**Energy Utility Platform**

**Assignment 1**

**Student: Lupsa Sergiu**

**Group: 30641**

**Contents**

[**1.** **Requirements analysis** 3](#_Toc118739008)

[**1.1** **Assignment specification** 3](#_Toc118739009)

[**1.2** **Functional requirements** 3](#_Toc118739010)

[**1.3** **Non-functional requirements** 3](#_Toc118739011)

[**2.** **Use-case model** 3](#_Toc118739012)

[**3.** **System architectural design** 5](#_Toc118739013)

[**3.1** **Architectural pattern description and diagrams** 5](#_Toc118739014)

[**Component diagram** 6](#_Toc118739015)

[**Deployment diagram** 6](#_Toc118739016)

[**4.** **Class design** 7](#_Toc118739017)

[**4.1**  **UML class diagram** 7](#_Toc118739018)

[**4.2** **Data model** 8](#_Toc118739019)

[**5.** **Bibliography** 8](#_Toc118739020)

1. **Analysis of requirements**
   1. **Assignment specification**

The first assignment requires an online platform regarding the energy utility which it should manage the association between users and smart energy devices that should include monitored pieces of information. The platform login can be done by two types of users: administrator and ordinary user (a client). The administrator can perform CRUD operations on both users and devices. Also, after the login of an admin, he will be redirected to devices page, and, after that, he can surf anywhere in the platform. In addition, the administrator can manage the mapping of users to devices, which will be stored in a database in tuples of the form <timestamp, energy consumption>. In case an user is logged in, he will be redirected to his personal page which contains his own devices only.

* 1. **Functional requirements**
* Authentication: administrator or user type of account
* Administrator login: redirected to devices page and see all devices and access any page
* Administrator login: CRUD operations on users and devices
* Administrator: can link users to devices (mapping)
* User login: redirected to his own page where he can see only his own devices and alocate new devices
* Security: the users corresponding to one role will not be able to enter the pages corresponding to the other role
  1. **Non-functional requirements**
* Usability: user-friendly interface for the application
* Security: minimal security assured by authentication and restrictions regarding access to different pages

1. **Use-case model**

Use case: Client login and see all his devices

Primary actor: Client

Main success scenario: Login -> See all devices associated

Extensions: This user doesn’t have any devices yet

Use case: Administrator login and see all devices and, by choice, all users

Primary actor: Administrator

Main success scenario: Login -> See all device -> Make CRUD operations on any devices -> Press users button -> Make CRUD operations on any user

Extensions: There are no devices yet, there are no users yet

**Diagram

Description automatically generated**

1. **System architectural design**
   1. **Architectural pattern description and diagrams**

Diagram

Description automatically generated

Layered architectures – the most common in software development. It describes an architectural pattern which is composed by separate horizontal layers that function together as a single unit of software.

This diagram contains all the components of the application. The main functionalities are based on 3 main components: the database (Postgres), REST API server and Angular Frontend for client visualization. The client makes a requirement which is transferred to the Server, and after that, a query is sent to the database. Then the results will be shown to the client.

### **Component diagram**

Diagram

Description automatically generated

### **Deployment diagram**

**Diagram

Description automatically generated**

This diagram shows the presence of three Docker containers.

These 3 containers are: the container for the database (Postgres container), the container for the server (Backend container) and the container for the client application (Frontend container).

Each container has an image built on the basis of a Dockerfile, an image that can be run locally from Docker containers and not from an IDE. For this application, it was deployed locally, running from Docker containers. The component that encapsulates the server is dependent on the database, and the client side is dependent on the database as well as the server.

The communication of the 3 components is done through a network that creates a communication bridge.

1. **Class design**
   1. **UML class diagram**

**A screenshot of a computer

Description automatically generated with medium confidence**

## 

## **4.2 Database model diagram**

**A picture containing diagram

Description automatically generated**

1. **Implementation**

The backend application was made in Spring using the Java language. The application was structured using the following packages: entities (for the application models), dtos (to transmit the first data from the frontend), repositories (for data persistence), services (for the services in which the application logic is), controllers (classes in which the endpoints of the application and the calls to the corresponding services are declared - RESTful API) and security (in which the security of the application is ensured).

The frontend application is written in Javascript, using the React framework. The structure of the application is organized by packages, so that it is as readable and easy to understand as possible. The main packages include components such as: Login component, Device-table component, User-table component etc. An ordinary user (client) has access only to his page (User-page component), and an administrator has access to all components. This is ensured by an Authorization Guard (does not allow access to pages if the user is not logged in) and a Role Guard (does not allow access to pages for which permission is not granted, depending on the role).

Data saving is ensured by the PostgresSQL application, where the database is located. The database is also secured, requiring a password and a username to access the databases

1. **Bibliography**

<https://dsrl.eu/courses/sd/>

<https://docs.docker.com/desktop/install/windows-install/>

<https://material.angular.io/components/table/overview>