MINISTERUL EDUCAȚIEI ȘI CERCETĂRII ȘTIINȚIFICE

UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE
DEPARTAMENTUL CALCULATOARE

DISTRIBUTED SYSTEMS

Assignment 3

WebSockets and Security

Student: Buda Andreea Rodica
Facultatea de Automatică și Calculatoare
Specializarea Calculatoare
Grupa 30244

Cuprins

1.Obiectivul Proiectului 3
2. Analiza 3
2.1 Cerințe Funcționale 3
2.2 Cerințe Non-Funcționale 3
2.3 Tehnologii Utilizate 3
2.4 Cazuri de Utilizare și Scenarii 4
3. Proiectare 4
3.1 Arhitectura Conceptuală a Sistemului 4
3.2 Diagrama de Deployment 5
4. Implementare 6
4.1 Instrumente Utilizate 7
4.2 Fișier README 7
5. Bibliografie 8

1.Obiectivul Proiectului

Scopul acestui proiect este de a dezvolta un microserviciu de chat care permite comunicarea bidirecțională între utilizatori și administratorii sistemului, precum și o componentă de autorizare pentru accesul securizat la microservicii. Chat-ul trebuie să permită:

- Comunicarea asincronă între utilizatori și administratori.
- Notificarea utilizatorilor despre citirea mesajelor.
- Notificarea într-un mod vizual între participanții conversației atunci când unul dintre ei tastează.

De asemenea, componenta de autorizare trebuie să ofere autentificare și autorizare securizată bazată pe JSON Web Tokens (JWT) folosind Spring Security.

2. Analiza

2.1 Cerințe Funcționale

1. **Microserviciu de chat**:
    - Chat box în aplicația front-end pentru introducerea mesajelor.
    - Trimiterea asincronă a mesajelor de la utilizatori către administratori.
    - Schimb bidirecțional de mesaje între utilizatori și administratori.
    - Notificări despre citirea mesajelor și starea de tastare.
2. **Componenta de autorizare**:
    - Autentificare securizată folosind JWT.
    - Integrarea componentelor de autorizare în microserviciul de management al utilizatorilor.

2.2 Cerințe Non-Funcționale

1. **Securitate**:
    - Toate conexiunile între componente trebuie să fie securizate prin HTTPS.
    - Validarea JWT pentru toate cererile către microservicii.
2. **Scalabilitate**:
    - Sistemul trebuie să suporte mai mulți utilizatori simultan, atât pentru chat, cât și pentru autorizare.
3. **Performanță**:
    - Răspuns rapid la evenimentele din chat, cum ar fi tastarea sau citirea mesajelor.
4. **Portabilitate**:
    - Sistemul trebuie să fie containerizat folosind Docker pentru portabilitate și ușurință în implementare.

2.3 Tehnologii Utilizate

Backend:

- **Limbaj:** Java.
- **Framework:** Spring Boot.
- **Autentificare și autorizare:** Spring Security JWT.
- **WebSockets:** Pentru comunicare bidirecțională în timp real.
- **Persistență:** PostgreSQL.
- **Middleware:** RabbitMQ pentru mesageria dintre microservicii.

Frontend:

- **Limbaj:** JavaScript.
- **Framework:** React pentru interfața utilizator.

Orchestrare și Securitate:

- **Containerizare:** Docker și Docker Compose.
- **Reverse Proxy:** Traefik pentru load balancing și TLS.

2.4 Cazuri de Utilizare și Scenarii

Cazuri de Utilizare

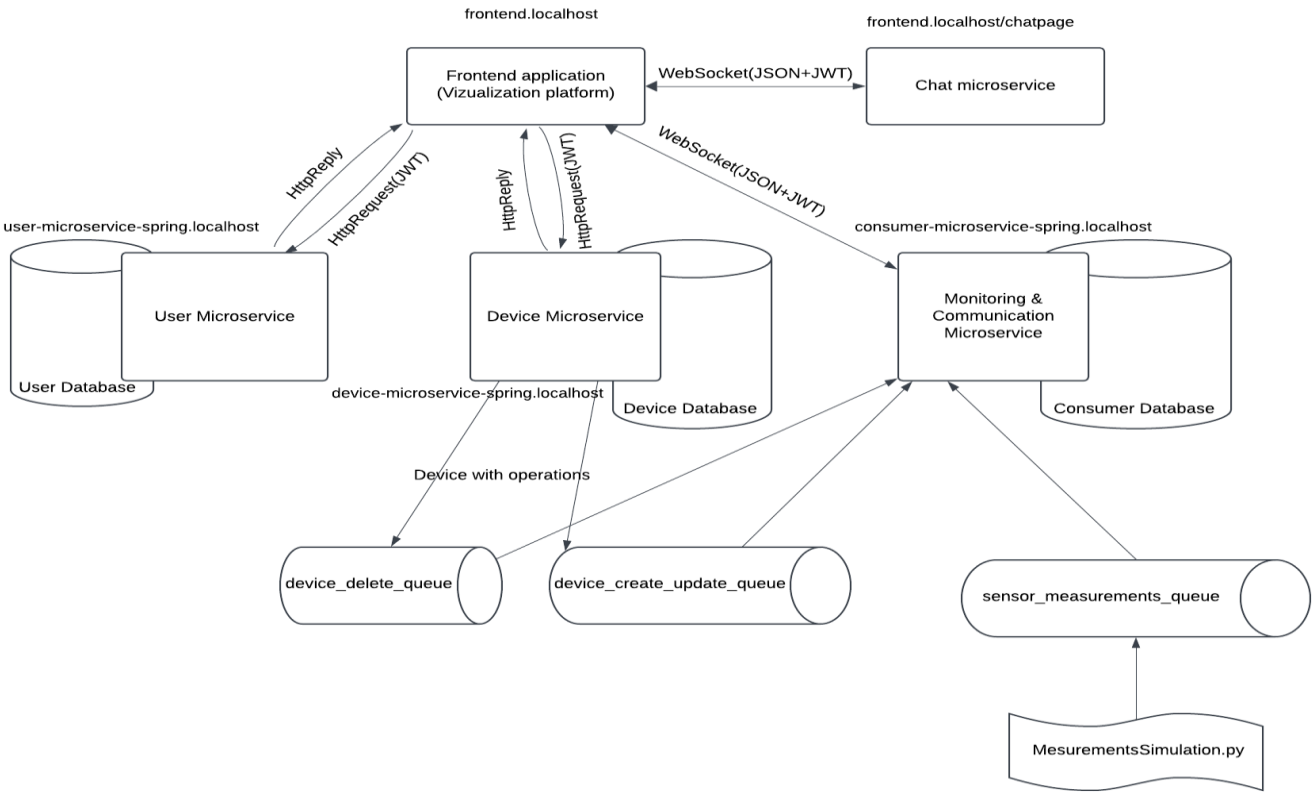
1. Trimiterea unui mesaj:
    - Utilizatorul introduce un mesaj în chat box, transmis asincron către administrator prin WebSocket.
2. Răspunsul administratorului:
    - Administratorul primește mesajul utilizatorului și trimite un răspuns, vizibil instant în interfața utilizatorului.
3. Notificare de citire:
    - Administratorul citește mesajul, iar utilizatorul primește confirmarea "mesaj citit".
4. Notificare de tastare:
    - Utilizatorul tastează un mesaj, iar administratorul vede notificarea vizuală despre starea de tastare.

Scenarii

1. Login și autentificare:
    - Utilizatorul introduce credentialele, serverul validează informațiile.
2. Accesarea chatului:
    - Utilizatorul accesează chatul, iar WebSocket-ul se inițializează folosind token-ul JWT pentru autentificare.
3. Comunicare în timp real:
    - Mesajele sunt transmise și recepționate instant între utilizatori și administratori prin WebSocket.

3. Proiectare

3.1 Arhitectura Conceptuală a Sistemului



Arhitectura sistemului este bazată pe un set de microservicii Spring Boot, fiecare cu o responsabilitate specifică, care comunică fie prin protocoale REST, fie prin mesaje asincrone utilizând RabbitMQ. Frontend-ul, dezvoltat în React, oferă o interfață intuitivă pentru gestionarea operațiunilor și comunicarea între utilizatori.

Microserviciul de Utilizatori (User Microservice) gestionează operațiunile CRUD pentru utilizatori și implementează mecanisme de autentificare și autorizare securizată prin JWT și Spring Security. De asemenea, validează toate cererile HTTP și handshake-urile WebSocket pentru a asigura securitatea.

Microserviciul de Chat este responsabil pentru conexiunile WebSocket și facilitează comunicarea bidirecțională în timp real. Mesajele sunt stocate într-o bază de date PostgreSQL, iar utilizatorii sunt notificați despre noi mesaje, starea de citire și tastare prin intermediul RabbitMQ.

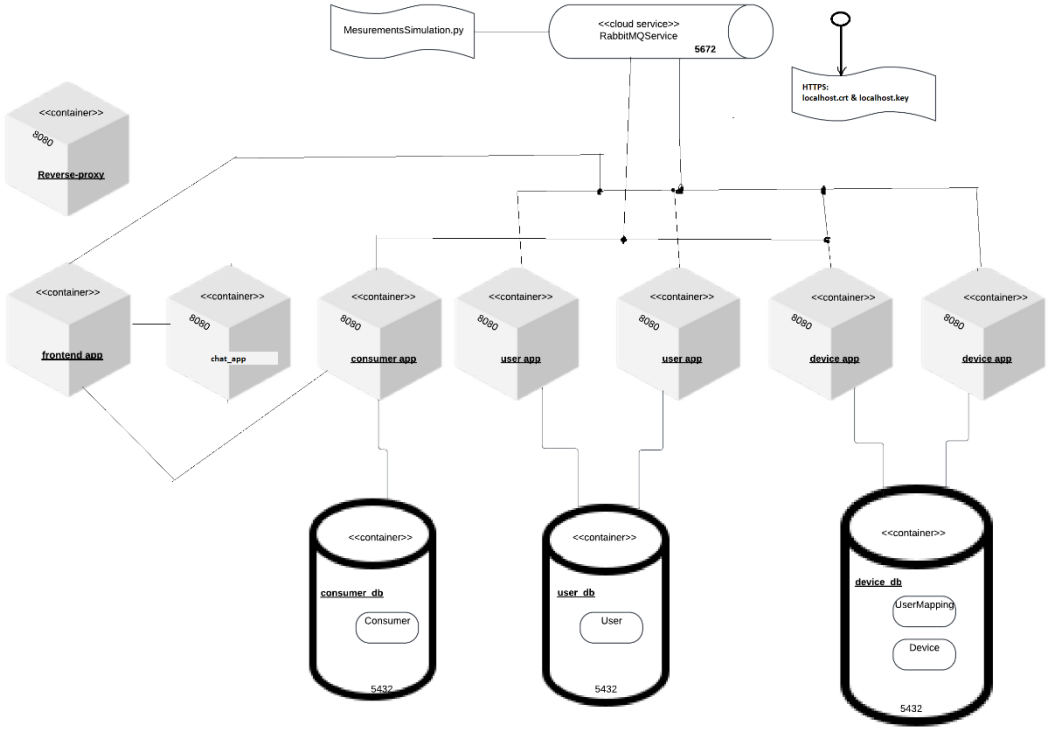
Microserviciul de Dispozitive (Device Microservice) oferă funcționalități CRUD pentru dispozitive și gestionează relațiile dintre utilizatori și dispozitivele lor. Orice modificare a dispozitivelor este comunicată către microserviciul de monitorizare.

Microserviciul de Monitorizare procesează datele despre consumul energetic al dispozitivelor și trimite alerte prin RabbitMQ dacă sunt depășite limitele de consum stabilite.

Frontend-ul React intermediază interacțiunile utilizatorilor prin pagini dedicate, cum ar fi cele pentru chat, administrarea dispozitivelor și autentificare. Comunicarea în timp real cu backend-ul este realizată prin SockJS și STOMP.

În centrul acestor interacțiuni, **RabbitMQ** acționează ca un middleware de mesagerie, sincronizând datele între microservicii și facilitând notificările în timp real. Această arhitectură modulară asigură scalabilitate, securitate și o experiență de utilizator optimă.

3.2 Diagrama de Deployment



Sistemul este implementat utilizând containere Docker, fiecare microserviciu având propriul container izolat. Reverse proxy-ul și load balancer-ul sunt gestionate de Traefik, care distribuie traficul între instanțele multiple ale microserviciilor.

Structura containerelor:

1. **Microservicii backend:**
   - Fiecare microserviciu rulează într-un container Docker bazat pe o imagine Spring Boot, configurată prin Dockerfile-uri specifice.
   - aBaza de date asociată fiecărui microserviciu rulează într-un container PostgreSQL.
2. **RabbitMQ:**
   - Rulează ca un container separat, configurat pentru a gestiona cozi pentru mesageria asincronă.
3. **Frontend React:**
   - Este găzduit într-un container Nginx, care servește aplicația către utilizatori.
4. **Traefik:**
   - Configurat ca reverse proxy, gestionează cererile HTTP și WebSocket, asigurând securitatea prin TLS.
5. **Simulatoare Python:**
   - Rulează în containere dedicate și generează date despre consumul energetic, trimise către RabbitMQ.

4. Implementare

Sistemul facilitează schimbul de mesaje în timp real printr-un flux bine definit între frontend, backend și RabbitMQ. Mesajele trimise sunt procesate de backend, publicate în RabbitMQ și livrate destinatarului prin WebSocket. Token-urile JWT sunt validate în timpul handshake-ului WebSocket, asigurând securitatea conexiunii.

Adăugiri Importante în Cod:

- AMQPConfig.java: Configurează cozi RabbitMQ și definește conversia mesajelor în format JSON.
- DeviceController.java: Integrează gestionarea dispozitivelor cu notificările de mesagerie.
- JwtValidator.java: Validează semnătura și expirarea token-urilor JWT pentru a asigura autenticitatea.

Scalabilitate

Sistemul este proiectat pentru a gestiona creșteri ale volumului de utilizatori:

- RabbitMQ: Utilizează cozi dedicate pentru fiecare utilizator, reducând coliziunile.
- Docker și Traefik: Permite replicarea microserviciilor pentru a gestiona volume mari de conexiuni WebSocket, cu distribuirea uniformă a traficului prin load balancing.

Securitate

Sistemul securizează comunicațiile prin:

- Validarea token-urilor JWT utilizând JwtValidator, verificând semnătura și expirarea acestora.
- Utilizarea HTTPS și autorizarea handshake-ului WebSocket pentru conexiuni sigure.

Instrucțiuni pentru Configurare și Depanare

- Configurarea certificatelor HTTPS în Traefik:
  - Generează un certificat self-signed:

```
openssl req -x509 -newkey rsa:2048 -keyout key.pem -out cert.pem -days 365
```

  - Configurează certificatul în traefik.yml pentru conexiuni sigure.
- Depanare:
  - Conexiuni WebSocket eșuate: Verificați validitatea token-urilor JWT și deschiderea porturilor necesare.

- Probleme RabbitMQ: Asigurați cozi corect configurate în RabbitMQ Management Console.

Fluxul unui Mesaj

1. Trimiterea mesajului:
   - Utilizatorul introduce mesajul în frontend, care este trimis prin WebSocket la /frontend/send.
   - Backend-ul salvează mesajul în baza de date și îl publică în RabbitMQ.
   - Destinatarul primește mesajul prin abonarea la /topic/newMessage.
2. Citirea mesajului:
   - Administratorul citește mesajul, iar frontend-ul notifică backend-ul prin /topic/read.
   - Backend-ul actualizează starea mesajului în baza de date și notifică expeditorul despre citire.

4.1 Instrumente Utilizate

Proiectul utilizează o suită de tehnologii moderne pentru a asigura funcționalitate, securitate și scalabilitate.

Backend:

Implementat în **Java** folosind **Spring Boot**, backend-ul oferă microservicii robuste. Autentificarea și autorizarea sunt gestionate prin **Spring Security** și **JWT**, cu validarea token-urilor realizată printr-un JwtValidator. Comunicarea în timp real este facilitată de **Spring WebSocket**, configurat în WebsocketConfig. Persistența datelor pentru utilizatori, mesaje și dispozitive este realizată prin **Spring Data JPA** cu baze de date PostgreSQL.

Frontend:

Frontend-ul construit în **React** oferă o interfață modernă și interactivă. Conexiunile WebSocket sunt gestionate prin **SockJS** și **STOMP**, iar design-ul este optimizat folosind **Material-UI** pentru componente responsive, cum ar fi tabelele și formularul de mesaje.

RabbitMQ:

Mesageria asincronă între microservicii este realizată cu **RabbitMQ**, integrat prin **Spring AMQP**. Configurația cozii și exchange-urilor este detaliată în AMQPConfig.

Orchestrare și Securitate:

Sistemul rulează complet containerizat în **Docker**, fiecare microserviciu având propriul container. **Traefik** este utilizat ca reverse proxy pentru load balancing și securitate prin TLS, configurat cu un certificat self-signed în traefik.yml. Rețelele și relațiile dintre containere sunt gestionate printr-un fișier docker-compose.

4.2 Fișier README

Aplicația finală poate fi rulată în Docker urmând pașii de mai jos:

1. Clonează repository-urile backend și frontend de pe acest repository principal: https://gitlab.com/budaandreea02/ds2024_30244_buda_andreea_assignment_3
2. Mută fișierul docker-compose.yml în directorul de lucru, același director unde au fost clonate anterior ambele repository-uri.
3. Deschide terminalul în acest director și pornește Docker Desktop pentru a activa daemon-ul Docker.
4. Construiește și rulează containerele utilizând comenzile: **docker-compose build, docker-compose up.**
5. Accesează aplicația în browser după ce containerele sunt pornite: https://frontend.localhost/

5. Bibliografie

[1] https://dsrl.eu/courses/sd/

[2] Lab Book: I. Salomie, T. Cioara, I. Anghel, T.Salomie, Distributed Computing and Systems: A practical approach, Albastra, Publish House, 2008, ISBN 978-973-650-234-7

[3] Lab Book: M. Antal, C. Pop, D. Moldovan, T. Petrican, C. Stan, I. Salomie, T. Cioara, I. Anghel, Distributed Systems – Laboratory Guide, Editura UTPRESS Cluj-Napoca, 2018 ISBN 978-606 737-329-5, 2018, https://biblioteca.utcluj.ro/files/carti-online-cu-coperta/329-5.pdf

[4] Spring Security, https://spring.io/projects/spring-security/