Chat Engine Architecture Summary

Introduction:

Chat Engine is a serverless chat application that enables real-time communication between users. The application is based on a microservices-like design and is modular with clear separation of concerns between components responsible for different aspects of the system.

System Components:

Frontend:

Static Resources:

- css: Contains the stylesheet files that style the user interface.
- js: Contains JavaScript files that handle frontend logic, such as establishing WebSocket connections and processing chat messages.

Templates:

- chat.html: The Thymeleaf template for the chat room interface.
- login.html: The Thymeleaf template for the login page.
- register.html: The Thymeleaf template for the user registration page.

Backend:

- ChatingApplication: The main Spring Boot application class responsible for launching the application.

Chat Package:

- ChatController: Manages WebSocket sessions and handles chat message sending and receiving operations.
- ChatMessage: The entity representing a chat message, which includes details like the sender, content, and timestamp.
- ChatMessageRepository: Spring Data JPA repository for CRUD operations on chat messages.
- MessageType: An enumeration defining the types of messages that can be exchanged, such as CHAT, JOIN, and LEAVE.

Config Package:

- ChatEventListener: Listens to WebSocket events, such as connect and disconnect, and handles broadcasting of messages to connected clients.
- SpringSecurityConfig: Configures security aspects, including user authentication and authorization.
- WebSocketConfig: Sets up WebSocket endpoints and message broker channels.

Login_System Package:

- LoginController: Handles web requests for user login and registration.
- User: The entity representing a system user, containing attributes like username, password, and email.
- UserDetailsServiceCustom: An implementation of Spring Security's

UserDetailsService, loading user details for authentication purposes.

- UserRepository: Spring Data JPA repository for accessing user information.
- UserService: The service interface for user-related operations.
- UserServiceHelper: Implements the UserService, providing concrete user management functionality.
- UserTransfer: A Data Transfer Object (DTO) for carrying user data safely across different layers of the application.

Data Flow and Component Interactions:

User Authentication Process:

Login:

- A user initiates the login process by entering credentials on the login.html page.
- The credentials are posted to the /login endpoint handled by Spring Security, which is configured in SpringSecurityConfig.
- Upon successful authentication, Spring Security redirects the user to the chat page, where a WebSocket session is established.

Registration:

- New users navigate to the register.html page to create an account.
- The registration form data is sent to the /register/save endpoint managed by LoginController.
- UserServiceHelper processes the registration, creating a new User entity and persisting it to the database using UserRepository.

Chat Functionality

Establishing WebSocket Connection:

- After login, chat.js establishes a WebSocket connection to the server, configured in WebSocketConfig, and managed by ChatController.
- The client subscribes to a public topic to receive messages and may send messages to server endpoint mappings defined in ChatController.

Message Handling:

- When a user sends a chat message, chat.js transmits the data over the WebSocket to an endpoint (/chat.sendMessage) handled by ChatController.
- ChatController receives the message and saves it to the MySQL database using ChatMessageRepository.
- The message is then broadcast to all subscribed clients through the /topic/public channel.

Joining and Leaving Notifications:

- When a user joins or leaves the chat, a special MessageType (JOIN or LEAVE) is sent by ChatController.
- The event is captured by ChatEventListener, which updates the list of active users and broadcasts the event to all connected clients.

Data Persistence:

- Chat messages and user data are stored in a MySQL database.
- ChatMessageRepository and UserRepository facilitate interaction with the

database to perform CRUD operations for chat messages and user entities, respectively.

Security:

- User authentication is enforced by Spring Security, that intercepts login requests for validation.
- Passwords are encrypted using BCrypt before being stored in the database.
- Spring Security also protects against CSRF attacks and ensures that WebSocket communication is only allowed for authenticated users.

Scalability and Performance:

- The application is designed to scale horizontally. As the user base grows, the application can be deployed across multiple servers behind a load balancer.
- WebSocket connections are stateful but lightweight, allowing the server to handle many concurrent users.
- Chat history is loaded in pages to prevent memory overload, reducing the load on the database and network.

Challenges and Solutions:

Challenges Faced:

1. User Authentication and Session Management:
   Maintain the user's authentication status in WebSocket communications to ensure all senders of the message is an authenticated user.
2. Real-time Communication Implementation:
   Establishing real-time chat functionalities among users, including join notifications and instant message delivery.
3. Frontend-Backend Data Synchronization:
   Ensuring that the information displayed on the frontend (like usernames) is in sync with the backend database.
4. Dynamic Updates to the User Interface:
   Updating the user interface in real-time when new messages are received or when users join/leave the chat room.
5. Dynamic Handling of User Information:
   Allowing users to change their username or password without refreshing the page.
6. Error Handling and Feedback:
   Providing user feedback visible to front-end when operations fail or errors occur.

Solutions Implemented:

1. Use Spring Security:
   Use Spring Security to integrate WebSocket for secure user authentication and session management.
2. Use Spring WebSocket:
   Use the WebSocket support provided by the Spring framework, real-time two-way communication between the server and the client is achieved.

3. Use Thymeleaf Template Engine:
   Use the Thymeleaf template engine to render user data on the server side to maintain front-end and back-end data consistency.
4. Use JavaScript and DOM Manipulation:
   Realized dynamic updates to the chat interface through JavaScript manipulation of the DOM.
5. Use AJAX Requests:
   Use AJAX to handle the user's request to change their username or password and update the user's status without requiring a page refresh.
6. Add Exception Handling Mechanism:
   Added an exception handling mechanisms on the backend, and ues JavaScript alerts to display error information at frontend JavaScript.