System Design Document

Name : Dhanushmanth savaturi

Institute : IIT Bhubaneswar

Department : Computer Science and Engineering .(B-Tech)

- Project setup and dependencies installation steps has been mentioned in the project folder README file in this Github repo link .
- Also few screenshots of the prototype has also been attached int the folder .

Small request :

- I Developed a real time chat application with group chat functionality along with user registration and authentication .

- I Couldn't complete the one -one messaging service due to some errors .Kindly consider this ..
  - ➔ When the shortlisted candidates are announced , I contacted my Student Internship coordinator that I'm willing to change to Data Scientist role if the management allows me after checking my profile .
  - ➔ So , he asked the HR about this and I was waiting for the HR reply for the first 3 days as the projects were different for these roles .After no response from the HR for 3 days ,then I started the project . Due to this I lost 3 days of valuable time comparatively .Although I used that time to revise my tech stack but I couldn't start the project as early as others .
  - ➔ Kindly consider this and I hope that whatever features I have developed they are working fine except one – one chat functionality that I couldn't resolve the issue . So , I needed to remove it and if you also feel the same that all other features good then please consider giving me a chance to present my skills and expertise in the interview and a chance to solve the issue and helping to complete the project.
    Thanks for the time and patience .
    > The documentation can be found from the next page .

APP:

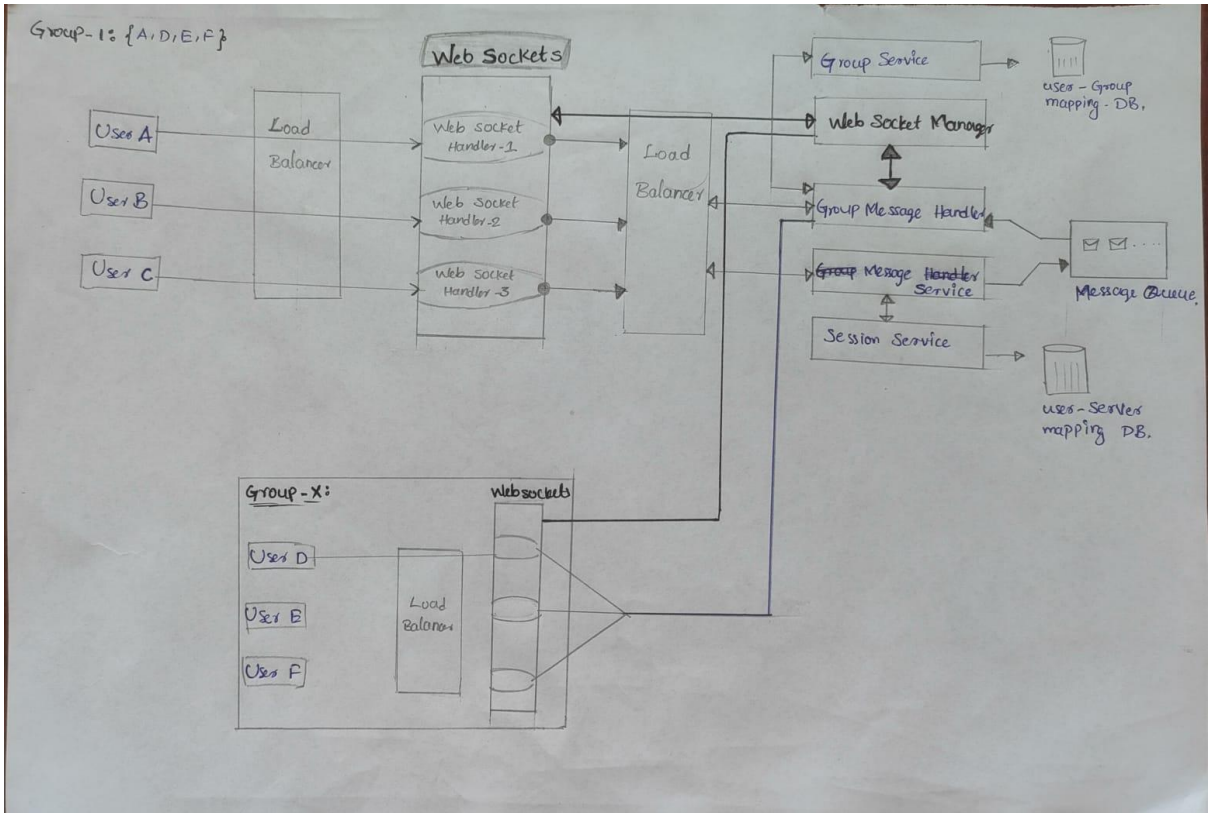
Wechat – A web messaging service prototype build with Django framework , channels , websockets , REST APIs with user registration and authentication .

- Developed a real time chat application with group chat functionality along with user registration and authentication .

System Architecture :

High-Level Architecture Diagram:

- This is a hand drawn diagram , I will update it with a screenshot of diagram drawn on a online board system .



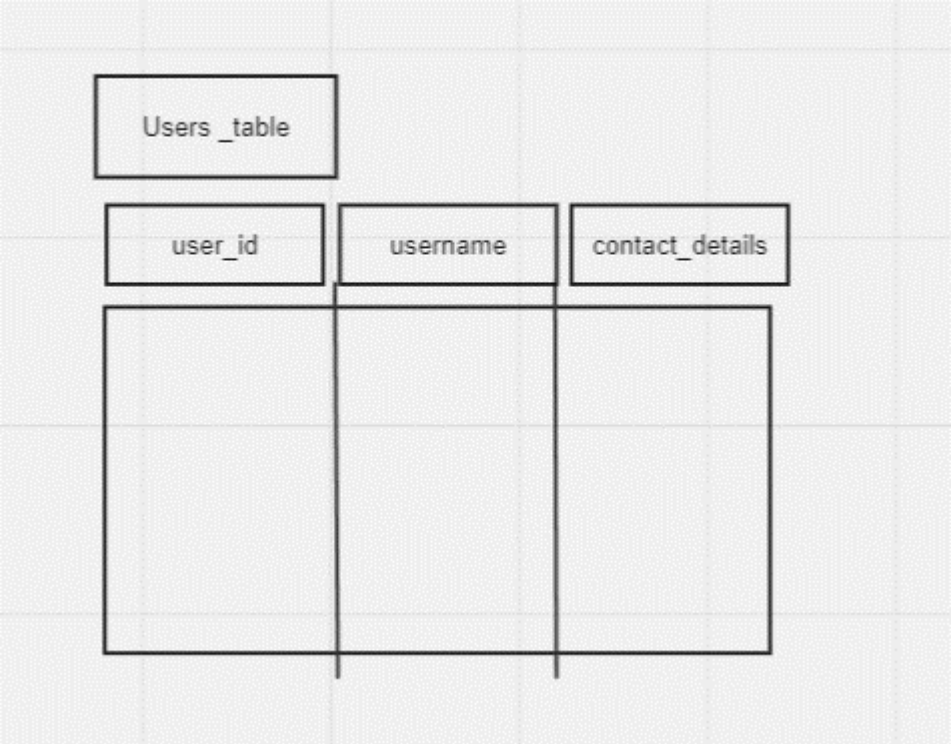
Architecture overview:

- Lets assume when User A wants to send a message to User B ,then first user A establishes a persistent connection with Messaging service via websocket protocol .
- Then User A sends a message request to Messaging service with a ID of User B .
- Websocket protocol comes in handy with the feature that the websocket protocol server can respond without any client request to be made .
- This is unlike a traditional HTTP protocol where the client needs to send a request every time when it requires some response .
- Now , the messaging service identifies User B via session service to deliver the message .
- The sessions service works in a way that whenever a user connects to the messaging service , it will tell the session service in which server that particular user has established the connection which is stored in a database .
- And later this data can be used to deliver messages to the other end .

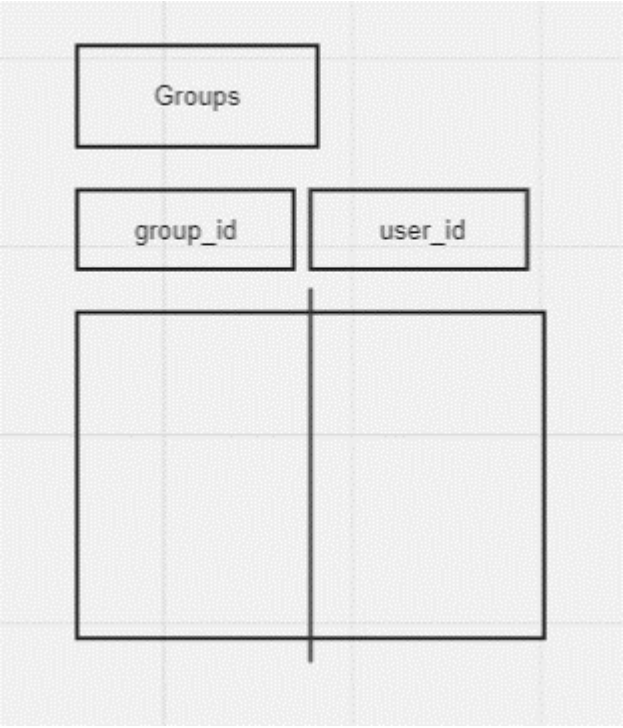
- When a user wants to send a message to a particular group . Websocket gets in touch with Message handler .
- Now , this message service will store the message in Queue / kafka message queue . And Automatation such as which user is sending message to which group basically Message service will act as kafka producer .
- Whenever message service posts a message to kafka message queue that a particular user is sending a message to a group . Group message handler will query Group service to get the list of all users which are in that particular group id .It gives that data from user – group mapping database .
- After that , when the Group message handler gets the list of all the users , this handler now needs the data of the respective list of machines those users are connected to which it will get from the websocket manager .
- Once it gets list of machines the Message handler will send message to individual machines by contacting the respective websocket handler .
- Where websocket is a light weight server which keeps an open bi-directional connection with all users .

Database Schema :

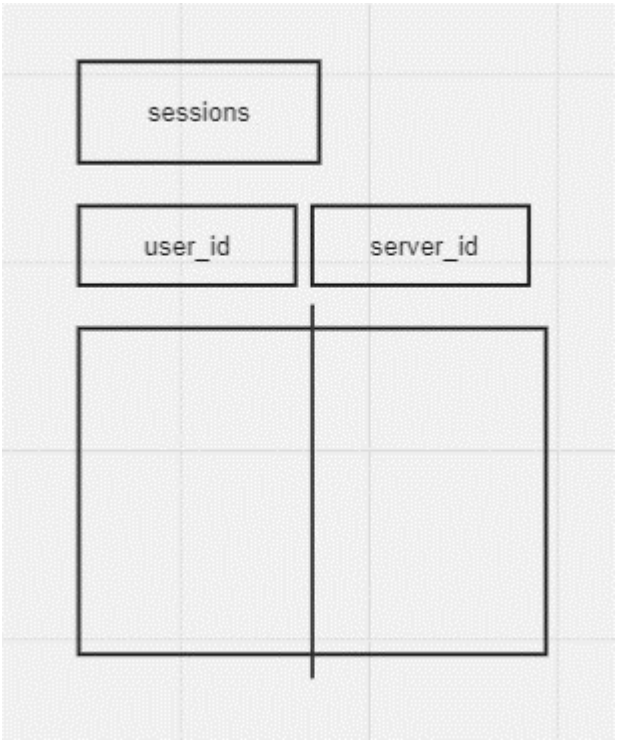
- Users table – which contains user_id , username and other contact details .



- Groups table :



- Sessions table : contains user_id and server_id which can be used to send the message to that respective server_id while delivering the message .



Components overview :

Django (Web Framework):

- **Description:** Django is a high-level Python web framework that provides a solid foundation for developing web applications. It follows the Model-View-Controller (MVC) pattern and promotes rapid development through its "batteries-included" philosophy.
- **Role:** Django serves as the primary web server, handling HTTP/HTTPS requests from clients. It manages routing, views, and integrates seamlessly with Django Channels for real-time communication.
- **Key Features:**
    - URL routing: Maps URLs to view functions.
    - ORM (Object-Relational Mapping): Simplifies database operations using Python classes.
    - User authentication: Provides built-in user management and authentication.
    - Middleware: Allows custom request/response processing.
    - Templating engine: Supports HTML templates for generating dynamic content.

Django Channels (Asynchronous Support):

- **Description:** Django Channels extends Django to provide support for asynchronous operations, including Websockets. It allows handling real-time events and long-lived connections efficiently.
- **Role:** Django Channels is responsible for managing Websocket connections and asynchronous communication in the application.
- **Key Features:**
    - ASGI (Asynchronous Server Gateway Interface): Allows handling asynchronous requests.
    - Channels Layer: Provides a communication layer for handling events and messages.
    - WebSocket consumers: Implement logic for handling Websocket events.
    - Routing: Routes events to appropriate consumers based on their type.

Websockets (Real-Time Communication):

- **Description:** Websockets are a protocol that enables bidirectional, real-time communication between the server and clients over a single, long-lived connection.
- **Role:** Websockets facilitate real-time chat functionality, allowing users to send and receive messages instantly without the need for frequent polling.
- **Key Features:**
    - Full-duplex communication: Both the server and client can send data independently.
    - Low latency: Real-time updates with minimal delay.

- Event-driven: Messages are sent and received as events, making it suitable for chat applications.
- Connection handling: Handles WebSocket connections efficiently, even for a large number of concurrent users.

REST APIs (Client-Server Communication):

- **Description:** REST (Representational State Transfer) is an architectural style for designing networked applications. RESTful APIs allow clients to interact with the server using HTTP requests (GET, POST, PUT, DELETE).
- **Role:** REST APIs provide an interface for client-server communication, enabling actions such as user registration, login, retrieving chat history, and managing chat rooms.
- **Key Features:**
  - Resource-based: Resources (e.g., users, messages) are represented as URLs.
  - Stateless: Each request from a client to the server must contain all the information needed to understand and fulfill the request.
  - CRUD operations: Supports Create, Read, Update, and Delete operations on resources.
  - Authentication: Typically uses token-based or session-based authentication.

> These components work together and run real-time web chat application. Django provides the foundation for handling HTTP requests and user management, while Django Channels and Websockets add real-time capabilities. REST APIs offer a structured way for clients to interact with the server, completing the architecture for chat app.

THE END