Energy Management System

Request-Reply Communication

Malina Lucacel

Contents

1. Introduction

1.1 Main objective

The objective of this project is to develop an Energy Management System (EMS) that facilitates efficient user and device management in a distributed architecture. This system includes a frontend interface and two microservices that handle user accounts and smart energy metering devices. The EMS aims to provide two types of access: administrator (manager) and client. Administrators are equipped with the tools to manage users and devices through CRUD operations, as well as assign devices to users. Clients can view their assigned devices, with role-based security measures in place to prevent unauthorized access to administrative functions.

1.2 Scope

The EMS system is designed to support essential management functions for energy metering devices across multiple user roles within a distributed architecture. Administrators can perform CRUD operations on both users and devices and manage user-device associations. Clients, with limited permissions, can view their assigned devices. The system enforces role-based access restrictions to ensure that users can only access pages relevant to their roles, enhancing security and maintaining data integrity.

1.3 Technologies Used

The EMS project leverages a microservices architecture, utilizing **Java Spring REST** for backend services and **React with TypeScript** for the frontend. React TypeScript integration provides type safety and enhanced development efficiency, while Spring REST facilitates reliable data handling and backend operations. **Session storage** in React is used for managing session data, providing a lightweight, secure approach to maintain role-based access control and ensuring users can only access authorized pages.

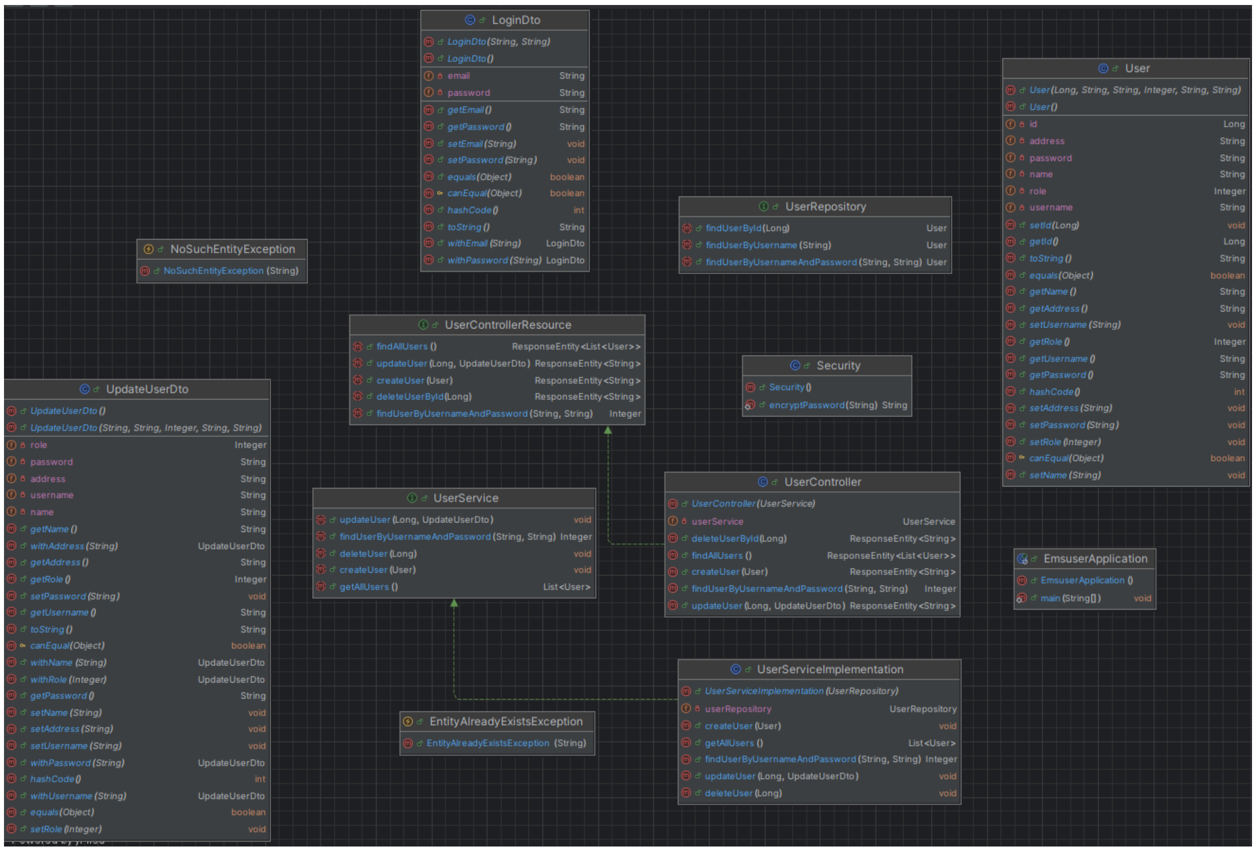
2. System Architecture

2.1 UML Diagrams

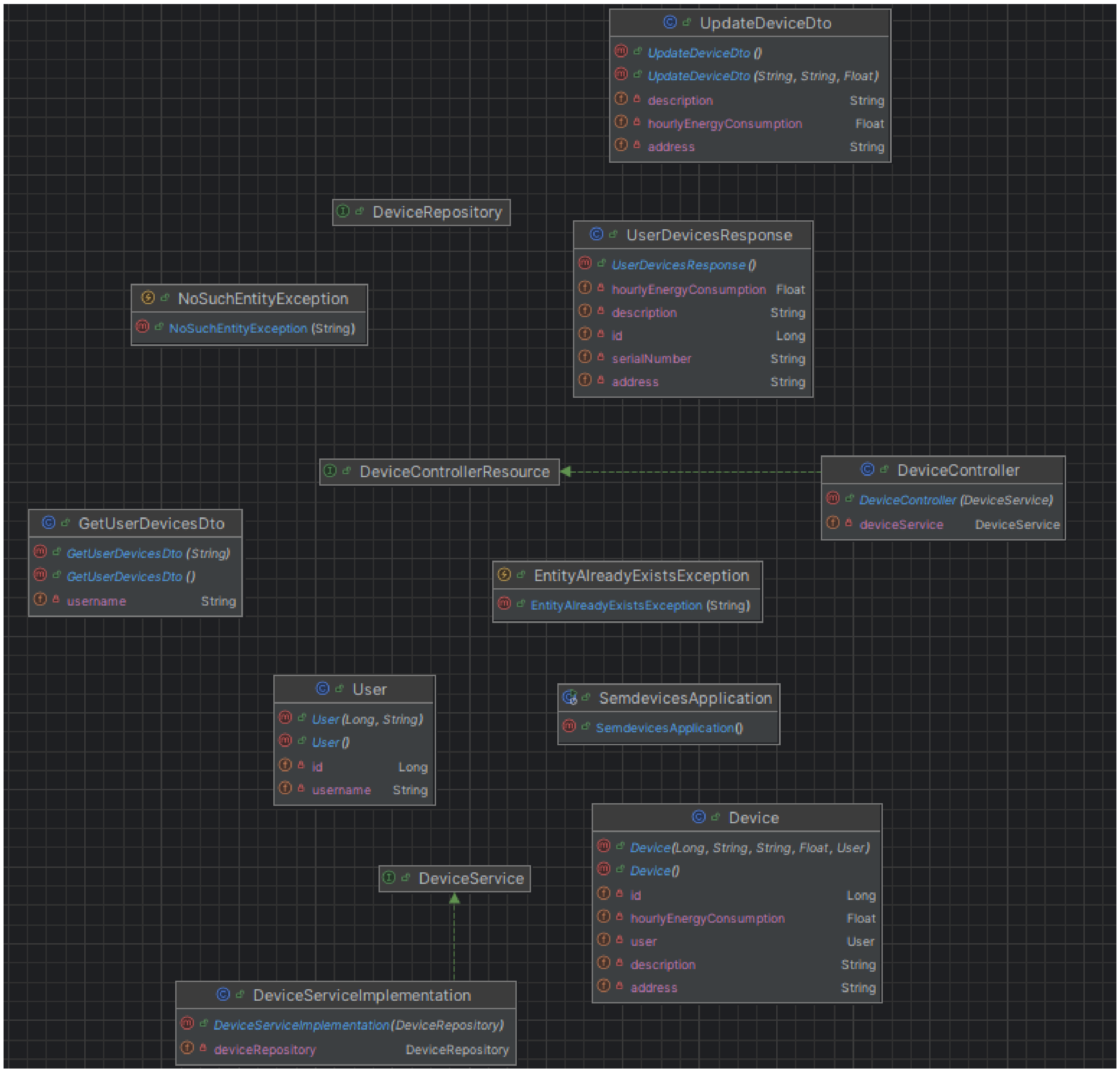
UML (Unified Modeling Language) diagrams are standardized visual representations used in software engineering to design, describe, and document the structure and behavior of systems. They help illustrate relationships and interactions between system components, making complex architectures more understandable.

Taking into consideration that the backend has 2 microservices, we have 2 Class Diagrams.

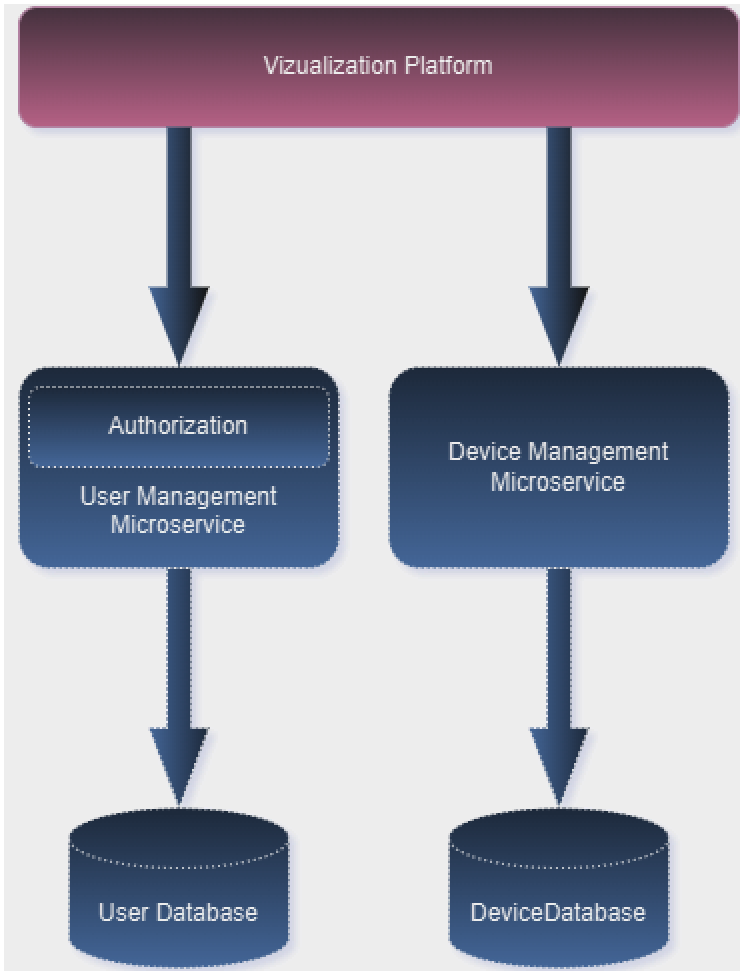
2.1.1 User Microservice Diagram



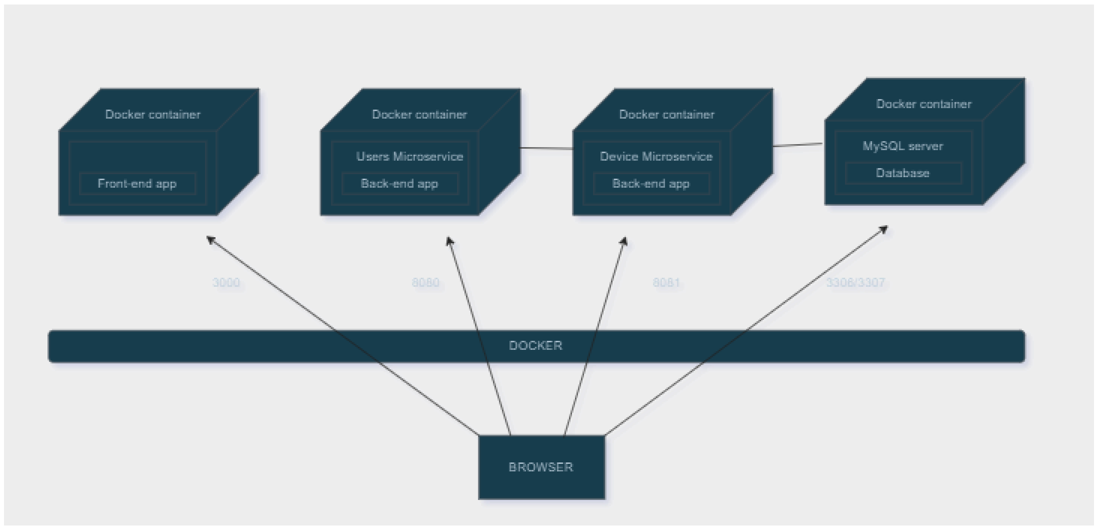
2.1.2 Device Microservice Diagram



2.1.4 Conceptual Architecture of the system



2.1.5 Deployment Architecture



3. Functional Requirements

Functional requirements specify the key features and behaviors the system must support. These ensure the EMS achieves its core purpose of user and device management.

- **User Authentication and Role-Based Access**:
    - Upon login, users are authenticated and redirected based on their role (administrator or client).
    - Administrators have a dashboard for user and device management, while clients are directed to a device view page.
- **Administrator Role**:
    - **CRUD Operations on Users**: Administrators can create, read, update, and delete user accounts, specifying each user's details.
    - **CRUD Operations on Devices**: Administrators can manage devices, with each device having an ID, description, address, and maximum hourly energy consumption.
    - **Assigning Devices to Users**: Administrators can map devices to users, allowing users to monitor specific devices.
- **Client Role**:
    - **Device Viewing**: Clients can view a list of all devices assigned to them. This interface provides an overview of energy monitoring data specific to the user's devices.
- **Access Control**:
    - **Role-Based Access Control**: Security measures prevent users from accessing areas outside their role. For instance, clients are blocked from accessing administrator functions, even if they try to access an admin URL directly.

4. Non-Functional Requirements

Non-functional requirements define the qualities or characteristics the EMS must maintain to function effectively. They focus on performance, security, scalability, and maintainability.

- **Performance**:

    - The EMS is expected to support efficient CRUD operations for both user and device management. A responsive frontend minimize latency, ensuring quick load times and smooth navigation.
- **Security**:
    - **Authentication**: User authentication is required to access the system. Role-based session storage in React maintains session security, ensuring that user data is protected.
    - **Role-Based Access Control**: Session storage restricts users to pages corresponding to their roles, adding a layer of security against unauthorized access.
- **Scalability**:
    - The microservices architecture allows for easy scaling by adding or removing services as needed. This modular approach ensures that as the EMS grows, additional resources can be allocated efficiently to meet increased demand.
- **Reliability and Fault Tolerance**:
    - Error-handling mechanisms ensure that unexpected failures (e.g., loss of connection to a microservice) are managed gracefully, preventing system crashes and protecting user data integrity.
- **Usability**:
    - The frontend, built with React TypeScript, provides a user-friendly interface that makes navigation intuitive for both administrators and clients, enhancing the overall user experience.

5. Implementation Details

5.1 Frontend

The frontend application has the following structure:

Components

The Components folder houses all the main UI components that make up the user interface. Each component is responsible for a specific part of the application or a particular page.

- **AccessDeniedPage**: A component that renders a message or view indicating that the user does not have permission to access a certain page or functionality.

- **AdminPage**: The main page for administrators, where they can perform their role-specific actions: managing users and devices.
- **ClientPage**: The main page for clients, where they view information about their assigned devices.
- **DevicePopUp**: A popup component designed to facilitate add and update operations for devices. It conditionally renders fields as either blank for adding a new device or pre-filled with existing information for updating an existing device.
- **EditUsersPage**: A page dedicated to editing user information, which administrators can access to update user details.
- **LoginPage**: The login interface where users authenticate to access the application.
- **PairDevicePopUp**: A popup component to pair devices with users, enabling administrators to assign specific devices to specific users.
- **ProtectedRoute**: A component that restricts access to certain routes based on user roles, ensuring that only authorized users can view specific pages.
- **Root**: Serves as the root component, which is the entry point of the application where primary components and routes are set up.
- **Router**: Manages the application's routes, linking different components to URLs and handling navigation.
- **UserPopUp**: A popup component designed to facilitate add and update operations for users. It conditionally renders fields as either blank for adding a new user or pre-filled with existing information for updating an existing user.

Library

The Library folder contains utility files, constants, types, and other shared resources used across multiple components.

- **Constants**:
    - constants.ts: Holds general constants used throughout the application, such as configuration settings, default values, or application-wide identifiers.
    - errorsConstants.ts: Contains constants specific to error messages, making it easier to manage and standardize error handling across the app.
    - themeConstants.ts: Contains styling constants, potentially for theme colors, font sizes, and other design-related configurations.
- **Enums**:

    - Role.ts: Defines roles of the application, Admin and Client, ensuring consistent role references across the application.

- **Models**:
    - IDevice.ts, IUser.ts, and IUserDevice.ts: TypeScript interfaces defining the structure for devices, users, and user-device relationships. These models standardize data types, enhancing type safety.

- **Services/Hooks**:
    - useUserRole.ts: A custom React hook that provides role-based information, determining the current user's role.

Utils

The Utils folder houses utility functions and reusable logic.

- **tableUtils.tsx**: Contains a utility function specific to table population and formatting table data, used by components displaying tabular data.

Resources/Images

This folder contains static resources, including images.

5.2 Backend

The backend part consists of 2 microservices(User Service and Device Service).

5.2.1 ***Device Microservice***

This microservice is responsible for managing device-related operations, including adding, updating, and retrieving devices.

- **Controller**: Contains classes that define REST endpoints related to device operations. These controllers handle HTTP requests for creating, updating, retrieving, and deleting device records, as well as mapping devices to users.
- **DTO** (Data Transfer Objects): Includes classes that serve as data carriers between different layers of the application, particularly for requests and responses.
- **Entity**: Defines the core data models representing the application's domain, such as Device and User entities. These classes map directly to database tables, providing the structure for the data managed by this microservice.
- **Exception**: Contains custom exception classes to handle specific error conditions, like attempting to create a duplicate entity or trying to perform operations on non-existent

entities. These exceptions enhance error handling and provide meaningful feedback to users.

- **Repository**: Provides the data access layer for device entities. This package contains repository interfaces that extend database frameworks allowing CRUD operations on device data.
- **Response**: Defines custom response structures, such as responses that return device information specific to a user.
- **Service**: Contains the business logic for device management. Service interfaces define the operations, while implementations provide the actual functionality. This package is central to managing device operations.
- **Utils**: Contains the security module of the backend, responsible for handling tasks such as password encryption to ensure secure storage and transmission of sensitive information.

5.2.2 User Microservice

This microservice is responsible for managing user-related operations, including adding, updating, and retrieving devices.

- **Controller**: Contains classes that define REST endpoints related to user operations. These controllers handle HTTP requests for creating, updating, retrieving, and deleting user records.
- **DTO** (Data Transfer Objects): Includes classes that serve as data carriers between different layers of the application, particularly for requests and responses.
- **Entity**: Defines the core data models representing the user domain, specifically the User entity, which maps to the database and represents user information.
- **Exception**: Contains custom exception classes to handle specific error conditions, like attempting to create a duplicate entity or trying to perform operations on non-existent entities. These exceptions enhance error handling and provide meaningful feedback to users.
- **Repository**: Houses the data access layer for user entities. Repository interfaces within this package interact with the database to perform CRUD operations related to user data.
- **Service**: Contains the business logic for user management, including operations related to login, account creation, and user updates. This package plays a central role in implementing user functionality and applying business rules.
- **Utils**: Contains the security module of the backend, responsible for handling tasks such as password encryption to ensure secure storage and transmission of sensitive information.

This package-level structure promotes clean separation of concerns, making each package responsible for a specific aspect of the application, such as data access, business logic, or request handling. This organization helps maintain scalability, readability, and modularity within the microservices.

6. Security

6.1 Authorization – Role based access

This application implements role-based access control to ensure that users can only access pages authorized for their specific role. This setup uses sessionStorage to store the user's role after login and a custom ProtectedRoute component to verify the role before rendering protected pages.

- **Storing User Role in sessionStorage:** After login, the user’s role is stored in sessionStorage. This allows the role information to persist during the session, enabling role-based access checks across navigations.

- **Role Verification with ProtectedRoute:** ProtectedRoute retrieves the user’s role from sessionStorage and checks if it matches the required role for each protected route. If roles match, the component is rendered; otherwise, the user is redirected to the access denied page.

- **Setting Up Protected Routes:** In the routes configuration, each restricted route is wrapped with ProtectedRoute, specifying the role required. For example, "admin" routes are restricted to users with an admin role, and "client" routes to client-role users.

These steps ensure the application securely restricts access based on user roles.

6.2 Authentication

The authentication process verifies the user's email and password, assigns them a role, and navigates to the appropriate page based on their role.

The frontend verifies credentials with the backend and returns a role indicator (0 for admin, 1 for client). Based on the role, sessionStorage stores "admin" or "client", and the user is redirected to the appropriate dashboard.

If credentials are invalid, an error alert is shown.

This setup ensures users are authenticated and directed to pages based on their roles, with sessionStorage enabling role-based access control across the session.

7. Resources

- https://docs.spring.io/spring-data/commons/docs/current/api/org/springframework/data/repository/CrudRepository.html#findAll()
- https://mui.com/material-ui/
-