DISTRIBUTED SYSTEMS

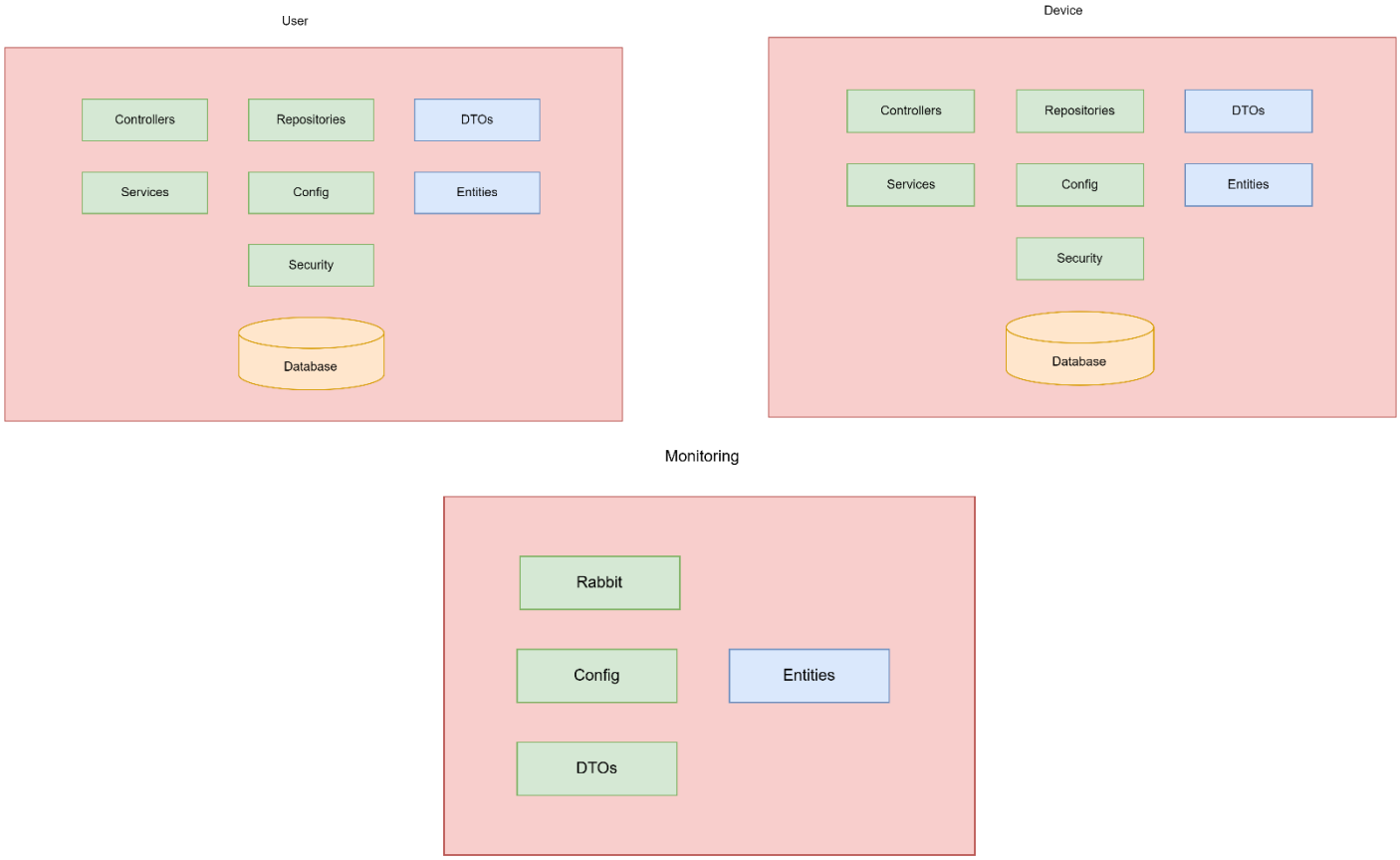
Assignment 2

Asynchronous Communication and Real-Time Notification

Moldovan Maria-Alexandra

30244

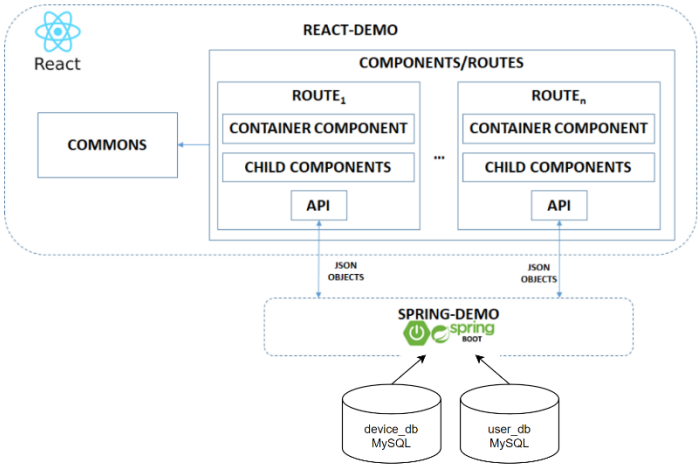
1. Project conceptual architecture - backend



Pe partea de backend, am creat trei microservicii: `device`, `user` și `monitoring`. Această diagramă reprezintă arhitectura layered.

- **`controllers`**: acest layer gestionează cererile HTTP(`GET`, `POST`, `PUT`, `DELETE`) venite din frontend și le direcționează spre serviciile corespunzătoare
- **`services`**: acest layer reprezintă logica de business, și gestionează regulile pentru operațiunile `CRUD` (`Create` – `Read` – `Update` – `Delete`) pe entitățile de `device` și `user`, respectiv. Serviciile interacționează cu `repositories` pentru a prelua sau stoca date în baza de date și creează `DTO`-uri (`Data Transfer Objects`) pentru entități, în funcție de cerințe
- **`config:` RabbitMQ Integration** *(pentru comunicare asincronă și notificări în timp real)*: Microserviciul **monitoring** ascultă mesajele venite din coada masuratori, unde sunt publicate măsurătorile de consum energetic de către producer. Dacă un device înregistrează un consum energetic mai mare decât limita maxHEC, **monitoring** trimite o alertă prin WebSocket către frontend.
- **`repositories`**: acest layer interacționează direct cu baza de date, iar faptul că extind interfața existentă `JpaRepository` asigură existența unor metode predefinite pentru operațuni CRUD, dar permit și definirea unor noi query-uri, de exemplu eu folosesc pentru autentificare `User findByNameAndPassword(String name, String password);`
- **`database`**: reprezintă stocarea principală a datelor sistemului, care reține datele despre `device`, `user` și `measurements`.

2. Project conceptual architecture - frontend



Pe partea de frontend, am creat un singur proiect în React, interacționând cu backendul folosind framework-ul Spring Boot.

Partea de react este accesibilă la `http://localhost:3000` și are diferite fișiere:

- **`commons`**: conține componente commune, precum felul în care îmi arată tabelele, host endpoints (de ex `hosts.js` include `device_backend_api: 'http://device.localhost '`)

- **`api`**: pentru fiecare microserviciu (`device` și `user și monitoring`), am o component de `api` care are endpointurile necesare, de ex:

```
const endpoint = {
    device: '/device'
};

function getDevices(callback) {
    let request = new Request(HOST.device_backend_api + endpoint.device, {
        method: 'GET',
    });
    console.log(request.url);
    RestApiClient.performRequest(request, callback);
}
```

- **`route_1, … n`**: fiecare rută corespunde unei pagini sau secțiuni, fiecare cu componentele container ale sale.

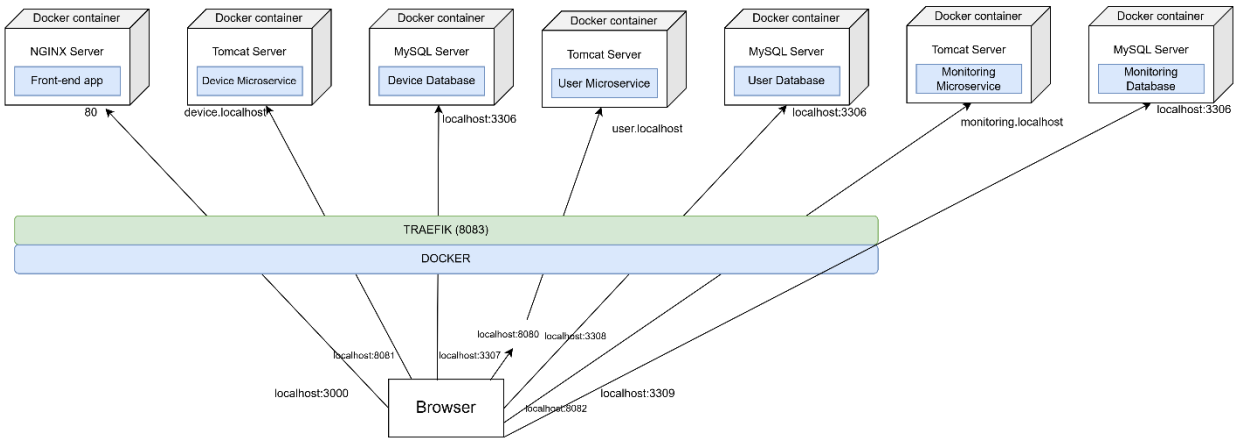
- WebSockets pentru notificări în timp real:

- Frontend-ul ascultă notificările WebSocket de la monitoring pe /topic/alerts.
- Dacă un device depășește consumul maxim, utilizatorul primește o notificare vizuală imediată.

Paginile mele disponibile sunt:

- `/` și `/login` duc spre pagina de `Login`
- `/admin` și `/admin-dashboard` duc spre pagina accesibilă doar administratorului, care îl duce spre `Device sau User Management`
- `/user` și `/device` duc spre paginile în care sunt implementate operațiunile CRUD pe aceste entități
- `/home` duce spre pagina de acasă, la care are acces clientul, momentan aceasta fiind singura funcționalitate a acestuia

3. Application Deployment Diagram



Pe partea de deploy, am folosit Docker, hub bazat pe containere. Cele folosite de mine sunt:

- **`NGINX server`**: acest container este responsabil cu rularea părții de front. Rulează pe portul 80 și este punctul de intrare pentru utilizatorii care accesează aplicația din browser pe `http://localhost:3000`
- **`Tomcat Server`** (pentru `Device`, respective `User` și `Chat` back-end app): aceste containere sunt responsabile cu rularea și gestionarea microserviciilor de device, user și chat. Ele rulează pe porturile 8081, respectiv 8080, 8082
- **`MySQL Server`** (pentru `Device`, respectiv `User` database): aceste containere conțin baza de date pentru device, respectiv user, și rulează pe portul 3306.
- **RabbitMQ Server**: gestionează comunicarea asincronă între device, producer și monitoring, dar e în Cloud

Pentru persistența datelor, am creat și volume, în care rămân stocate datele din bazele de date ale microserviciilor: