UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

SISTEME DISTRIBUITE

Energy Management System

Balint Cătălin 30642/1

Facultatea de Automatica
si Calculatoare

2024

Cuprins

1 Assignment1

1.1 Descriere generală

Aplicația este o platformă web dedicată gestionării utilizatorilor și dispozitivelor asociate sistemului de măsurare inteligentă a energiei, oferind funcționalități pentru înregistrare, autentificare și acces la diverse pagini în funcție de rolul utilizatorului.

Administratorul are acces la pagina de administrare , unde poate vizualiza toți utilizatorii aplicației și dispozitivele acestora sub formă de tabele si poate efectua operatiile CRUD.

Utilizatorul are acces restricționat, fără autorizarea de a accesa pagina de admin, și poate vizualiza doar propriile dispozitive pe pagina dedicată utilizatorului standard.

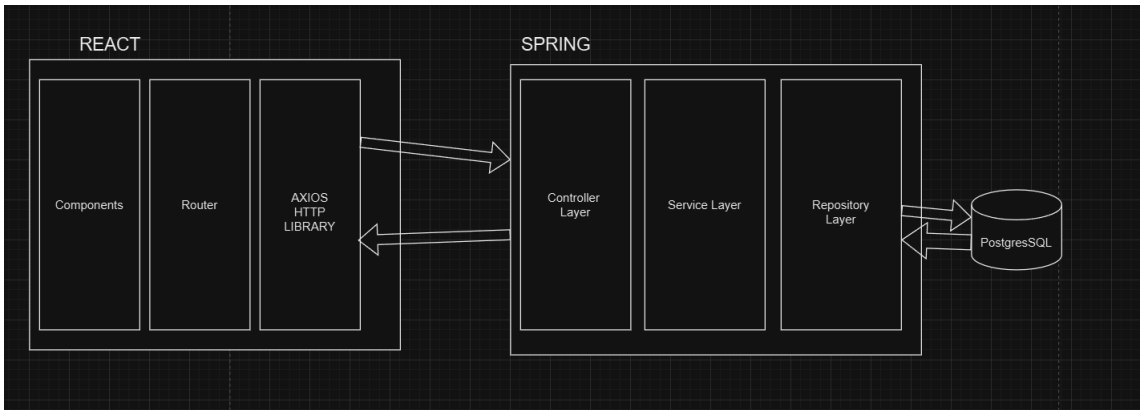
1.2 Structura aplicatiei

Aplicația este compusă dintr-un **frontend** care include un sistem de autentificare pentru utilizatori. După logare, utilizatorii sunt redirecționați în funcție de roluri către paginile `/admin` sau `/persondevice`. Fiecare utilizator poate accesa pagina dedicată, unde își poate vizualiza dispozitivele asociate.

Administratorul are capacitatea de a efectua operațiuni CRUD (Creare, Citire, Actualizare, Ștergere) atât pe tabela dispozitivelor, cât și pe cea a utilizatorilor. Acesta poate adăuga noi utilizatori și poate asocia dispozitivele cu utilizatorii în funcție de ID, conectând astfel două baze de date.

Backend-ul aplicației este construit pe Spring Boot, oferind endpoint-uri REST pentru autentificare, adăugare, inserare, ștergere, vizualizare și verificarea câmpurilor, cum ar fi validarea rolului de administrator.

Conexiunea cu baza de date permite stocarea și gestionarea utilizatorilor, asigurând o administrare eficientă a acestora.



1.3 Conexiunea la baza de date

Conexiunea la baza de date este gestionată prin fișierul de configurare `application.properties`. Parametrii de configurare definiți sunt următorii:

- `database.ip`: Aceasta este adresa IP a serverului unde este găzduită baza de date.
- `database.port`: Este portul pe care serverul PostgreSQL ascultă cererile. Valoarea implicită este `5432`.
- `database.user`: Utilizatorul implicit este `postgres`.
- `database.password`: Este parola asociată contului utilizatorului .

- `database.name`: Numele bazei de date la care aplicația se va conecta. În cazul nostru, avem `persons-db` pentru microserviciul de management al utilizatorilor și `devices-db` pentru managementul dispozitivelor.

1.4 Login

Clientul trimite o cerere `POST` la `/auth/login` . `PersonService` caută utilizatorul în baza de date. Dacă utilizatorul există, se verifică dacă parola introdusă corespunde cu cea stocată.

În funcție de rezultatul login-ului, se verifică rolul utilizatorului folosind o cerere `GET` la `/auth/isAdmin`. În funcție de rol, utilizatorul este redirecționat către pagina corespunzătoare. De asemenea, rolul utilizatorului este salvat în `localStorage` în aplicația React, permițând astfel implementarea accesului restricționat pe baza rolului utilizatorului.

1.5 Sincronizare

Pentru simplificare, am duplicat datele legate de utilizatori (ID-urile) din tabela `Person` din microserviciul de utilizatori prin crearea unei entități noi, `PersonReference`, în microserviciul de management al dispozitivelor. Aceasta va conține toate ID-urile aferente utilizatorilor.

Atunci când administratorul adaugă un utilizator nou în frontend, se va trimite o cerere `POST` la `/person-references` din microserviciul de management al dispozitivelor pentru a adăuga ID-ul utilizatorului. Similar, atunci când administratorul șterge un utilizator din baza de date, se va trimite o cerere `DELETE` pentru a șterge ID-ul utilizatorului din `personReferences`.

Este definită o relație de tip Many to One între `Device` și `PersonReference`, fiecare utilizator având posibilitatea de a avea o listă cu multiple dispozitive.

1.6 Arhitectura conceptuală

Aplicația urmează o arhitectură pe layere, incluzând layerele `Controller`, `Service` și `Repository`. Fiecare microserviciu — unul pentru utilizatori (`persons-microservice`) și unul pentru dispozitive (`devices-microservice`) — implementează această structură și are propria bază de date.

- **Controller**: gestionează cererile HTTP și definește endpoint-urile.
- **Service**: conține logica de business și gestionează sincronizarea datelor între microservicii.
- **Repository**: oferă acces la baza de date pentru operațiuni CRUD.

Structura modulară asigură scalabilitatea și independența fiecărui microserviciu.

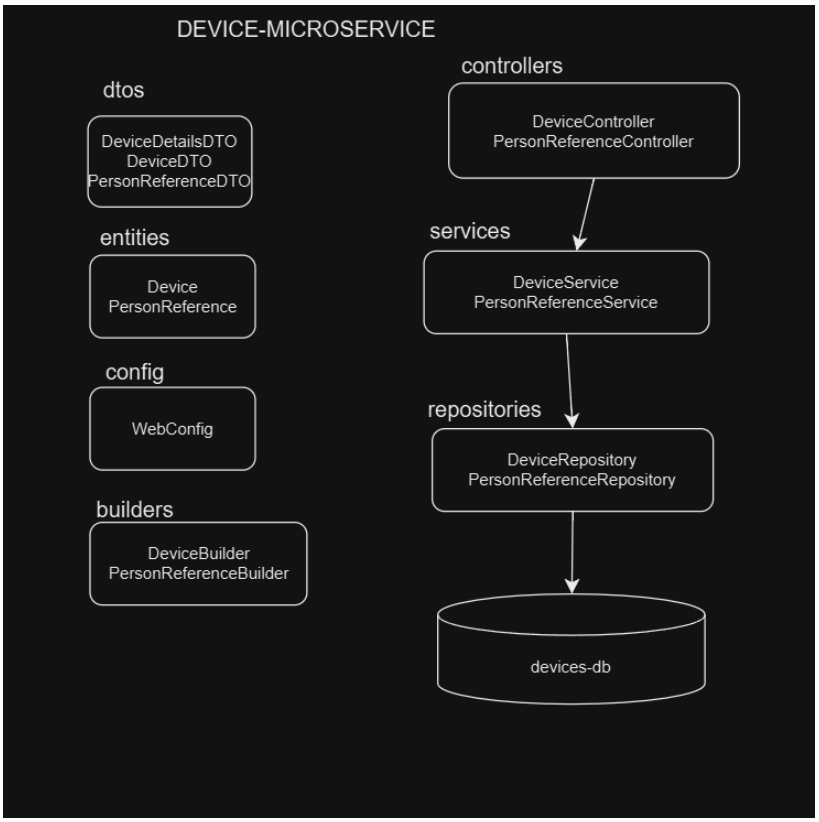


Figura 1: Device Microservice

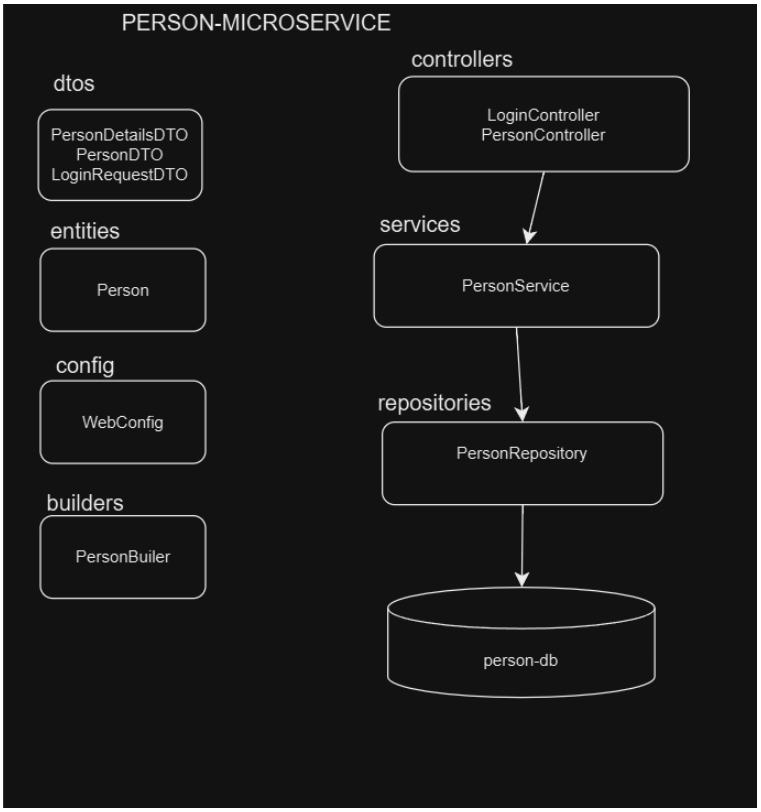
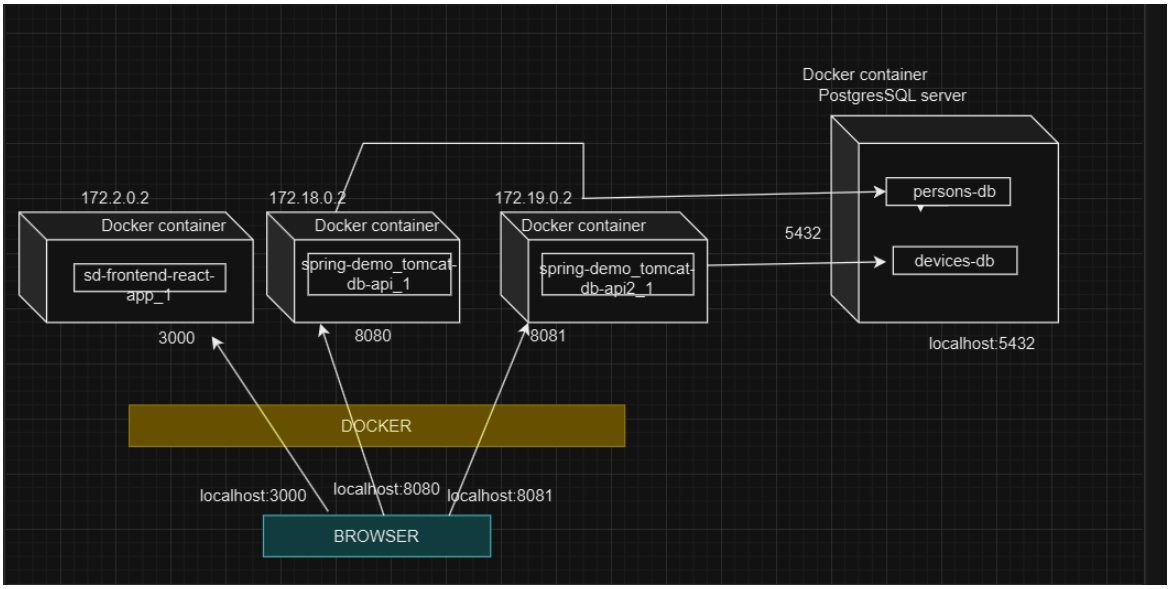


Figura 2: Person Microservice

1.7 Diagrama UML Deployment

- **FRONTEND (React - `sd-frontend`)**: Rulează într-un container Docker separat, accesibil utilizatorului la adresa `localhost:3000` prin intermediul browserului.
- **BACKEND - Microservicii**:
  - `persons-microservice`: Rulează pe portul `8080` și poate fi accesat de browser la adresa `localhost:8080`.
  - `devices-microservice`: Rulează pe portul `8081`, accesibil de asemenea în browser la adresa `localhost:8081`.
- **BAZA DE DATE (PostgreSQL)**: Rulează într-un container Docker dedicat și este accesibilă pe portul `5432`. Aceasta conține două baze de date, `persons-db` și `devices`.



2 Assignment2

2.1 Descriere generala

Proiectul constă într-un Microserviciu de Monitorizare și Comunicare pentru un Sistem de Management al Energiei. Microserviciul folosește RabbitMQ pentru a colecta date de la dispozitive de măsurare inteligente și pentru a calcula consumul orar de energie, stocând aceste date în baza sa de date.

2.2 Structura

Message Producer - Simulatorul
Simulatorul este o aplicație standalone care funcționează ca producător de mesaje. Acesta citește valori de consum energetic dintr-un fișier CSV (`sensor.csv`) și, la fiecare 10 minute, trimite date către RabbitMQ sub forma unui mesaj JSON ce conține: timestamp, device_id și measurement_value.

Message Broker - Intermediarul de Mesaje
RabbitMQ acționează ca un broker de mesaje care gestionează coada `sensor_data_queue`. Mesajele de la simulator sunt colectate aici și livrate către Microserviciul de Monitorizare pentru procesare.

Message Consumer
Microserviciul preia mesajele din coadă, agregă valorile primite şi le stochează în baza de date. Dacă valoarea consumului depășește limita, microserviciul trimite notificări asincrone utilizatorului printr-un WebSocket.

2.3 Sincronizarea între Microservicii

Sincronizarea între baza de date a microserviciului *Device Management* și noul microserviciu de *Monitoring și Comunicare* se realizează printr-un sistem bazat pe evenimente care utilizează o coadă de mesaje. Astfel, la fiecare schimbare în tabelul de dispozitive (*Device Tables*)—fie adăugare, actualizare sau ștergere—se trimite automat un mesaj conținând informațiile actualizate ale dispozitivului către coada `device_queue` din RabbitMQ. Aceasta coadă transmite mai departe datele către microserviciul *Monitoring și Comunicare* pentru procesare .

Clasa `DeviceEventProducer` , definită în microserviciul *Device Management*, trimite evenimentele de schimbare a stării dispozitivelor folosind `RabbitTemplate`. Se trimit datele dispozitivului către `device-exchange` cu cheia de rutare `device-routing-key`.

2.4 Configurația RabbitMQ

Pentru sincronizarea între Device Microservice și Monitoring and Communication Microservice

- Se definește coada `device_queue`, configurată să fie persistentă.
- Se definește un schimbător de mesaje de tip *DirectExchange*, denumit `device-exchange`.
- Se realizează legarea (*binding*) între coada `device_queue` și `device-exchange`, folosind cheia de rutare `device-routing-key`.

Pentru comunicarea între Messsage Producer(Simualtor-standalone application) și Message Broker(componenta a noului microserviciu)

- Fiecare mesaj JSON generat de simulator este trimis la coada `sensor_data_queue` folosind metoda `basicPublish()`.
- Parametri pentru `basicPublish`:
  - `exchange`: lăsat gol (`""`), indicând că mesajul va fi trimis direct către coadă.
  - `QUEUE_NAME`: numele cozii RabbitMQ (adică, `sensor_data_queue`).
  - `message`: Mesajul format în JSON, trimis ca șir de caractere către RabbitMQ.

2.5 WebSockets

Pentru transmiterea notificărilor în timp real către utilizatori, sistemul implementează **WebSockets** folosind Spring Boot şi protocolul STOMP. Această tehnologie permite comunicarea bidirecţională între server şi client. În acest caz, notificările sunt trimise utilizatorilor atunci când valoarea de măsurare a unui dispozitiv depășește valoarea sa de prag maxim .

Fiecare mesaj de notificare este construit în format JSON. Mesajul este publicat pe un canal de notificare dedicat pentru fiecare utilizator, sub forma `/topic/notifications/{personId}`, asigurând că notificările sunt personalizate pentru fiecare utilizator în funcţie de ID-ul său.

Pe partea de client, conexiunea WebSocket este iniţializată şi gestionată printr-un abonament STOMP folosind biblioteca `SockJS`. Fiecare utilizator se conectează la un topic dedicat notificărilor sale.

2.6 Chart

Pentru a monitoriza consumul de energie al dispozitivelor conectate, aplicația include un grafic interactiv, care prezintă consumul total de energie al fiecărui client în funcție de ora din zi. Utilizatorii pot selecta o anumită dată, iar sistemul va prelua și afișa consumul energetic cumulat pentru toate dispozitivele unui client pe parcursul zilei respective. Date sunt procesate și afișate într-un grafic de tip `line`.

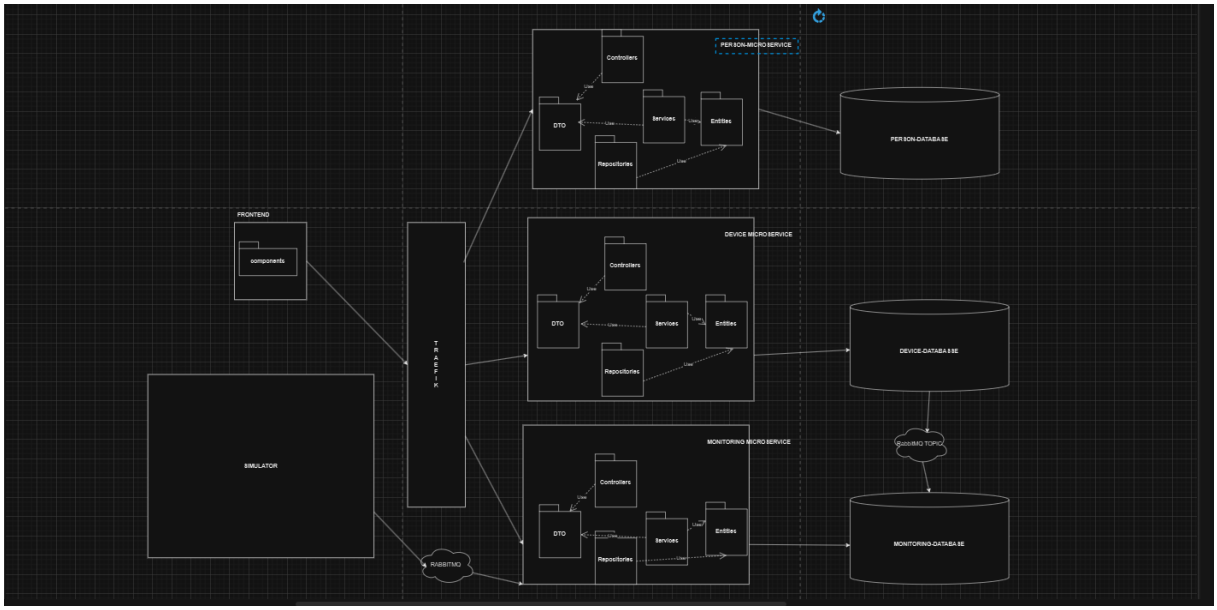
Graficul este desenat într-un element HTML de tip `canvas`. Pentru a preveni supraîncărcarea memoriei prin grafice multiple, un grafic existent este distrus înainte de a crea unul nou, folosind o referință pentru a păstra instanța actuală a graficului.

2.7 Traefik

Traefik este un load balancer modern și un proxy revers utilizat pentru a simplifica gestionarea traficului într-un mediu de microservicii. În contextul proiectului, Traefik este configurat pentru a gestiona cererile HTTP și WebSocket, redirecționându-le către microserviciile corespunzătoare pe baza regulilor definite. Datorită integrării native cu RabbitMQ, Docker, Traefik poate detecta automat schimbările de infrastructură și reconfigura dynamic rutele de trafic.

2.8 Diagrama Conceptuală

Diagrama conceptuală reprezintă o imagine de ansamblu asupra arhitecturii sistemului și a interacțiunilor dintre componentele acestuia. Sistemul este compus din trei microservicii: *Person*, *Device* și *Monitoring*, fiecare având propria bază de date PostgreSQL pentru gestionarea datelor. Microserviciile comunică între ele prin intermediul unui sistem de mesagerie RabbitMQ, utilizând cozi dedicate și un topic principal pentru sincronizare. Simulatorul standalone generează date despre consumul energetic al dispozitivelor și le trimite în format JSON către RabbitMQ, de unde sunt preluate de microserviciul *Monitoring*.



2.9 Diagrama Deployment

Diagrama de deployment prezintă modul în care componentele aplicației sunt distribuite în mediu și interconectate. Fiecare microserviciu (*Person*, *Device* și *Monitoring*) rulează în containere Docker individuale, împreună cu bazele lor de date PostgreSQL, fiecare container fiind izolat pentru a asigura independența componentelor. RabbitMQ este implementat într-un container separat pentru a gestiona comunicarea asincronă dintre microservicii și simulator. În plus, un container dedicat găzduiește Traefik, utilizat pentru routing și load balancing, gestionând traficul dintre frontend-ul React și microservicii. Toate aceste containere sunt interconectate într-o rețea Docker unificată, denumită my $_{n}etwork1$.