WEBSOCKET SERVER WITH STOMP CLIENT

1. What are Websockets?

- WebSocket is a protocol which allows for communication between the client and the server/endpoint using a single TCP connection. Sounds a bit like HTTP doesn't it? The advantage WebSocket has over HTTP is that the protocol is full-duplex (allows for simultaneous two-way communication) and it's header is much smaller than that of a HTTP header, allowing for more efficient communication even over small packets of data.

2. WebSocket Client

- We are using Stomp as a client which will establish a connection to our websocket server and communicate with it.

2.1. Dependencies required

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-websocket</artifactId>
</dependency>

<dependency>
    <groupId>org.webjars</groupId>
    <artifactId>sockjs-client</artifactId>
    <version>1.1.2</version>
</dependency>
<dependency>
    <groupId>org.webjars</groupId>
    <artifactId>stomp-websocket</artifactId>
    <version>2.3.3-1</version>
</dependency>
```

2.2. Establishing a connection

```
List<Transport> transports = new ArrayList<>(1);
transports.add(new WebSocketTransport(new StandardWebSocketClient()));
WebSocketClient transport = new SockJsClient(transports);
WebSocketStompClient stompClient = new WebSocketStompClient(transport);

stompClient.setMessageConverter(new MappingJackson2MessageConverter());

StompSessionHandler sessionHandler = new MyStompSessionHandler();
stompClient.connect(URL, sessionHandler);
```

- Here *URL* can be any websocket URL in *String* format.
  eg: "`ws:`*`//localhost:8082/socket/user/`*"

- Here `ws` is the Web-Socket protocol, `localhost:8082` is the server + port number on which your websocket server is running, and `/socket/user` is the end-point on which your server is going to receive the message.

- `MyStompSessionHandler` is a class which extends `StompSessionHandlerAdapter` class and maintains the session so that client can send messages.

2.3. Handling the Session

- `MyStompSessionHandler.class`

```
public class MyStompSessionHandler extends StompSessionHandlerAdapter {

   @Override
   public void afterConnected(StompSession session, StompHeaders
                                             connectedHeaders) {
       session.subscribe("/user/topic/messages", this);
   }

   @Override
   public void handleException(StompSession session, StompCommand command,
            StompHeaders headers, byte[] payload, Throwable exception) {
       System.out.println("Got an exception" + exception.toString());
   }

   @Override
   public Type getPayloadType(StompHeaders headers) {
       return Message.class;
   }

   @Override
   public void handleFrame(StompHeaders headers, Object payload) {
       System.out.println("Received : " + payload);
   }
}
```

- After connecting, a client can subscribe to a specific topic and send/receive data only through that topic.
  - Here client is subscribing to `/user/topic/messages` endpoint.
- If any exception happens while connection, it goes to the `handleException()` method.

2.4. Sending the data

- Message.class

```java
public class Message {

   private String message;
   private Date timeStamp;

   public String getMessage() {
       return message;
   }

   public void setMessage(String message) {
       this.message = message;
   }

   public Date getTimeStamp() {
       return timeStamp;
   }

   public void setTimeStamp(Date timeStamp) {
       this.timeStamp = timeStamp;
   }
}
```

- The object of this message class will be send through websocket.

```java
Message msg = new Message();
msg.setMessage("Hello");
msg.setTimeStamp(new Date());

session.send("/socket/app/user/msg", msg);
```

- Here `/msg` is the controller endpoint in MessageMapping annotation inside websocket server.

2.4. Disconnection

```java
session.disconnect();
```

2.5. Implementing Security

- For implementing security from client side, we are sending HTTP headers while connection.

```
HttpHeaders httpHeaders = new HttpHeaders();
httpHeaders.add("Authorization", “bearerToken”);

WebSocketHttpHeaders headers = new WebSocketHttpHeaders(httpHeaders);

StompSessionHandler sessionHandler = new MyStompSessionHandler();
stompClient.connect(URL, headers, sessionHandler);
```

3. Websocket Server

3.1. Dependencies required

```
<dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-websocket</artifactId>
</dependency>
```

```
<dependency>
   <groupId>org.springframework.security</groupId>
   <artifactId>spring-security-web</artifactId>
   <version>${spring-security.version}</version>
</dependency>
```

```
<dependency>
   <groupId>org.springframework.security</groupId>
   <artifactId>spring-security-config</artifactId>
   <version>${spring-security.version}</version>
</dependency>
```

```
<dependency>
   <groupId>org.springframework.security</groupId>
   <artifactId>spring-security-messaging</artifactId>
   <version>5.1.2.RELEASE</version>
</dependency>
```

```
<dependency>
   <groupId>org.springframework</groupId>
   <artifactId>spring-websocket</artifactId>
   <version>5.1.3.RELEASE</version>
</dependency>
```

```
<dependency>
   <groupId>org.springframework</groupId>
   <artifactId>spring-messaging</artifactId>
   <version>5.1.3.RELEASE</version>
</dependency>
```

```
<dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-security</artifactId>
   <version>2.1.1.RELEASE</version>
</dependency>
```

```
<dependency>
   <groupId>org.springframework.security.oauth</groupId>
   <artifactId>spring-security-oauth2</artifactId>
   <version>2.3.4.RELEASE</version>
</dependency>
```

3.2. WebSocket Server configuration

- First, we need to add a configuration to our application and annotate this class with *@EnableWebSocketMessageBroker*.
- It enables WebSocket message handling, backed by a message broker:

```
@Configuration
@EnableWebSocketMessageBroker
public class WebSocketConfig extends
AbstractWebSocketMessageBrokerConfigurer {

   @Override
   public void configureMessageBroker(MessageBrokerRegistry config) {
       config.enableSimpleBroker("/socket/user/topic");
       config.setApplicationDestinationPrefixes("/socket/app/user");
   }

   @Override
   public void registerStompEndpoints(StompEndpointRegistry registry) {
       registry.addEndpoint("/socket/user").withSockJS();
   }
}
```

- The method *configureMessageBroker* is used to configure the message broker. First, we enable an in-memory message broker to carry the messages back to the client on destinations prefixed with “/topic”.
- We complete our simple configuration by designating the “/app” prefix to filter destinations targeting application annotated methods (via *@MessageMapping*).
- The *registerStompEndpoints* method registers the “/chat” endpoint, enabling Spring’s STOMP support.
- This endpoint, when prefixed with “/app”, is the endpoint that the *send()* method from client side is mapped to handle.

3.3. Receiving a Message

- Spring’s approach to working with STOMP messaging is to associate a controller method to the configured endpoint. This is made possible through the *@MessageMapping* annotation.
- The association between the endpoint and the controller gives us the ability to handle the message if needed.

```
@RestController
public class WSController {

   @MessageMapping("/msg")
   @SendTo("/topic/messages")
   public void receive(@RequestBody Message msg) throws Exception {
       ObjectMapper objectMapper = new ObjectMapper();
       String json = objectMapper
                        .writerWithDefaultPrettyPrinter()
                        .writeValueAsString(msg);
       System.out.println(json);
   }
}
```

3.4. Implementing Security

- Socket Security Configuration

```
@Configuration
public class SocketSecurityConfig extends ce Server Configuration


@Configuration
@EnableResourceServer
class ResourceServerConfiguration extends ResourceServerConfigurerAdapter {

   @Override
   public void configure(HttpSecurity http) throws Exception {
          http.cors().disable()
                     .authorizeRequests()
                     .antMatchers("/socket/**").permitAll()


            .


            .


            .
```

```
    }
}
             AbstractSecurityWebSocketMessageBrokerConfigurer {
    @Override
    protected void configureInbound(MessageSecurityMetadataSourceRegistry messages) {
        messages.simpDestMatchers("/socket/**").authenticated();
    }

    @Override
    protected boolean sameOriginDisabled() {
        return true;
    }
}
```

- Resource Server Configuration

```
@Configuration
@EnableResourceServer
class ResourceServerConfiguration extends ResourceServerConfigurerAdapter {

   @Override
   public void configure(HttpSecurity http) throws Exception {
          http.cors().disable()
                     .authorizeRequests()
                     .antMatchers("/socket/**").permitAll()
           .
           .
           .
   }
}
```