Chat APP Logic flow

User Connects to the Chat Application:

The user opens the chat application in their web browser, which establishes a WebSocket connection with the Spring Boot server.

WebSocket Connection Handshake:

The WebSocket connection is established, and the client sends a handshake request to the server.

Spring Boot WebSocket Endpoint:

- *The WebSocket connection is handled by the **ChatController** in the Spring Boot application.*
- *The **@MessageMapping("/chat")** annotation indicates that the **sendMessage** method should be invoked when a message is sent to the "**/chat**" destination.*

User Sends a Message:

- *The user types a message in the chat UI and sends it.*
- *The client sends a WebSocket message to the **"/chat**" destination with the user's message.*

Spring Boot Processes the Message:

- *The **sendMessage** method in the **ChatController** is invoked, receiving the **ChatMessage** object.*
- *The **ChatService** processes the message, which can include business logic, message validation, or any other required operations.*
- *The processed message is then sent to the Kafka topic (e.g., "**chat-room-arun**") using the **KafkaProducerService**.*

Kafka Message Distribution:

- *The Kafka producer sends the processed **ChatMessage** to the specified Kafka topic ("**chat-room-arun**").*
- *Kafka, being a distributed message broker, ensures the reliable distribution of messages to all interested consumers.*

Spring Boot Kafka Consumer:

- *Another instance of the **Spring Boot application**, running on a different server or as a separate process, acts as a **Kafka consumer** for the "**chat-room-arun**" topic.*
- *This **consumer** retrieves the message from Kafka and forwards it to the **WebSocketService**.*

WebSocket Broadcast:

- *The WebSocketService broadcasts the message to all connected clients subscribed to the "/topic/public" destination.*
- *The broadcasted message is sent back to the original user and other users in the chat room.*

User Receives the Message:

- *The WebSocket connection on the user's browser receives the broadcasted message.*
- *The chat UI updates to display the received message, and the user sees the message from the sender.*

Users can continue to send and receive messages, and the flow repeats for each new message.

This end-to-end chat flow demonstrates how messages flow from the client to the Spring Boot server, then to Apache Kafka for distribution, and back to clients through WebSockets. The use of Kafka enables a scalable and fault-tolerant message distribution system, and WebSockets provide real-time bidirectional communication between clients and the server.