylesheet
* Client-side JavaScript

7. Conclusion

**Project Documentation: Real-time Chat App**

**Introduction**

The Real-time Chat App is a web application built using Flask, Socket.IO, HTML, CSS, and JavaScript. It allows users to join chat rooms, exchange messages in real-time, and create new chat rooms.

**Project Overview**

The project consists of several components:

1. Main Application (main.py)

The main.py file serves as the core of the Real-time Chat App. It contains the Flask application and handles various functionalities, including routing, socket communication, and message handling. Here's an overview of its key components:

* Flask Application: Initializes the Flask app and configures it with necessary settings.
* Routing: Defines routes for different URLs, such as the home page and chat room page.
* Socket Communication: Utilizes Flask-SocketIO to enable real-time communication between clients and the server.
* Message Handling: Implements functions to send, receive, and manage messages within chat rooms.

2. HTML Templates (base.html, home.html, room.html)

The HTML templates define the structure of the web pages served by the Flask application. These templates utilize Jinja templating to include dynamic content and facilitate code reuse. Here's a brief overview of each template:

* base.html: Acts as the base template that other templates extend. It includes common elements like the <head> section and provides a placeholder for content specific to each page.
* home.html: Represents the home page of the application where users can join existing chat rooms or create new ones. It includes input fields for name and room code.
* room.html: Represents the chat room page where users can exchange messages in real-time. It displays the chat history and provides an input field to send new messages.

3. CSS Stylesheets (style.css)

The CSS stylesheet (style.css) is responsible for styling the HTML templates, enhancing the visual appeal of the user interface. It provides rules for various elements such as buttons, input fields, and message boxes. Here's an overview of its key features:

* Global Styling: Defines global styles for HTML elements to ensure consistency across pages.
* Component Styling: Styles specific components such as buttons, input fields, and message boxes to create a cohesive and visually appealing UI.
* Responsive Design: Includes media queries to ensure that the application is well-presented on different screen sizes and devices.

4. Client-side JavaScript

The client-side JavaScript code embedded in home.html and room.html handles interactive functionalities and real-time communication between clients. Here's an overview of its functionalities:

* Real-time Communication: Utilizes Socket.IO to establish a WebSocket connection with the server and exchange messages in real-time.
* Message Handling: Implements functions to send and receive messages, update the chat interface with new messages, and handle user interactions such as joining and leaving chat rooms.
* Input Validation: Validates user inputs such as name and room code to ensure data integrity and prevent potential issues.

Overall, these components work together to create a seamless and interactive chat experience for users of the Real-time Chat App.

**Installation Guide**

To run the Real-time Chat App locally on your machine, follow these steps:

1. **Clone the Repository**
   * Clone the repository to your local machine using Git:

git clone <repository\_url>

* + Alternatively, you can download the repository as a ZIP file and extract it to your desired location.

1. **Install Dependencies**
   * Navigate to the project directory in your terminal.
   * Use pip to install Flask and Flask-SocketIO:

pip install Flask Flask-SocketIO

1. **Run the Flask Application**
   * Start the Flask application by running the **main.py** file:

python main.py

1. **Access the Chat Application**
   * Once the Flask application is running, open your web browser.
   * In the address bar, type **http://localhost:5000** and press Enter.
   * This will take you to the home page of the Real-time Chat App, where you can join existing chat rooms or create new ones.
2. **Start Chatting**
   * Follow the prompts on the home page to enter your name and choose a chat room.
   * Once inside a chat room, you can exchange messages with other users in real-time.

**Requirements**

**User Requirements:**

1. **User Authentication**: Users should be able to enter their name to identify themselves in chat rooms.
2. **Join/Create Chat Rooms**: Users should have the option to join existing chat rooms by entering a room code or create new chat rooms.
3. **Real-time Messaging**: Users should be able to exchange messages with other participants in real-time.
4. **Message History**: Chat rooms should display message history to users, allowing them to view previous conversations.
5. **User Presence**: Users should be able to see who else is currently present in the chat room.
6. **Leave Chat Rooms**: Users should have the ability to leave chat rooms when they no longer wish to participate.
7. **User Interface**: The application should have an intuitive and visually appealing user interface for ease of use.

**Functional Requirements:**

1. **Routing**: The application should have routes for the home page and individual chat rooms.
2. **Socket Communication**: Utilize WebSocket connections to enable real-time messaging between clients and the server.
3. **Message Handling**: Implement functionality to send, receive, and display messages within chat rooms.
4. **Room Creation**: Allow users to create new chat rooms with unique room codes.
5. **Joining Rooms**: Users should be able to join existing chat rooms by entering the correct room code.
6. **User Authentication**: Ensure that users provide a name to identify themselves within chat rooms.
7. **Message History**: Display previous messages in chat rooms to provide context for ongoing conversations.
8. **User Presence**: Show a list of users currently present in each chat room to provide visibility into who is online.
9. **Leave Rooms**: Provide an option for users to leave chat rooms when they no longer wish to participate.

**Non-functional Requirements:**

1. **Performance**: The application should be responsive and able to handle multiple concurrent users without significant latency.
2. **Scalability**: The architecture should support horizontal scaling to accommodate a growing number of users and chat rooms.
3. **Reliability**: Ensure that the application is robust and able to recover gracefully from errors or unexpected events.
4. **Security**: Implement measures to protect user data and prevent unauthorized access to chat rooms.
5. **Accessibility**: Design the user interface to be accessible to users with disabilities, following best practices for web accessibility.
6. **Compatibility**: Ensure that the application is compatible with a wide range of web browsers and devices to reach a broad user base.
7. **Usability**: Design the user interface to be intuitive and easy to navigate, with clear instructions for users on how to use the application.
8. **Maintainability**: Write clean, well-documented code to facilitate future maintenance and updates to the application.
9. **Scalability**: Design the application architecture to be scalable, allowing it to handle increased load as user traffic grows over time.

Performance Requirements:

1. **Real-time Messaging**: Messages should be delivered to recipients within milliseconds of being sent.
2. **Message History**: The application should load message history quickly, even for chat rooms with a large number of messages.
3. **Scalability**: The application should be able to handle a high volume of concurrent users without experiencing performance degradation.
4. **Resource Usage**: The application should use system resources efficiently to minimize CPU, memory, and network usage.

**Application Domain Requirements:**

1. **Messaging Platform**: The application domain revolves around providing a messaging platform where users can engage in real-time communication within chat rooms.
2. **User Interaction**: Users should be able to interact with each other through text-based messages, facilitating discussions, sharing information, and collaborating on various topics.
3. **Room Management**: The system should allow users to create new chat rooms, join existing ones, and manage their participation in multiple rooms simultaneously.
4. **Authentication and Identity**: Users need to authenticate themselves within the system by providing a unique username or identity to distinguish themselves from others.
5. **Message Persistence**: The application domain includes the storage and retrieval of message history, ensuring that users can access past conversations within chat rooms.
6. **Presence Awareness**: The system should provide visibility into the presence of other users within chat rooms, indicating who is currently online and active.
7. **Real-time Communication**: The primary focus of the application domain is to enable real-time communication, ensuring that messages are delivered instantly and participants can engage in live conversations.

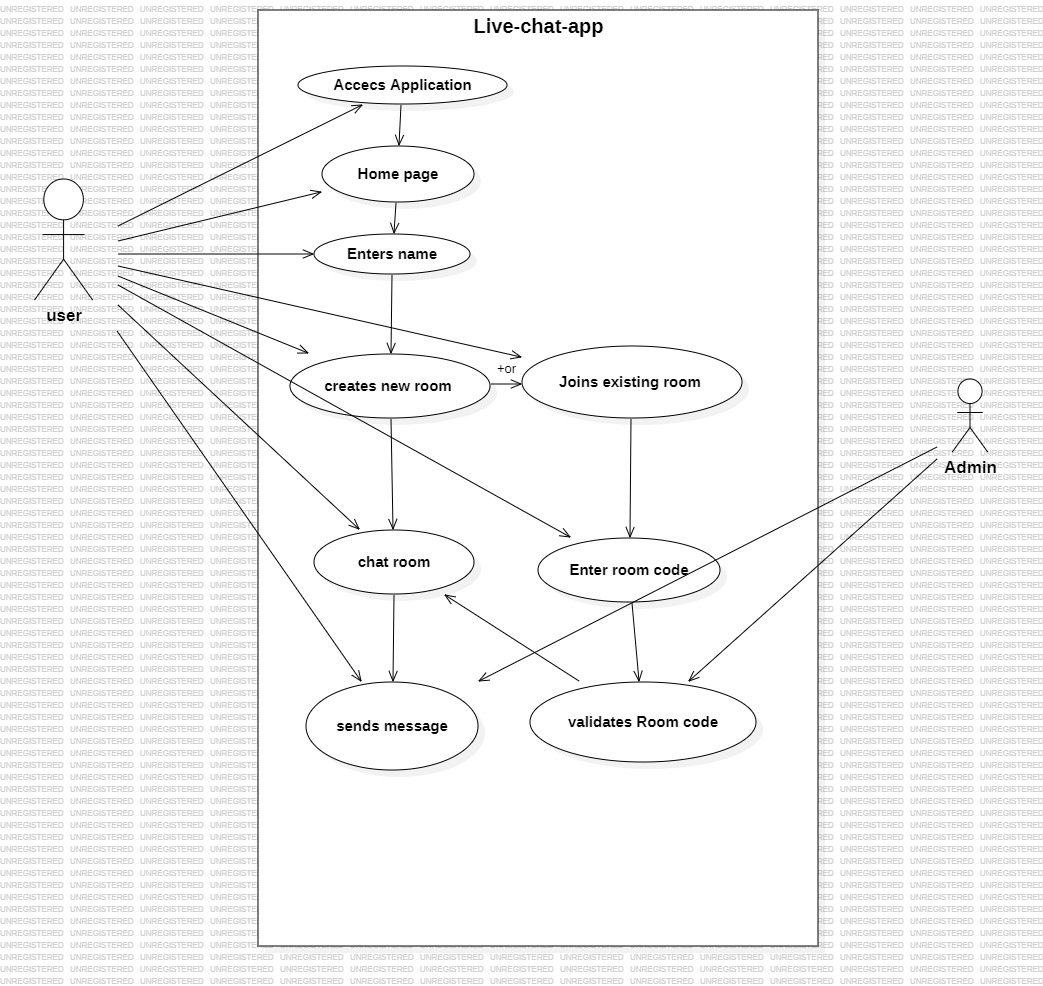
**Problem Domain Requirements:**

1. **Scalability**: The system should be scalable to accommodate a growing user base and increasing message traffic over time.
2. **Performance**: It is crucial for the system to perform efficiently, delivering messages promptly and maintaining responsiveness even under heavy load.
3. **Reliability**: The application must be reliable, ensuring minimal downtime and the ability to recover gracefully from failures or disruptions.
4. **Security**: Protecting user data and ensuring the privacy of conversations is paramount. The system should implement robust security measures to prevent unauthorized access and data breaches.
5. **Usability**: The user interface should be intuitive and easy to navigate, catering to users of varying technical backgrounds and abilities.
6. **Accessibility**: The application should be accessible to users with disabilities, complying with accessibility standards and providing features such as screen reader compatibility and keyboard navigation.
7. **Maintainability**: The system should be maintainable, with clean and well-documented code that facilitates ongoing development, bug fixes, and updates.
8. **Compatibility**: Ensure compatibility with a wide range of devices and web browsers to maximize accessibility and reach a broader audience.

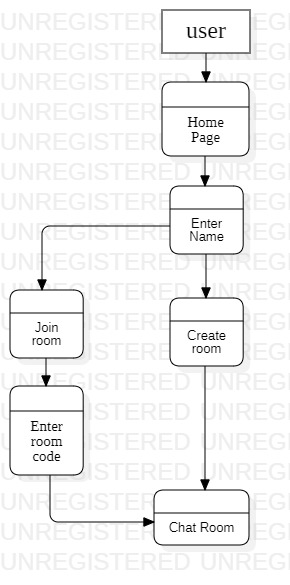
**System Requirements:**

1. **Hardware Requirements**: The system should run on standard hardware configurations, including servers for hosting the application and clients accessing it through web browsers.
2. **Software Requirements**: The application should be compatible with common web development technologies such as Python (Flask), JavaScript (Socket.IO), HTML, and CSS.
3. **Dependencies**: Ensure that all required dependencies, libraries, and frameworks are installed and properly configured on the server and client environments.
4. **Network Requirements**: The system requires a stable internet connection to facilitate real-time communication between clients and the server.
5. **Storage Requirements**: The system should include storage mechanisms for persisting message history and user data, with adequate capacity and performance for efficient data retrieval.
6. **Performance Requirements**: Define performance metrics such as message delivery time, response times, and system throughput to ensure optimal performance under normal and peak load conditions.
7. **Security Requirements**: Implement security measures such as encryption, authentication, and authorization to protect user data and prevent unauthorized access.
8. **Scalability Requirements**: Design the system architecture to scale horizontally or vertically to handle increased user traffic and message volume as the application grows.
9. **Monitoring and Logging**: Implement monitoring and logging mechanisms to track system performance, identify issues, and troubleshoot problems as they arise.
10. **Backup and Recovery**: Establish backup and recovery procedures to safeguard data integrity and ensure quick recovery in the event of data loss or system failure.

These requirements ensure that the Real-time Chat App meets the needs of its users while providing a reliable, responsive, and secure messaging platform.

****

**USE CASE DIAGRAM**

****

**FLOWCHART DIAGRAM**

**Code Explanation**

**Main Application (main.py)**

**Overview:**

The **main.py** file serves as the primary application logic for the Real-time Chat App. It handles routing, socket communication, and message handling between clients and the server.

**Functionality:**

1. **Routing**: Defines routes for the home page (**/**) and room page (**/room**), ensuring proper navigation within the application.
2. **Session Management**: Manages user sessions to store information such as username, room code, and message history.
3. **Socket Communication**: Utilizes Flask-SocketIO to establish WebSocket connections and enable real-time communication between clients and the server.
4. **Message Handling**: Implements event handlers for sending and receiving messages within chat rooms, updating message history, and managing user presence.
5. **Room Management**: Supports creating new chat rooms with unique codes and joining existing rooms by entering room codes.
6. **Error Handling**: Validates user inputs and provides error messages for missing or invalid data.

**Key Components:**

* **Flask Application**: Initializes the Flask app and configures it with session management and SocketIO support.
* **Routes**: Includes routes for the home page and room page, along with corresponding request handling functions.
* **SocketIO Event Handlers**: Implements handlers for **message**, **connect**, and **disconnect** events to manage real-time communication.
* **Session Storage**: Stores session data such as username, room code, and message history using Flask session management.

**2. Backup Server (backup\_server.py)**

**Overview:**

The **backup\_server.py** file serves as a backup server for the Real-time Chat App, providing redundancy and failover capabilities in case the main server (**main.py**) goes down.

**Functionality:**

1. **Server Initialization**: Initializes the Flask app and configures it with session management and SocketIO support.
2. **Routing**: Defines routes for the home page (**/**) and room page (**/room**) similar to the main server.
3. **Session Management**: Manages user sessions to store information such as username, room code, and message history, ensuring seamless transition in case of failover.
4. **Socket Communication**: Utilizes Flask-SocketIO to establish WebSocket connections and enable real-time communication between clients and the backup server.
5. **Message Handling**: Implements event handlers for sending and receiving messages within chat rooms, updating message history, and managing user presence, mirroring the functionality of the main server.

**Key Components:**

* **Flask Application**: Initializes the Flask app and configures it with session management and SocketIO support, similar to the main server.
* **Routes**: Includes routes for the home page and room page, along with corresponding request handling functions, identical to the main server.
* **SocketIO Event Handlers**: Implements handlers for **message**, **connect**, and **disconnect** events to manage real-time communication, mirroring the functionality of the main server.
* **Session Storage**: Stores session data such as username, room code, and message history using Flask session management, ensuring consistency between the main server and backup server.

**Integration with Prometheus Monitoring:**

**Overview:**

Both the main server (**main.py**) and backup server (**backup\_server.py**) integrate with Prometheus monitoring to track various metrics related to server performance and resource utilization.

**Functionality:**

1. **Metrics Collection**: Captures metrics such as CPU usage, memory usage, disk usage, and network traffic using Prometheus client libraries.
2. **Exposure via HTTP Endpoint**: Exposes metrics through an HTTP endpoint (**/metrics**), allowing Prometheus to scrape and collect data at regular intervals.
3. **Monitoring Server Performance**: Monitors server performance, identifies bottlenecks, and ensures optimal resource utilization for handling chat traffic.
4. **Alerting and Alertmanager Integration**: Implements alerting rules and integrates with Alertmanager to send notifications in case of performance degradation or system failures.

**Key Components:**

* **Prometheus Metrics**: Includes metrics for CPU usage, memory usage, disk usage, and network traffic, defined using Prometheus client libraries.
* **HTTP Endpoint**: Exposes metrics through an HTTP endpoint (**/metrics**), allowing Prometheus to collect data for monitoring.
* **Monitoring and Alerting Rules**: Defines monitoring rules and alerting thresholds to detect anomalies and trigger alerts via Alertmanager.
* **Integration with Alertmanager**: Integrates with Alertmanager to send alerts via various channels (e.g., email, Slack) in response to critical events or performance issues.

3. HTML Templates (base.html, home.html, room.html)

* Description:
  + base.html:
    - Serves as the base template that other templates extend.
    - Contains common HTML elements and structure shared across multiple pages.
  + home.html:
    - Represents the template for the home page where users can join existing chat rooms or create new ones.
    - Includes input fields for users to enter their name and room code.
  + room.html:
    - Represents the template for the chat room page where users can exchange messages in real-time.
    - Displays the chat messages, input field for typing messages, and other room-related information.
* Purpose:
  + Provides a structured layout and visual representation of the web pages presented to users.
  + Ensures consistency in the appearance and behavior of different pages across the application.
  + Facilitates dynamic content rendering using Jinja templating, allowing for the insertion of dynamic data into HTML elements.

4. **CSS Stylesheet (style.css)**

* Description:
  + Defines the visual appearance and styling rules for HTML elements across all pages of the application.
  + Specifies properties such as colors, fonts, margins, paddings, and positioning to create a visually appealing user interface.
  + Includes responsive design elements to ensure optimal display on various screen sizes and devices.
* Purpose:
  + Enhances the user experience by providing a visually appealing and intuitive interface.
  + Improves readability, accessibility, and usability and usability of the web pages.
  + Maintains consistency in the visual presentation of elements, branding, and styling throughout the application.

**5. Client-side JavaScript**

* **Description:**
  + Handles client-side interactions and events triggered by user actions, such as sending and receiving messages, joining and leaving chat rooms.
  + Establishes WebSocket connections with the server using Socket.IO library to enable real-time communication.
  + Dynamically updates the user interface based on received data and user inputs without the need for page reloads.
* **Purpose:**
  + Enables real-time interaction between users within the chat application.
  + Facilitates seamless communication between the client and server, ensuring quick and efficient data exchange.
  + Enhances user engagement and i

**Conclusion**

The Real-time Chat App demonstrates the integration of Flask and Socket.IO to create a real-time messaging application. Users can easily join existing chat rooms or create new ones, allowing for seamless communication in real-time.