DISTRIBUTED SYSTEMS
ASSIGNMENT 1

REQUEST-REPLY COMMUNICATION

Student: Deac Dan Cristian

Faculty of Automation and Computer science

Group: 30443

CONTENTS

SPECIFICATIONS

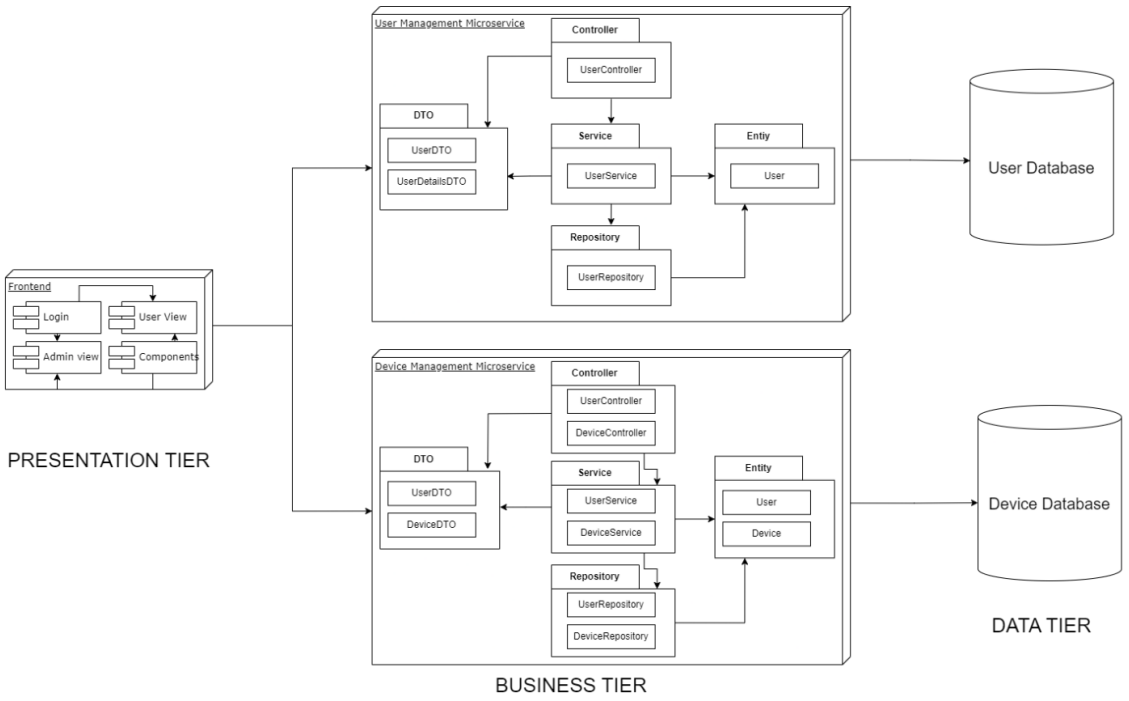
This application represents an Energy Management System that comprises a frontend and two microservices created to oversee users and their linked smart energy measuring devices. After a login process, the system can be accessed by two categories of users: supervisor (manager), and customers. The supervisor can execute CRUD (Create-Read-Update-Delete) actions on user accounts (defined by ID, name, role: admin/client), smart energy measuring devices (defined by ID, description, location, maximum hourly energy utilization), and on the linking of users to devices (each user can possess one or more smart devices in various places).

KEY CHARACTERISTICS:

- User login is required. Users are redirected to a suitable page according to their role.
- Admin role
  - CRUD operations on users
  - CRUD operations on devices
  - Creating user-device associations
- User/Client role
  - Can view their page with all linked devices
- Users in one role are prohibited from accessing the pages of other roles (e.g., by logging in and then copying the admin URL in the browser)

ARCHITECTURE

This application consists of a React front-end and two distinct microservices implemented in Spring REST. These microservices manage different aspects of the application: User Management and Device Management. The architecture employs separate databases for each microservice, emphasizing the organization of the User Management microservice with a 'user_db' database housing the 'users' table, and the Device Management microservice functioning with a 'devices_db' database containing 'devices' and 'users' tables.



FRONT-END: REACT + JAVASCRIPT

The front-end of the application is developed using React, a modern JavaScript framework recognized for its simplicity and adaptability. React allows for the development of interactive and user-friendly interfaces, enabling smooth interactions between users and the application. Its component-based structure simplifies the development process and improves maintainability.

BACK-END MICROSERVICES

USER MANAGEMENT MICROSERVICE

The User Management microservice is a foundational component of the application. It manages all user-related functionalities. This microservice is built using Spring REST, a robust and versatile framework for constructing scalable web services.

DATABASE:

- user_db

TABLES:

- users:

DEVICE MANAGEMENT MICROSERVICE

The Device Management microservice is responsible for overseeing devices within the application. It is also developed using Spring REST.

DATABASE NAME:

- devices_db

TABLES:

- device: Stores device-specific information.
- users: Contains a replicated version of the 'users' table from the User Management microservice to support device-related functionalities that require user data.

The decision to replicate the 'users' table in the Device Management microservice's database aims to enhance performance and reduce dependencies between microservices. It enables the Device Management microservice to efficiently handle device operations without continuous communication with the User Management microservice.

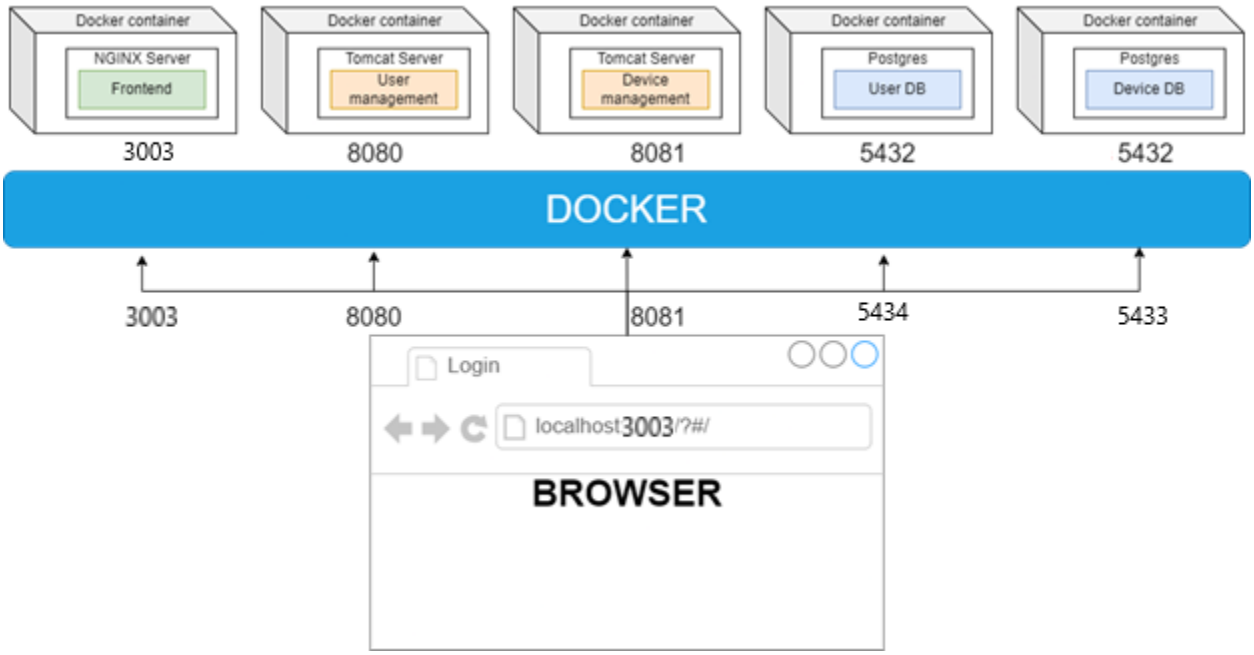
COMMUNICATION BETWEEN MICROSERVICES

Microservices communicate with each other through well-defined RESTful APIs, facilitating a loosely coupled and modular structure. Communication primarily occurs through HTTP requests, ensuring interoperability and enabling each microservice to operate independently while fulfilling its designated functionality.

The architecture of the full-stack application guarantees an organized and efficient approach to managing users and devices. With React on the front-end, the application offers an intuitive user interface. Meanwhile, the Spring REST-based microservices and their corresponding databases facilitate the efficient handling of user and device-related operations, creating a scalable and modular system that encourages flexibility and maintainability. The division of responsibilities between the two microservices, coupled with the replication of user data in the Device Management microservice, optimizes performance and enhances the overall functionality of the application.

DEPLOYMENT IN DOCKER

The deployment strategy for the full-stack application involves Docker, a containerization platform that enables the encapsulation of each application component into separate containers. This chapter details the deployment configuration for the React front-end, User Management microservice, Device Management microservice, and their respective databases, which are isolated within individual Docker containers. These containers expose internal ports to enable interaction with external systems, such as a web browser for the front-end.



CONTAINERIZATION CONFIGURATION

FRONT-END CONTAINER (REACT)

- The React front-end application is enclosed in a Docker container. The container encompasses all essential front-end resources, including HTML, CSS, JavaScript, and any other static assets. It utilizes NGINX as the web server.
- Docker Container Name: frontend
- Internal Docker Port: 3000
- Exposed Port: 3003
- The internal port 3000 within the Docker container is made accessible externally on port 3003, allowing interaction with the application through a web browser.

USER MANAGEMENT MICROSERVICE CONTAINER

- The User Management microservice, constructed using Spring REST, is containerized for deployment.
- Docker Container Name: users-microservice
- Internal Docker Port: 8080
- Exposed Port: 8080
- The internal port 8081 of the User Management microservice container is exposed directly, facilitating communication with other components and external systems.

DEVICE MANAGEMENT MICROSERVICE CONTAINER

- Similar to the User Management microservice, the Device Management microservice is also containerized.
- Docker Container Name: devices-microservice
- Internal Docker Port: 8081
- Exposed Port: 8081
- The internal port 8081 of the Device Management microservice container is exposed for interaction and integration with other system components.

USER DATABASE CONTAINER

- The User Management microservice's database, 'user_db,' is contained within its designated Docker container.
- Docker Container Name: postgres
- Internal Docker Port: 5432
- Exposed Port: 5434
- The internal port 5432, where the user database operates, is exposed to the external system for accessing user-related data.

DEVICE DATABASE CONTAINER

- The database for the Device Management microservice, 'devices_db,' is housed within its respective Docker container.
- Docker Container Name: postgres
- Internal Docker Port: 5432
- Exposed Port: 5433
- The internal port 5432, dedicated to device database operations, is exposed to facilitate communication and data management for devices.

DOCKER SUMMARY

The exposed ports of the Docker containers enable interaction with the application components through a standard web browser. Users can access the front-end application hosted on the exposed port 3003, which then communicates with the user and device microservices through their respective exposed ports (8080 and 8081) to perform user and device-related operations.

CONCLUSION

The deployment of the application components within Docker containers ensures encapsulation and isolation, enhancing portability, scalability, and consistency across various environments. Each component, including the front-end, microservices, and databases, is housed within its designated container, facilitating interaction via exposed ports for seamless communication and operation. The configuration allows external systems, such as web browsers, to interact with the application's front-end, initiating requests to the user and device microservices through their exposed ports, thereby enabling a fully functional and integrated system.