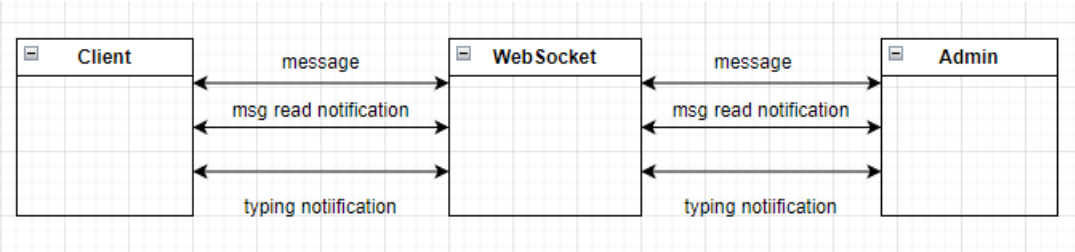
Chat System for Client Support - documentatie

1. Arhitectura conceptuala



Aplicatia de mai sus, integrata in aplicatia de la assignment-ul 1 ca si assignment-ul 2, reprezinta un sistem de chat prin intermediul caruia utilizatorii platformei energetice pot vizualiza si trimite mesaje intre ei, in acelasi timp primind si notificari in momentul in care partenerii de conversatie citesc mesajele sau scriu un nou mesaj.

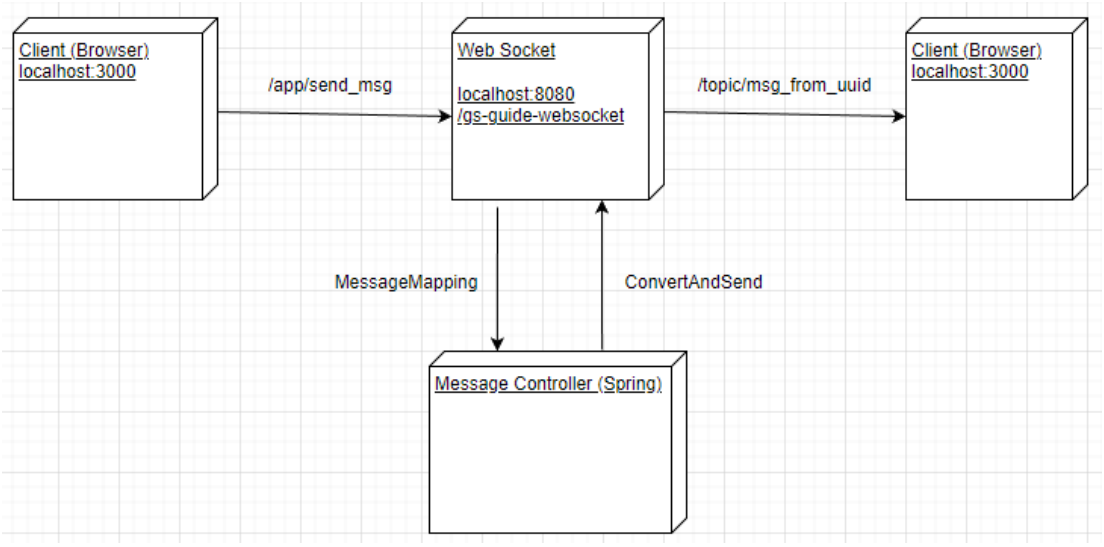
Pentru comunicarea asincrona intre doi utilizatori ai aplicatiei (fie doi clienti, fie doi administratori, fie un client si un administrator, am folosit aceeasi tehnologie ca si la assignment-ul anterior, si anume websockets. Este folosit acelasi websocket configurat in backend la assignment-ul 2, cu endpoint-ul */gs-guide-websocket* prin metoda registry.addEndpoint, iar broker-ul de websocket se afla la adresa */topic*, in timp ce adresele primite de socket din partea clientului au prefixul */app*. Notificarile trimise de websocket clientilor sunt distribuite in functie de clientul care urmeaza sa primeasca notificarea si filtrate in frontend dupa utilizatorul curent logat in aplicatie. De aceasta configurare se ocupa clasa WebSocketConfig, adnotata cu @Configuration si @EnableWebSocketBroker. Clientul se conecteaza la socket-ul configurat in backend folosind un protocol STOMP (Simple Text Oriented Messaging Protocol). Teoretic backend-ul trimite notificari la toti clientii care acceseaza pagina web, dar vor vedea doar notificarile ce privesc device-urile lor. Pentru lucrul cu socket-uri in Javascript a fost nevoie de librariile sockjs-client si stompjs.

Chat-ul se conecteaza o alta instanta a socket-ului */gs-guide-websockets* decat cea folosita de Message Receiver pentru transmiterea de notificari privind consumul de eergie al device-urilor. Prin intermediul socket-ului folosit de chat, utilizatorul poate trimite mesaje altor utilizator la adresa */app/send_msg*, poate anunta un alt utilizator ca i-a citit mesajul la adresa */app/send_read* sau ca ii scrie un mesaj la adresa */app/send_typing*. Tot prin intermediul socket-ului, clientul se aboneaza (subscribe) la adresele */topic/msg_from_user_uuid* (coada de mesaje din partea utilizatorului *user_uuid*), la adresele */topic/read_from_user_uuid* (coada de notificari de mesaje citite din partea utilizatorului *user_uuid*) si la adresele */topic/typing_from_user_uuid* (coada de notificari de scriere de mesaje din partea utilizatorului *user_uuid*) specifice fiecarui utilizator.

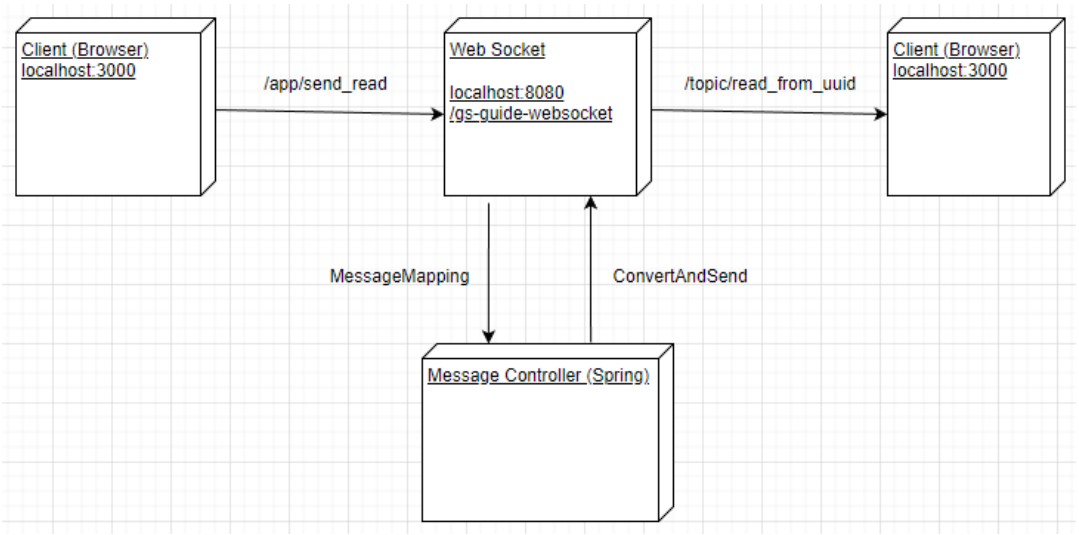
Socket-ul redirectioneaza mesajele si notificarile intre utilizatori. In timp ce utilizatorul comunica asincron cu socket-ul la adresele de prefix */app*, socket-ul comunica in mod sincron cu utilizatorii, redirectionandu-le mesajele si notificarile corespunzatoare din partea altor utilizatori, folosind adrese de prefix */topic*. Redirectionarea este realizata de controllerul *MessageController*, care mapeaza adresele de unde vin datele de la clienti cu adnotarea @MessageMapping la cate o functie cu care transmite acele date catre clientii corespunzatori la adresele */topic* corespunzatoare, folosind functia *convertAndSend*.

2. Diagrama de deployment UML

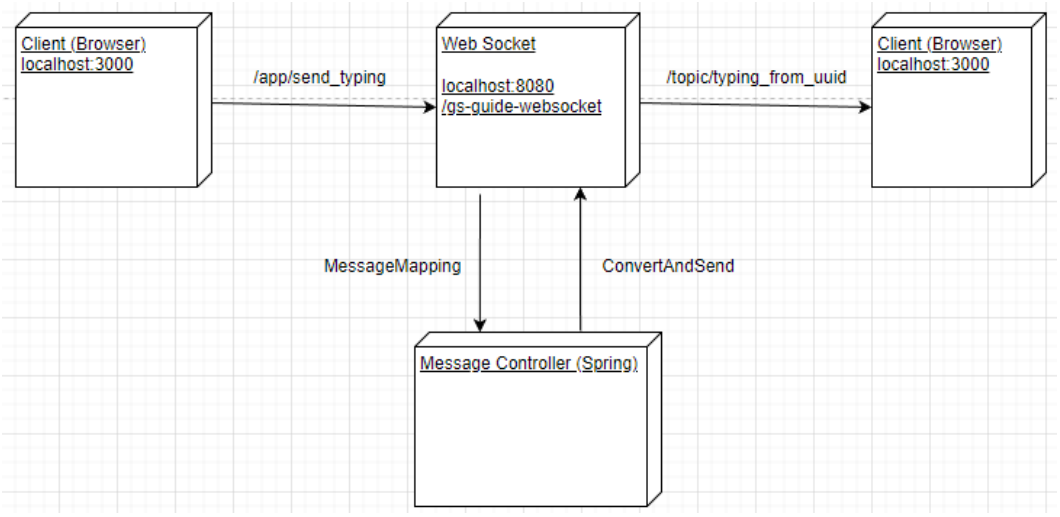
Trimitere de mesaje



Trimitere de notificari de citire



Trimitere de notificari de scriere



3. Bibliografie

- https://spring.io/guides/gs/messaging-stomp-websocket/
- https://www.callicoder.com/spring-boot-websocket-chat-example/