

ENERGY MONITORING SYSTEM

Introduction

The project proposes an energy monitoring dashboard where the user can retrieve data from a server and view devices and their consumption. In our highly connected world, users need to keep track of their Internet of Things Devices. This project proposes a solution for managing a database of such devices. This system has two components:

- Front-end user interface
- Back-end server and database

Besides the two components, the system must also accommodate two types of users:

- Admin users
- Normal users

Front end

The front-end part of the application consists of a web app. Users should be able to create an account and log in. Once logged in, each user will be directed to a particular page depending on their role. If the user is an admin, he will be directed to the admin dashboard where he can view all users. The admin should be able to perform crud operations on users and devices. He must be able to manage both devices and users as well as to create associations between the two groups. When a device is associated with a user, the user can see the device in his device list. Most importantly, the admin should be able to add and remove devices, as well as users. He is also able to modify the email address of a user.

The user on the other hand will be redirected to the user page. He is more constrained in the web application, being able to only read data. Once he logs in, he should be taken to the user page where he should be able to view his device list. He should be able to select one of such devices to view details about it, such as id, description, address and maximum hourly consumption.

Besides the data about the device, he should also be allowed to view a graph of energy consumption. He should have the option of choosing the date he wants to view the consumption on, and once he does so, he will be shown each level of energy consumed every hour on that day.

Back end

The back end of the application is a web server that should expose a number of endpoints to allow both user groups to perform their desired actions. These endpoints should provide functionality through web requests. There should be a set of CRUD endpoints for each resource in the backend, i.e. user, device and energy consumption. The endpoints should perform queries on the application database to retrieve the requested data or make changes to already existing data. Some endpoints should be accessed only by users, while others only by

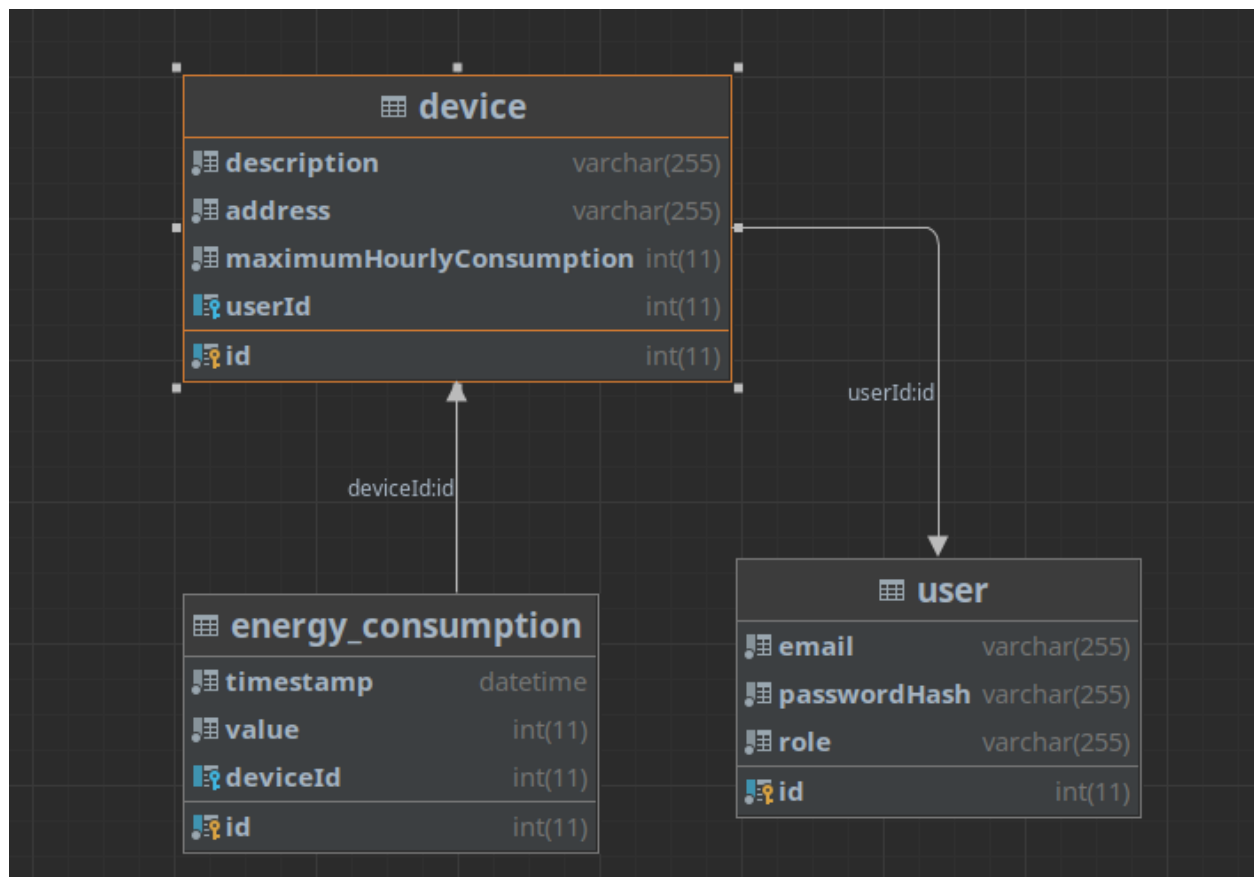
administrators, therefore there are special endpoints for each group, like retrieving all users for administrators or retrieving device energy consumption for a specific device for user.

Implementation

Database

The database of the project was implemented using a MySQL database server. In it, three tables are created to hold information on our users, devices and energy consumption. Between these entities, there are relations. Each device has associated an user or no users. The devices without any users are referred to as orphans. Each user, therefore, has multiple devices. To each device in turn there is a set of energy consumption objects associated, representing the data gathered from the device. Each energy consumption entry has a timestamp and a value representing the consumption in kilowatts, therefore each device has multiple energy consumption records.

Each entity in the database has also a unique id associated to it. The database schema looks as follows:



Back-end

The backend server of the application was realized using Nest JS. Nest JS is a modern framework for creating REST applications, microservices, GraphQL APIs and many more. It is powered by Typescript and has many very well-implemented plugins for a number of Typescript/Javascript packages to achieve multiple common tasks. Its design is a three-tier architectural pattern, consisting of a controller, service and data access layers.

In the implementation, multiple NestJs plugins were used:

- Passport

Passport is a javascript library used for authentication and authorization. It implements the strategy pattern in order to assess the access of users. Among these strategies were local and jwt authentication.

- TypeORM

TypeORM is a fully featured ORM for typescript. It works by mapping each entity to a typescript object and combines the powerful typescript feature of decorators to achieve ease of use when defining columns, keys and relations. It can be used in both active entity pattern mode and repository pattern mode. For this project, the repository pattern was used. TypeORM can infer relations and features of entities in your app and automatically generate repositories that simplify common queries like filtering, pagination, creating, updating and deleting records. This was the main ORM in the application and serves as the entirety of the data access layer.

The service layer is facilitated by the powerful dependency injection engine of NestJS, which compartmentalizes each resource as a module which exposes controllers, and providers and imports resources from other modules. Each resource in turn has a Controller, Service and Repository. When needed, modules can import resources from other modules. This way, the class coupling is kept low and the code is very orderly.

The controller layer exposes multiple endpoints that can be accessed from the outside. This tier has the task of mapping requests to their appropriate handlers. Again, using decorators, NestJS can automatically generate these mappings. A controller is just a class that contains a set of functions that are mapped to the router of the web server.

Front-end

The front end of the application was realized in Vue.js 3 using composition API. The app consists of four pages managed by the main router, allowing for the app to be a single-page application. The state management was done using Pinia stores and the interaction with the backend was achieved using the Axios package. The routing was done with the official Vue router package, which provided many features like re-routing, route guards and parameter extraction.

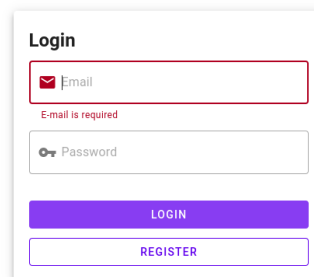
The four pages are:

- User page
- Admin page
- Register page
- Login page

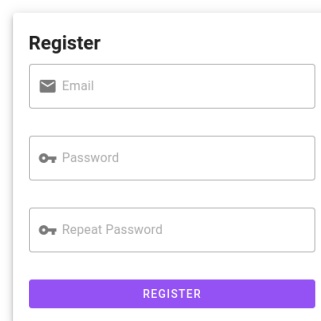
If the user is not logged in, he will be redirected to the login page from anywhere in the app.

From the login page, he can access the register page to create an account. If logging in with an administrator account, he is taken to the admin page where he can manage devices and other users. If he is a normal user, he is taken to his own page to view the devices associated to him.

Login Pages:

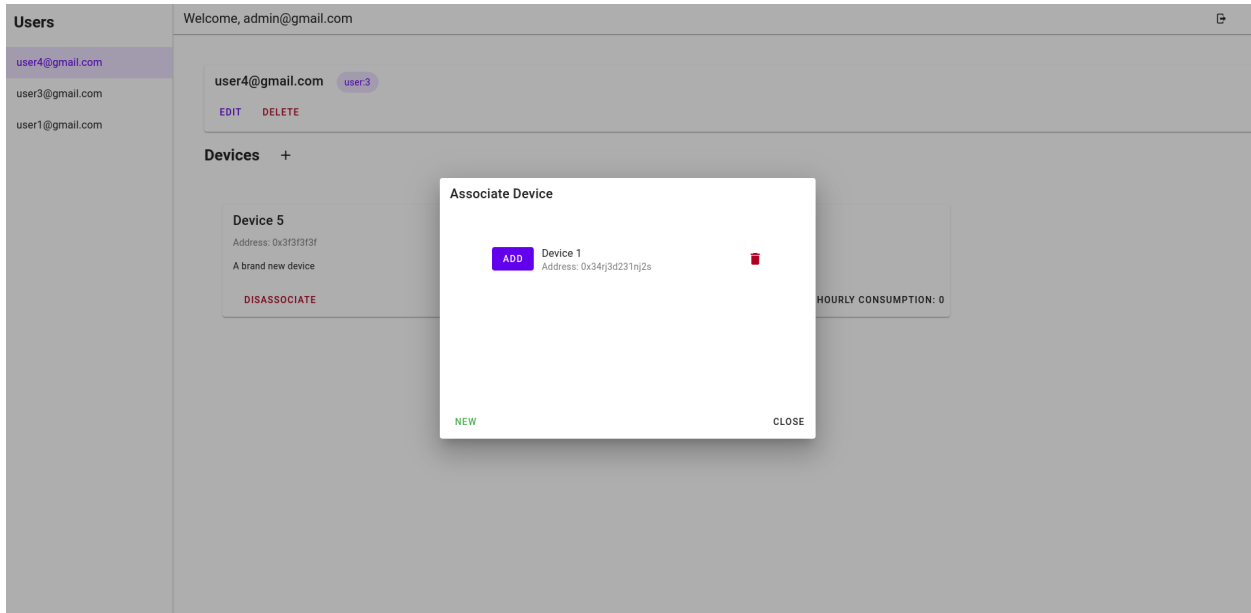
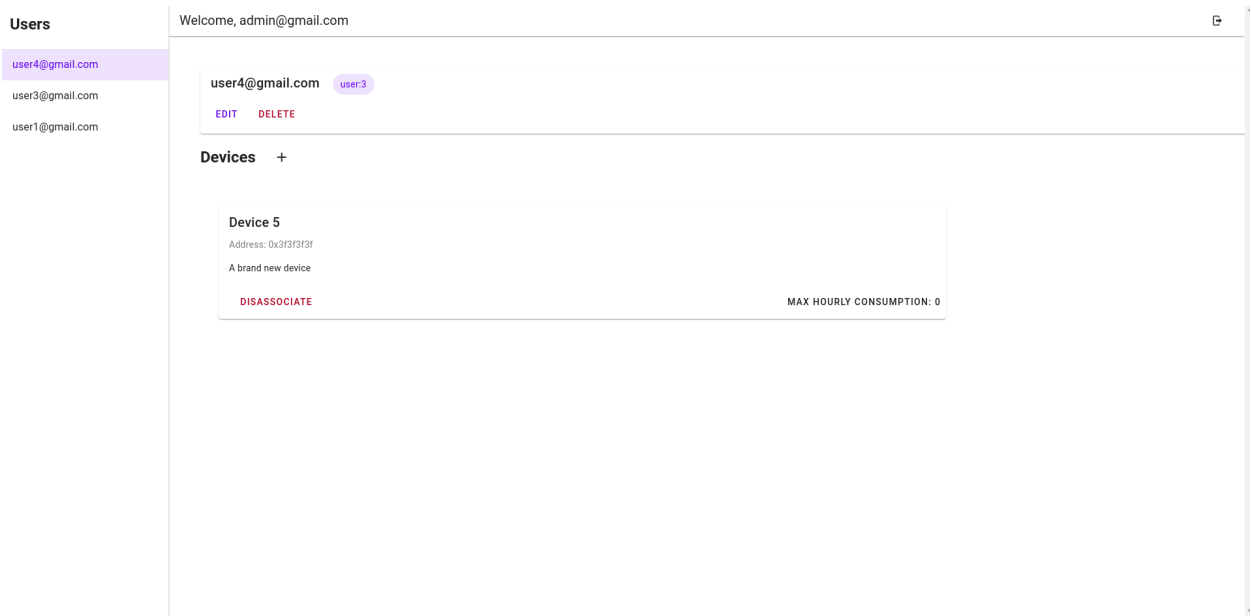


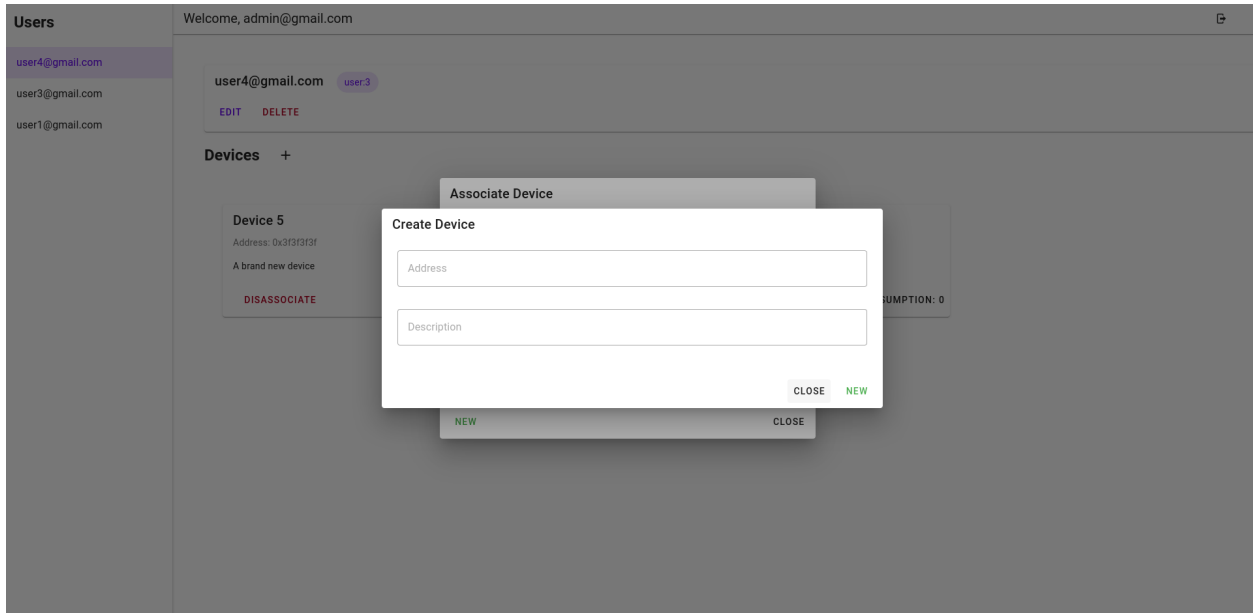
The image shows a 'Login' form with a title 'Login' at the top. Below the title is an email input field with an envelope icon and the placeholder text 'Email'. A red error message 'E-mail is required' is displayed below the email field. Below the email field is a password input field with a key icon and the placeholder text 'Password'. At the bottom of the form are two buttons: a solid purple 'LOGIN' button and a white 'REGISTER' button with a purple border.



The image shows a 'Register' form with a title 'Register' at the top. Below the title are three input fields: an email field with an envelope icon and placeholder 'Email', a password field with a key icon and placeholder 'Password', and a repeat password field with a key icon and placeholder 'Repeat Password'. At the bottom of the form is a solid purple 'REGISTER' button.

Admin Page:





User Page:

