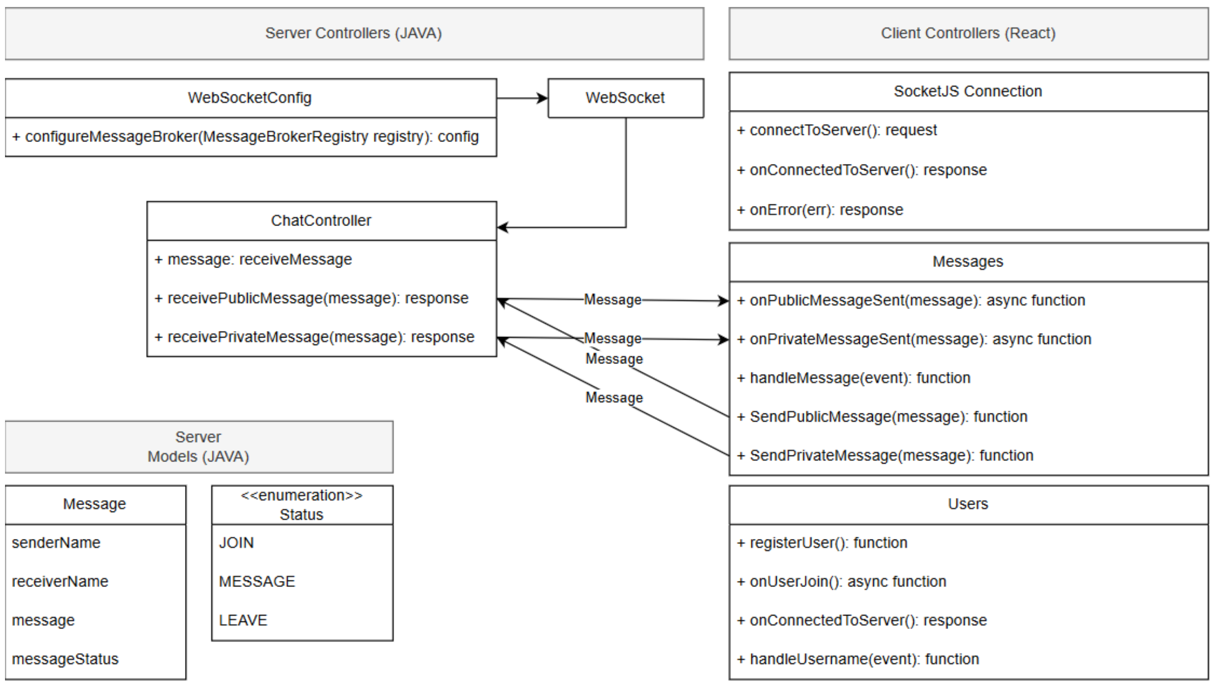
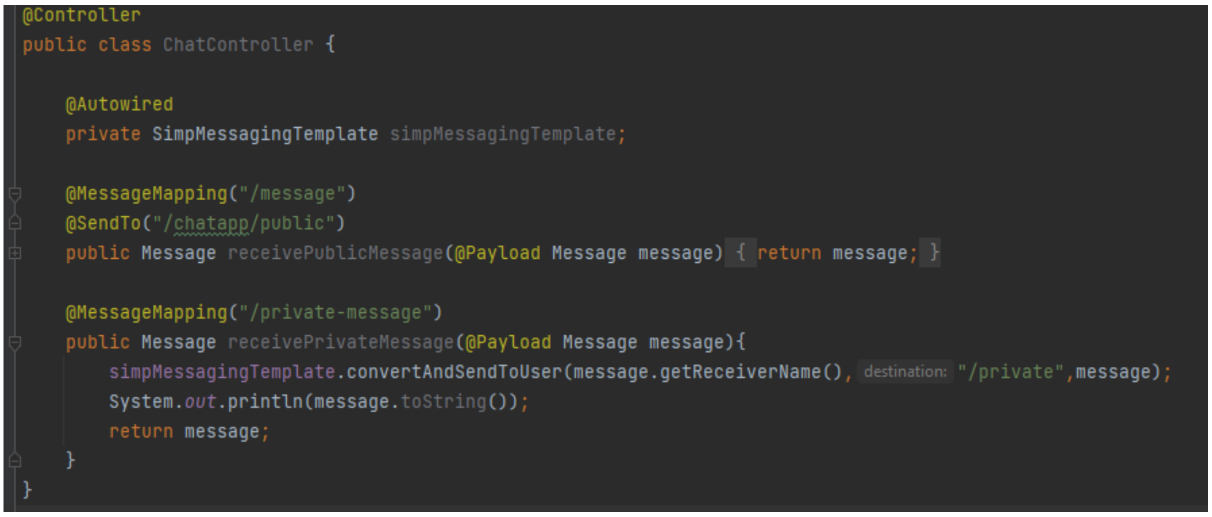
Entity Diagram:


Server Controllers (JAVA)
Client Controllers (React)
WebSocketConfig
+ configureMessageBroker(MessageBrokerRegistry registry): config
WebSocket
SocketJS Connection
+ connectToServer(): request
+ onConnectedToServer(): response
+ onError(err): response
ChatController
+ message: receiveMessage
+ receivePublicMessage(message): response
+ receivePrivateMessage(message): response
Message
Message
Message
Message
Messages
+ onPublicMessageSent(message): async function
+ onPrivateMessageSent(message): async function
+ handleMessage(event): function
+ SendPublicMessage(message): function
+ SendPrivateMessage(message): function
Server Models (JAVA)
Message
senderName
receiverName
message
messageStatus
<<enumeration>> Status
JOIN
MESSAGE
LEAVE
Users
+ registerUser(): function
+ onUserJoin(): async function
+ onConnectedToServer(): response
+ handleUsername(event): function

Data Layer:



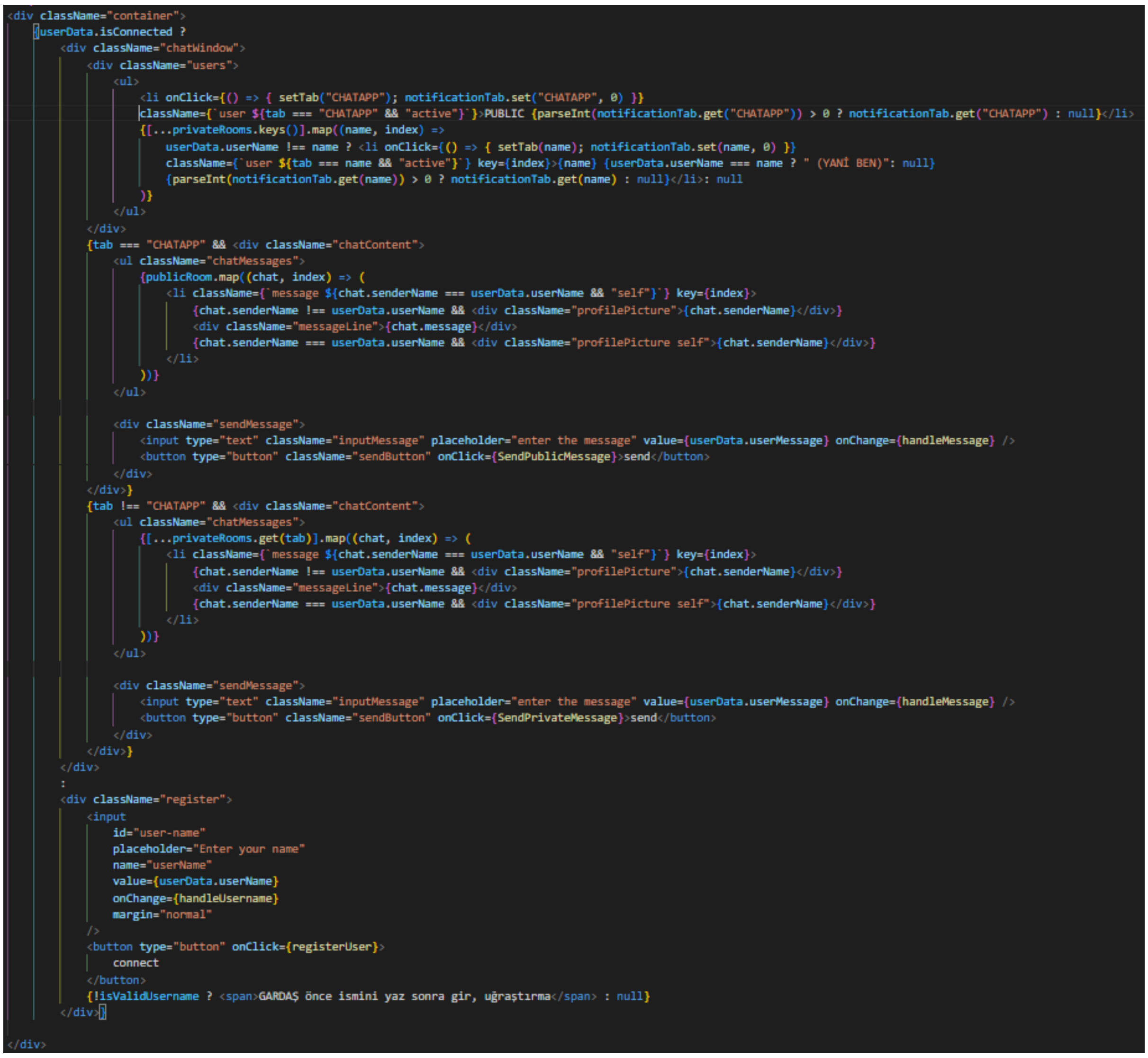
```
@Controller
public class ChatController {

    @Autowired
    private SimpMessagingTemplate simpMessagingTemplate;

    @MessageMapping("/message")
    @SendTo("/chatapp/public")
    public Message receivePublicMessage(@Payload Message message) { return message; }

    @MessageMapping("/private-message")
    public Message receivePrivateMessage(@Payload Message message){
        simpMessagingTemplate.convertAndSendToUser(message.getReceiverName(), destination: "/private",message);
        System.out.println(message.toString());
        return message;
    }
}
```

Presentation Layer:



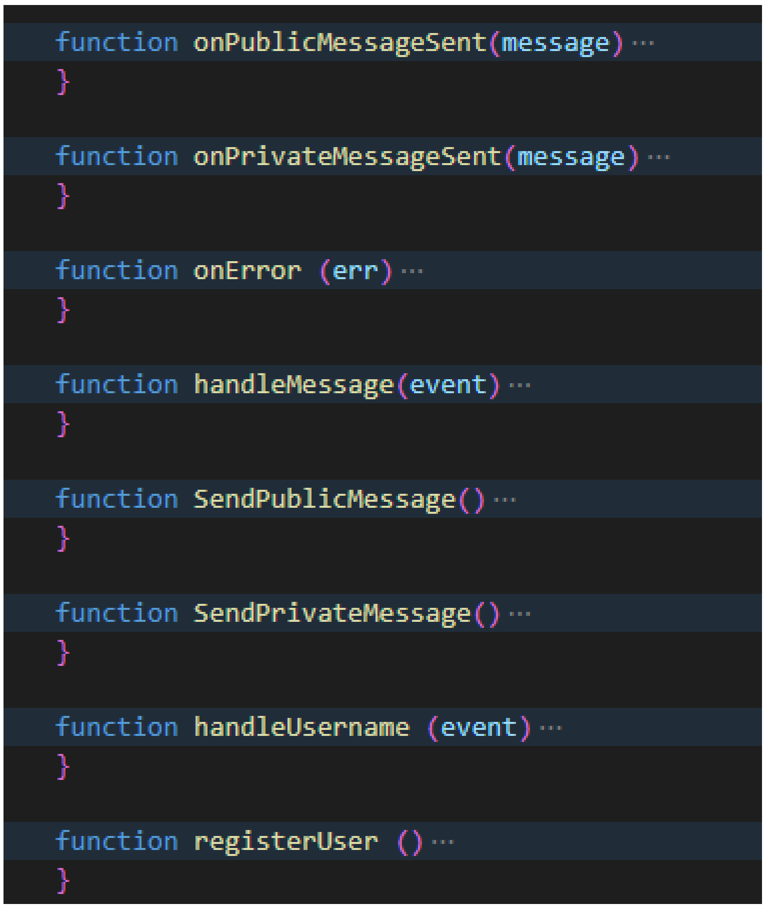
```jsx
<div className="container">
    {userData.isConnected ?
        <div className="chatWindow">
            <div className="users">
                <ul>
                    <li onClick={() => { setTab("CHATAPP"); notificationTab.set("CHATAPP", 0) }}
                    className={`user ${tab === "CHATAPP" && "active"}`}>PUBLIC {parseInt(notificationTab.get("CHATAPP")) > 0 ? notificationTab.get("CHATAPP") : null}</li>
                    {[...privateRooms.keys()].map((name, index) =>
                        userData.userName !== name ? <li onClick={() => { setTab(name); notificationTab.set(name, 0) }}
                        className={`user ${tab === name && "active"}`} key={index}>{name} {userData.userName === name ? " (YANİ BEN)": null}
                        {parseInt(notificationTab.get(name)) > 0 ? notificationTab.get(name) : null}</li>: null
                    )}
                </ul>
            </div>
            {tab === "CHATAPP" && <div className="chatContent">
                <ul className="chatMessages">
                    {publicRoom.map((chat, index) => (
                        <li className={`message ${chat.senderName === userData.userName && "self"}`} key={index}>
                            {chat.senderName !== userData.userName && <div className="profilePicture">{chat.senderName}</div>}
                            <div className="messageLine">{chat.message}</div>
                            {chat.senderName === userData.userName && <div className="profilePicture self">{chat.senderName}</div>}
                        </li>
                    ))}
                </ul>

                <div className="sendMessage">
                    <input type="text" className="inputMessage" placeholder="enter the message" value={userData.userMessage} onChange={handleMessage} />
                    <button type="button" className="sendButton" onClick={SendPublicMessage}>send</button>
                </div>
            </div>}
            {tab !== "CHATAPP" && <div className="chatContent">
                <ul className="chatMessages">
                    {[...privateRooms.get(tab)].map((chat, index) => (
                        <li className={`message ${chat.senderName === userData.userName && "self"}`} key={index}>
                            {chat.senderName !== userData.userName && <div className="profilePicture">{chat.senderName}</div>}
                            <div className="messageLine">{chat.message}</div>
                            {chat.senderName === userData.userName && <div className="profilePicture self">{chat.senderName}</div>}
                        </li>
                    ))}
                </ul>

                <div className="sendMessage">
                    <input type="text" className="inputMessage" placeholder="enter the message" value={userData.userMessage} onChange={handleMessage} />
                    <button type="button" className="sendButton" onClick={SendPrivateMessage}>send</button>
                </div>
            </div>}
        </div>
        :
        <div className="register">
            <input
                id="user-name"
                placeholder="Enter your name"
                name="userName"
                value={userData.userName}
                onChange={handleUsername}
                margin="normal"
            />
            <button type="button" onClick={registerUser}>
                connect
            </button>
            {!isValidUsername ? <span>GARDAŞ önce ismini yaz sonra gir, uğraştırma</span> : null}
        </div>}

</div>
```

Business Layer:



```
function onPublicMessageSent(message)···
}

function onPrivateMessageSent(message)···
}

function onError (err)···
}

function handleMessage(event)···
}

function SendPublicMessage()···
}

function SendPrivateMessage()···
}

function handleUsername (event)···
}

function registerUser ()···
}
```

Components:

Message.java

Includes necessary variables for a message.

Status.java

Includes ENUM of state of the message.

ChatController.java

Includes functions of when receiving public or private messages.

WebSocketConfig.java

Includes config methods for Spring Boot Web Sockets.

ChatApp.js

Handles connecting SocketIO, connection & registering users and handling message sends.