DISTRIBUTED SYSTEMS

Assignment 3

Web sockets and security

STUDENT NAME: Marin Andreea

GROUP: 30442/1

**CONTENTS**

1. Objectives
2. Requirements
3. Conceptual architecture of the distributed system
4. UML Deployment Diagram
5. Build and execution considerations
6. **Objectives**

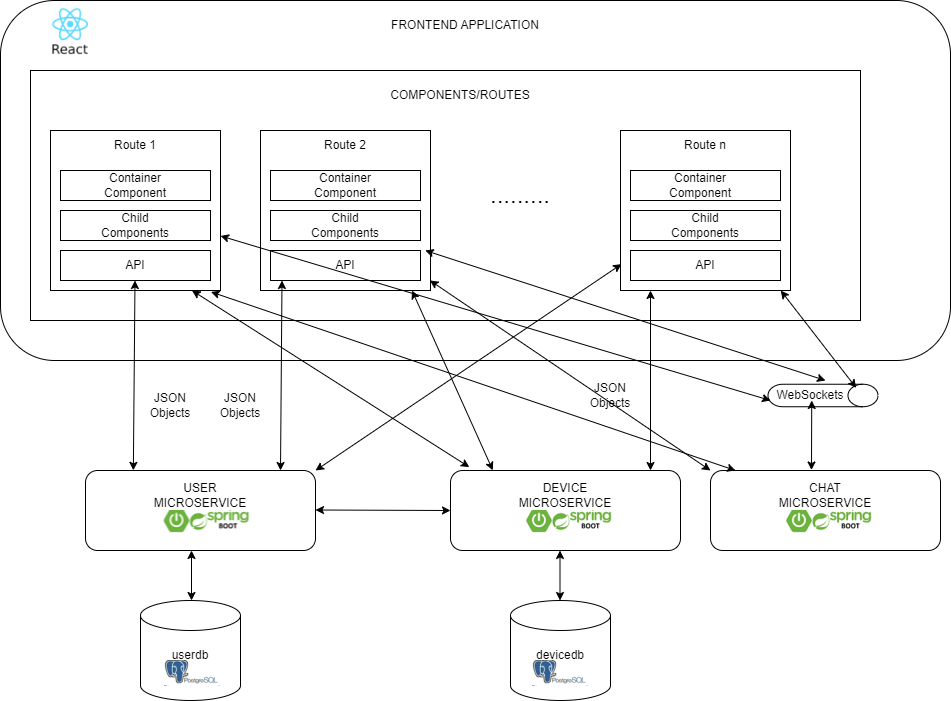
The main objective of this project is to develop a chat microservice and an authorization component for the Energy Management System. The authorization component should provide secured access of users to systems’ microservices. The chat microservice should allow communication between the users and the administrator of the system, allowing them to ask questions and receive answers.

1. **Requirements**

Functional, non-functional requirements and implementation techniques were identified for this project.

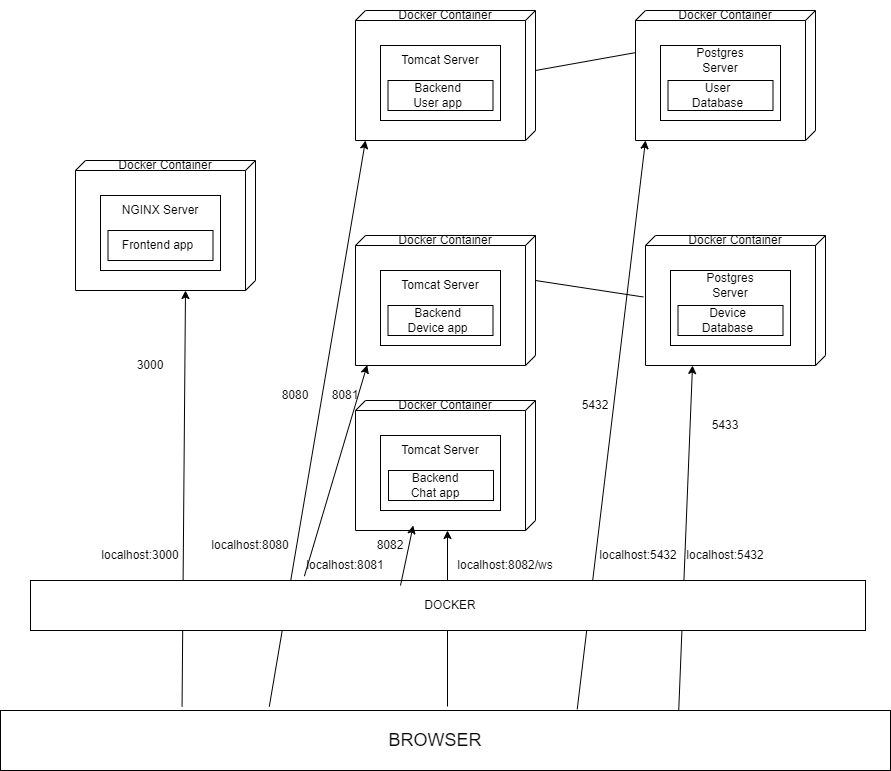
* **Functional requirements**
* The front-end application displays a chat box where users can type messages
* The message is sent asynchronously to the administrator, that receives the message together with the user identifier, being able to start a chat with the user
* Messages can be sent back and forth between the user and the administrator during a chat session
* The administrator can chat with multiple users at once
* A notification is displayed for the user (e.g., typing) while the administrator from the other end of communication types of its message and vice versa
* The authorization service generates access tokens to the client application. The tokens will be used to access other microservices
* **Non-functional requirements**
* One of the existing services should be chosen as the authorization server
* The application should use authentication to restrict users to access the administrator pages
* **Implementation techniques**
* The following techniques were used for developing the application: REST for microservices (Java Spring REST) and JavaScript-based framework for client applications (ReactJS)
* The chat component uses web sockets technology
* The authorization component uses JWT based authorization - for user’s authentication and authorization to all microservices and generates tokens that will be recognized by other microservices which share the same secret key as the authorization service

1. **Conceptual diagram**

****

* **Frontend Application**
* A new package was introduced and contains the JavaScript and CSS classes related to the chat microservice. As WebSockets provide a bi-directional communication, data is sent from the client to the server and vice-versa. These classes manage opening a connection to the server, sending data to the server and receiving data from the server. A notification is also displayed for the user when the administrator types the message and vice-versa.
* When an API call is performed, the jwt token corresponding to the authenticated user is sent to the server in the request header in order to be verified.
* **User Microservice**
* The following packages contain classes that act as the authorization component:
* Config package: is the package in charge of generating tokens when a user tries to log in and validate those tokens, confirm whether the credentials of the users are correct and check if they have the corresponding authority. A secret key is used when generating the access tokens so that the other microservices that share the same secret key will recognize the token.
* Auth package: defines a classes for authentication request and response, an authentication service where the log in method is implemented and the authentication controller. The credentials needed for authentication are the username and the password.
* **Device Microservice**
* Based on the shared secret key, the microservice recognizes the access token and performs a call to the User Microservice in order to see whether the token is valid and the user has the authority to perform the corresponding request
* **Chat Microservice**
* This microservice is the one managing the operation of sending messages back and forth between the administrator and other users. It consists of the following packages:
* ConfigWebSocket package: constains the WebSocketConfig class in which two methods are overridden: public void registerStompEndpoints(StompEndpointRegistry registry) method that configures the WebSocket endpoint for clients to connect to by registering a WebSocket endpoint at “/ws” path and public void configureMessageBroker(MessageBrokerRegistry registry) method that configures the message broker, defines the prefix to use for mapping message-handling methods on the server (messages sent to destinations prefixed with "/app" will be routed to methods annotated with @MessageMapping
* Controller package: contains the ChatController class which handles WebSocket communication for a chat application. It defines several methods annotated with @MessageMapping and @SendTo to process incoming WebSocket messages and send responses to specific destinations.
* Model package: contains the following classes: Message class which represents the message in the chat application and contains the following fields: senderName, receiverName, message, date, status; Status enum class: used for setting a status to the message: MESSAGE, TYPING, STOP-TYPING and TypingNotification class used for describing a notification when somebody is typing a message
* Security package: used for validating the incoming tokens

1. **UML Deployment Diagram**



On the host computer runs the docker runtime that will host six containers, one for each application:

• Docker container for frontend application – runs a NGINX server and maps local port 3000 to host computer port 3000

• Three Docker containers for backend applications – run three TOMCAT servers and map local port 8080 to host computer port 8080, local port 8081 to host computer port 8081 and local port 8082 to host computer port 8082

• Two Database containers for database servers – run two Postgres servers and map local port 5432 to host computer port 5432 and host computer port 5433

This means that from the host computer we can access the servers within the containers as follows:

• Frontend application: localhost: 3000

• Backend application: localhost: 8080, 8081 and 8082

• Postgres server: localhost: 5432

1. **Build and execution considerations**

* When running the application locally, the following steps need to be followed:
* Open user and device microservice applications in IntelliJ
* Start running the applications simultaneously
* Open Frontend application, open a terminal and run the command: “npm start”
* After running the command “npm start” in frontend, a web page will appear. Note that, at the beginning, the userdb contains a table “userr” with a single record (user with admin role) and the devicedb contains two tables: device and user, which are empty.
* The first step is to log in as admin, add new users with client role, add new devices and map devices with the new users added and then the remained functionalities can also be tested
* In order to test the chat functionality, after logging in, the user should click the button “Chat” and will be redirected to the corresponding chat page based on his role
* The credentials for login are: username: “andre”, password: “andre”
* For testing the functionality assigned to users with client role, the logout is necessary and then login with the credentials of a newly introduced client user
* When running the application in docker, the following steps need to be followed:
* Enter the docker desktop application
* Start the four containers
* Open the application in browser by introducing the URL : “http://localhost:3000”
* Note that, at the beginning, the userdb contains a table “userr” with a single record (user with admin role) and the devicedb contains two tables: device and user, which are empty.
* The first step is to log in as admin, add new users with client role, add new devices and map devices with the new users added and then the remained functionalities can also be tested
* In order to test the chat functionality, after logging in, the user should click the button “Chat” and will be redirected to the corresponding chat page based on his role
* The credentials for login are: username: “andre”, password: “andre
* For testing the functionality assigned to users with client role, the logout is necessary and then login with the credentials of a newly introduced client user