STRT\_MEET\_001 – Real-Time Video Meeting Backend

This module enables real-time messaging, video signaling, and user presence features using WebSockets and STOMP protocol in Spring Boot. It does not use JWT authentication.

# WebSocketConfig.java

Purpose:  
- Configures WebSocket entry point  
- Enables STOMP messaging with application/broker prefixes

@Configuration

@EnableWebSocketMessageBroker

public class WebSocketConfig implements WebSocketMessageBrokerConfigurer {

@Override

public void registerStompEndpoints(StompEndpointRegistry registry) {

registry.addEndpoint("/ws").setAllowedOrigins("\*");

}

@Override

public void configureMessageBroker(MessageBrokerRegistry registry) {

registry.setApplicationDestinationPrefixes("/app");

registry.enableSimpleBroker("/topic");

}

}

Explanation:  
- registerStompEndpoints: Defines the endpoint clients connect to.  
- setAllowedOrigins: Allows all origins (CORS).  
- configureMessageBroker: Routes application messages and topic subscriptions.

# WebSocketController.java

Purpose:  
Handles STOMP messages from frontend clients for:  
- Chat  
- WebRTC signaling  
- Presence tracking

@Controller

public class WebSocketController {

@MessageMapping("/chat/{roomId}")

@SendTo("/topic/chat/{roomId}")

public ChatMessage handleChat(@DestinationVariable String roomId, ChatMessage chat) {

return chat;

}

@MessageMapping("/signal/{roomId}")

@SendTo("/topic/signal/{roomId}")

public SignalMessage handleSignaling(@DestinationVariable String roomId, SignalMessage signal) {

return signal;

}

@MessageMapping("/presence/{roomId}")

@SendTo("/topic/presence/{roomId}")

public UserPresenceMessage handlePresence(@DestinationVariable String roomId, UserPresenceMessage presence) {

return presence;

}

}

Each method accepts a message to a specific room and broadcasts it to the correct topic for subscribers to receive.

# DTO Classes

## ChatMessage.java

Used for transmitting plain chat messages in a room.

public class ChatMessage {

private String roomId;

private String sender;

private String message;

}

## SignalMessage.java

Carries WebRTC signaling data like offers, answers, or ICE candidates.

public class SignalMessage {

private String type;

private String senderId;

private String receiverId;

private String sdp;

private String candidate;

}

## UserPresenceMessage.java

Notifies users when others join, leave, or reconnect in a room.

public class UserPresenceMessage {

private String roomId;

private String userId;

private String status;

}

# Postman WebSocket Testing

To test the WebSocket connection in Postman, follow these steps:

## 1. CONNECT Frame

CONNECT

accept-version:1.2

heart-beat:10000,10000

\u0000

## 2. SUBSCRIBE Frame

SUBSCRIBE

id:sub-0

destination:/topic/chat/room123

\u0000

## 3. SEND Frame

SEND

destination:/app/chat/room123

content-type:application/json

{"sender":"Ranjit","message":"Hello","roomId":"room123"}\u0000

**JWT Token Authorization and WebSocket Setup in Postman**

**1. Login and Retrieve JWT Token**

* **Endpoint:** POST http://localhost:8080/auth/login
* **Headers:** No Authorization required here
* **Body (JSON):**

{

"username": "yourUsername",

"password": "yourPassword"

}

Expected Response (200 OK):

{

"code": 200,

"success": true,

"message": "Authentication successful",

"data": {

"role": "ROLE\_USER",

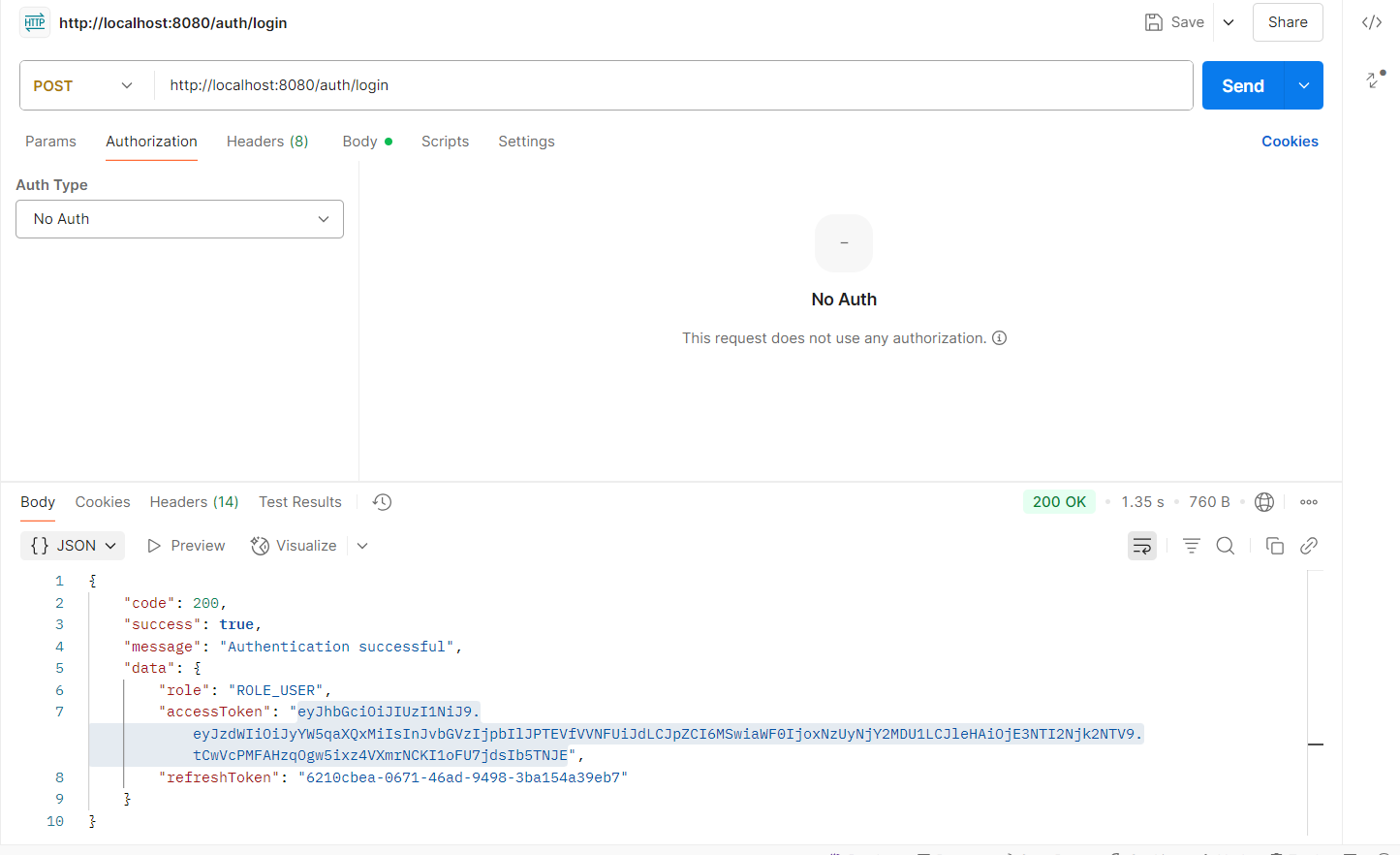
"accessToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9....",

"refreshToken": "..."

}

}

**-> Copy the accessToken (JWT) from the data field.**



**2. Paste Token into WebSocket Authorization Header**

**Websocket: Click on the "New" button → Select "WebSocket Request"**

**a. Open Postman WebSocket tab**

* **URL:** ws://localhost:8080/ws

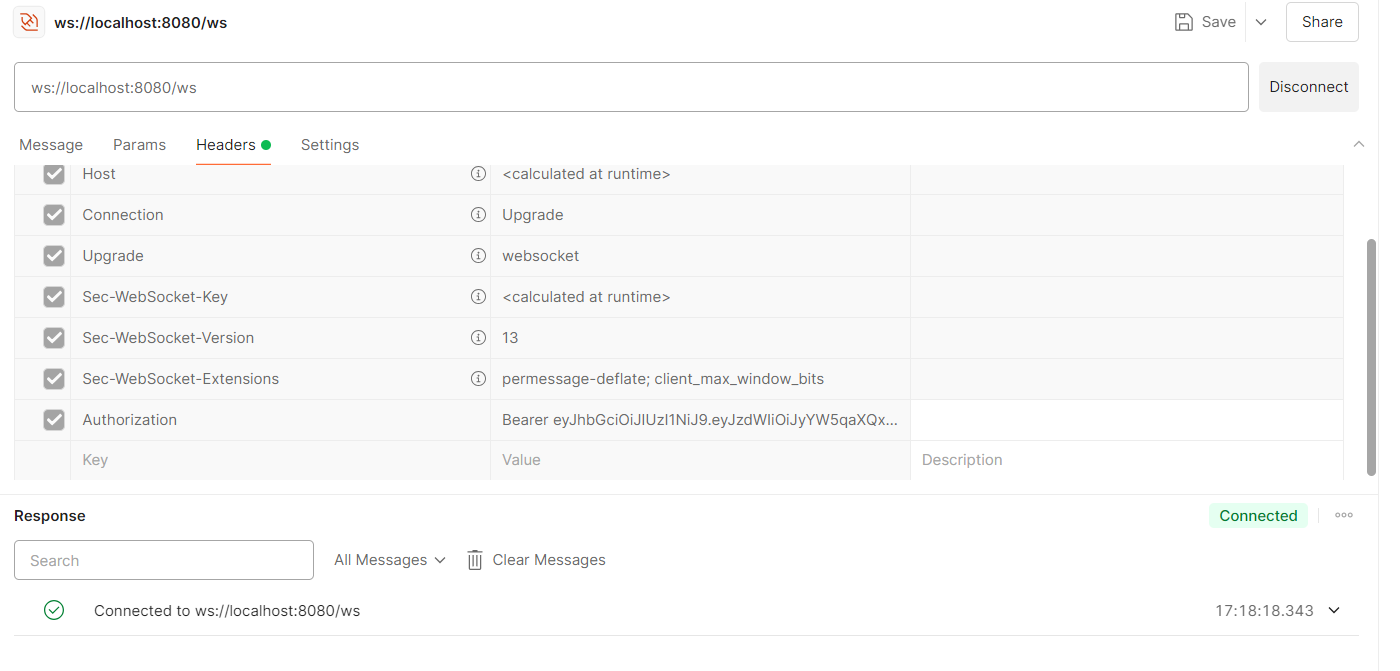
**b. Go to Headers tab**

* Add the following key–value pair:

Key: Authorization

Value: Bearer <Paste Your Token Here>

This header sends your JWT during WebSocket handshake.



**3. Connect to WebSocket**

* Click **Connect**
* You should now see:

✅ Connected to ws://localhost:8080/ws

**4. Send a STOMP CONNECT Frame**

Under the **Message** tab, paste:

CONNECT

accept-version:1.2

heart-beat:10000,10000

\0

After sending this, you should receive a successful CONNECTED frame response

5. **Subscribe to a Topic (SUBSCRIBE Frame)**

SUBSCRIBE

id:sub-0

destination:/topic/chat/room123

\0

This means the client will now listen to any messages sent to /topic/chat/room123.

6. **Send a Message (SEND Frame)**

SEND

destination:/app/chat/room123

content-type:application/json

{"sender":"Ranjit","message":"Hello Everyone","roomId":"room123"}\0

This message will be broadcast to all clients subscribed to /topic/chat/room123.

**Final Flow Summary**

| Step | Action | Purpose |
| --- | --- | --- |
| 1️⃣ | Login via REST /auth/login | Get JWT token |
| 2️⃣ | Add Authorization: Bearer <token> | Use token in WebSocket header |
| 3️⃣ | Connect to ws://localhost:8080/ws | Open WebSocket with STOMP |
| 4️⃣ | Send CONNECT frame | Establish STOMP session |
| 5️⃣ | Send SUBSCRIBE frame | Listen to a topic |
| 6️⃣ | Send SEND frame with JSON message | Chat/signaling/presence communication |