DISTRIBUTED SYSTEMS

Assignment 1

Request-Reply

Communication

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 an Energy Management System that consists of a frontend and two microservices designed to manager users and their associated smart energy metering devices. The system can be accessed by two types of users after a login process: administrator (manager) and clients.

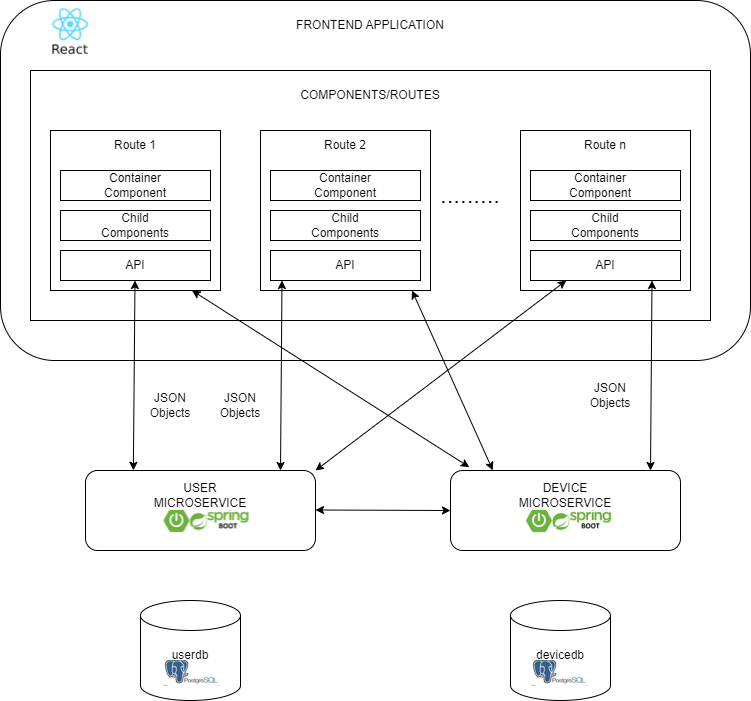
The administrator can perform CRUD (Create-Read-Update-Delete) operations on user accounts (defined by ID, name, role: admin/client), smart energy metering devices (defined by ID, description, address, maximum hourly energy consumption) and on the mapping of users to devices (each user can own one or more smart devices in different locations).

1. **Requirements**

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

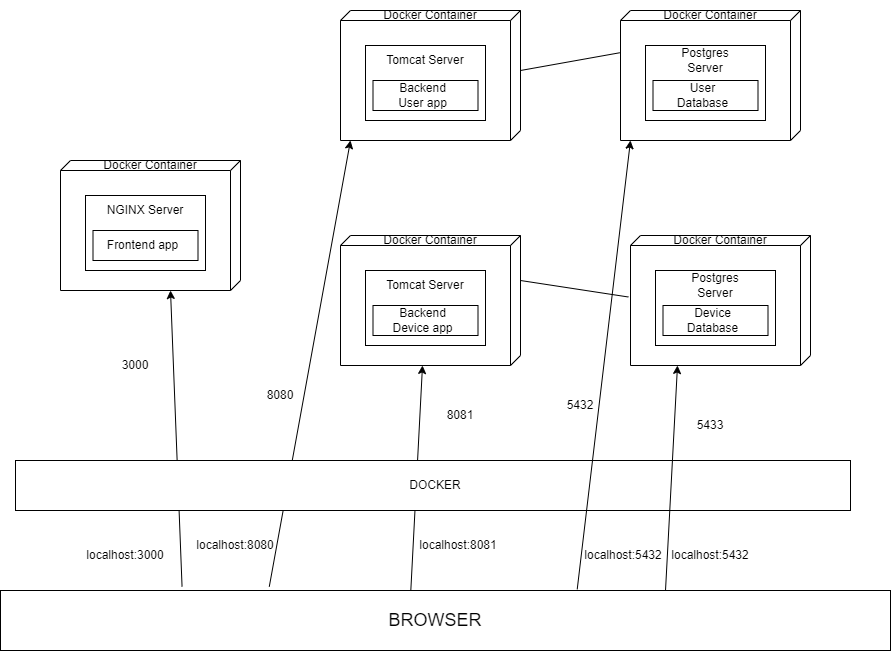
* **Functional requirements**
* Users can log in the application. They are redirected to the page corresponding to their role
* Users having the administrator/manager role can:
* Perform CRUD operations on user accounts
* Perform CRUD operations on devices
* Create mapping user-device
* Users having the client role can:
* View on their page all the devices associated to them
* The users corresponding to one role will not be able to access the pages corresponding to other roles (e.g. by log-in and then copy-paste the admin URL in the browser)
* **Non-functional requirements**
* The application should be divided into two microservices: User Management Microservice and Device Management Microservice
* 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)

1. **Conceptual architecture of the distributed system**



* **Frontend application**
* Each module defined for each route consists of a component container, a set of child components and an API used to send specific HTTP requests to the user microservice or device microservice applications. The communication between the React app and Spring Boot apps is done through JSON (JavaScript Object Notation) objects. JSON is a syntax which can be used to store and exchange data.
* The application is divided in the following packages:
* User package: contains JavaScript and CSS classes related to the user microservice. Inside these classes, the GET users API call to the user microservice is performed for fetching the existent user accounts. In addition, API calls for deleting, updating and creating users are also called in these classes and the received response is processed in order to correctly manipulate the data.
* Device package: contains JavaScript and CSS classes related to the device microservice. Inside these classes, the GET users API call to the device microservice is performed for fetching the existent devices. In addition, API calls for deleting, updating and creating devices are also called in these classes and the received response is processed in order to correctly manipulate the data.
* The App.js file is the one managing the log in process by calling the log in API to the user microservice and then, based on the response, the redirection to the corresponding pages is made.
* **User Microservice**
* This microservice is the one managing the user accounts and the log in process. It is divided in the following packages:
* Model package: contains the User entity class (defined by ID, name, username, password, role) and the Role (which is an enum class) and which is used to define the role of a user
* Dtos package: The role of a DTO (Data Transfer Object) is to transfer data between software application subsystems, especially in cases where those subsystems have different data structures. DTOs are often used to encapsulate and transport data between layers of an application, such as between the backend and frontend, or between different components within the same system. This package defines classes of different dtos for user accounts. Also, it contains a package “builder” that contains a class in which the transformation between dto to entity and vice versa is handeled.
* Repository package: contains the interface UserRepository that extends Crud Repository, a repository that helps programmers perform CRUD operations on entities
* Service package: with the help of repository package, this package is responsible for encapsulating and providing business logic, application-specific functionality, and a well-defined set of operations
* Controller package: this package contains classes that receive and handle incoming requests from the user interface or external systems
* Config package: is the package in charge of generating tokens when a user tries to log in and validate that tokens, confirm whether the credentials of the users are correct and check if thi have the corresponding authority
* Auth package: defines a classes for authentication request and response, an authentication service where the log in method is implemented and the authentication controller.
* ErrorHandler package: contains a class used for error handling
* **Device Microservice**
* This microservice is the one managing the devices. It is divided in the following packages:
* Model package: contains entity classes for Devices and User
* Dtos package: The role of a DTO (Data Transfer Object) is to transfer data between software application subsystems, especially in cases where those subsystems have different data structures. DTOs are often used to encapsulate and transport data between layers of an application, such as between the backend and frontend, or between different components within the same system. This package defines classes of different dtos for devices. Also, it contains a package “builder” that contains a class in which the transformation between dto to entity and vice versa is handeled.
* Repository package: contains the interfaces DeviceRepository, UserRepository that extend Crud Repository, a repository that helps programmers perform CRUD operations on entities
* Service package: with the help of repository package, this package is responsible for encapsulating and providing business logic, application-specific functionality, and a well-defined set of operations
* Controller package: this package contains classes that receive and handle incoming requests from the user interface or external systems
* Security package: this package is in charge of validating the tokens generated at authentication
* ErrorHandler package: contains a class used for error handling
* **Databases:** Two databases are defined for this application: userdb (corresponding to user accouns) and devicedb (corresponding to devices)

1. **UML Deployment Diagram**



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

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

• Docker container for backend application – runs two TOMCAT servers and maps local port 8080 to host computer port 8080 and local post 8081 to host computer port 8081

• Database container for database server – runs two Postgres servers and maps 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 and 8081

• 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 remain functionalities can also be tested
* 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 three 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 remain functionalities can also be tested
* 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