4/11/2025

Guezzane, Ahmed Fateh

**SmartPower Hub**

Content Table

[Introduction 2](#_Toc195525620)

[Report 3](#_Toc195525621)

[Technologies Used 3](#_Toc195525622)

[Database Structure 4](#_Toc195525623)

[Backend Architecture 5](#_Toc195525624)

[Frontend Features 6](#_Toc195525625)

[Web Application Pages 7](#_Toc195525626)

[Additional Details and Documentation 12](#_Toc195525627)

[Express.js Architecture 12](#_Toc195525628)

[Mongoose Data Model 12](#_Toc195525629)

[API RESTful 12](#_Toc195525630)

[User Interface 13](#_Toc195525631)

[User Experience (UX) 13](#_Toc195525632)

[Suggestions for Future Improvements 13](#_Toc195525633)

[Conclusion 14](#_Toc195525634)

# Introduction

This project is a modern web application designed for the centralized management of electrical devices. It allows users to add new devices, view their details, modify or delete them. Each device is associated with specific data such as daily, monthly, and yearly energy consumption, standby mode, category, brand, and type. This information is displayed clearly and intuitively using graphical elements like interactive circles representing the different consumption metrics.

The application is distinguished by its responsive and user-friendly interface, making it easy to navigate and interact with the data. Thanks to a dynamic search system, users can quickly filter the displayed devices. The design is modern, supports both light and dark mode, and is intended to provide a smooth and pleasant user experience.

The project is especially suitable for educational or professional contexts—such as schools, labs, companies, or any space requiring careful management of electronic equipment. Development students can also use it as a base for academic projects, portfolios, or local deployments.

It showcases a robust and well-structured architecture, with a clear separation between the frontend (React) and backend (Express.js), and uses MongoDB to store device data. It is a good example of a complete, functional, and extensible web application.

# Report

## Technologies Used

The main language used in this project is JavaScript, recognized as one of the most popular and versatile languages in modern web development. JavaScript is used throughout the entire stack—both on the client side (frontend) and the server side (backend)—which simplifies code unification and communication between the different layers of the application.

**Frontend**

The frontend is developed using React, a JavaScript library that allows for building dynamic and responsive user interfaces through modular components. React simplifies state management, routing (via React Router), and interactions with the backend API.  
The visual design is handled by TailwindCSS, a utility-first CSS framework that enables styling directly within HTML class attributes. This provides fast development, consistent design, and native support for dark mode.

**Backend**

Le backend repose sur **Node.js**, un environnement d’exécution JavaScript côté serveur, associé à **Express.js**, un framework léger et puissant pour la création d’API REST. Express permet de gérer les routes, les middlewares (comme CORS, parsing JSON) et les communications entre le frontend et la base de données. Le serveur est configuré pour recevoir des requêtes HTTP, traiter les données, et répondre de manière sécurisée.

**Database**

The device data is stored in a MongoDB database, a document-oriented NoSQL database ideal for flexible data structures. The Mongoose library is used to model the data, define schemas, and perform CRUD operations in a structured way. It also simplifies validation and manipulation of MongoDB objects.

**Development Tools**

The project uses **Vite** as a build and development tool, offering an ultra-fast development environment with Hot Module Replacement (HMR) for a smooth and reactive development cycle.

Additionally, **MongoDB Compass** is used to visually explore the database, view collections and documents, run queries, and monitor storage status without using the command line.

## Database Structure

The Mongoose model used to interface with MongoDB defines the following fields:

* **id**: A unique string used to identify each device. This is used in addition to the MongoDB \_id to ensure a custom identifier within the application.
* **name**, **brand**, **type**, **category**: Text fields used to describe the device. For example, a device could be named “Canon Printer,” be of type “printer,” branded “Canon,” and categorized under “Office.”
* **energyLabel**: A standardized label (e.g., A++, A+, B, etc.) used to classify devices based on energy efficiency. This helps quickly identify energy-saving devices.
* **description**: A free-text field allowing users to add additional details about the device (features, location, remarks, etc.).
* **averageConsumption**: An object containing three numeric values:
  + dailyKWh: average daily consumption in kilowatt-hours.
  + monthlyKWh : monthly consumption.
  + annualKWh : yearly consumption.
* **powerRating**: An object representing the device’s power rating
  + **minWatts**: minimum power in watts.
  + **maxWatts**: maximum power in watts.
* **standbyConsumption** : Indicates the device’s power usage while in standby mode, which is especially important for always-connected or idle devices.

## Backend Architecture

The backend of this application is built with **Express.js**, a lightweight and powerful framework based on **Node.js**. Express allows the creation of high-performance, well-structured web servers while simplifying the implementation of a clear and maintainable REST API.

**Main role**: Expose a REST API

The backend's primary function is to provide a communication interface between the frontend and the MongoDB database. To do so, it exposes a set of HTTP routes that follow REST principles — meaning a set of URLs and methods (GET, POST, PUT, DELETE) enabling all CRUD operations (Create, Read, Update, Delete).

**Available Routes**

* **GET /api/devices:** Retrieves the list of all registered devices.
* **GET /api/devices/:id:** Returns a single device matching the specified ID.
* **POST /api/devices:** Creates a new device by sending its data in the request body (in JSON format).
* **PUT /api/devices/:id:** Updates the information of an existing device.
* **DELETE /api/devices/:id:** Permanently deletes a device from the database.

These routes are logically grouped in a dedicated file, routes/devices.js, which helps maintain a clean and modular project structure.

The index.js file serves as the main entry point of the backend server. It is responsible for configuring the Express application, establishing the database connection, and integrating the API routes.

## Frontend Features

The frontend of the application is developed using **React**, a highly popular JavaScript library for building dynamic user interfaces. It is combined with **TailwindCSS**, a modern utility-first CSS framework that allows for fast styling directly within JSX using class names.

The entire interface is **responsive** (adapted to different screen sizes), supports both **light and dark themes**, and is intuitively organized around a fixed **sidebar** for navigation. Page transitions are handled using **React Router**, a library designed for routing in single-page React applications (SPA).

**Component and Page Structure**

Here are the main components and pages of the frontend:

* **Home.jsx**  
  The homepage provides an overview of the application's main features (adding, viewing, editing devices) and introduces the different sections of the interface. It serves as an entry point to guide the user.
* **Sidebar.jsx**  
  A fixed vertical menu on the left side of the screen with interactive icons for navigating between the homepage, the device list, and the add device form. It also includes a button to toggle dark mode, with icon changes on hover.
* **DeviceList.jsx**  
  This page displays the list of devices along with a search bar to filter by name, brand, or type. When a device is selected, its details appear on the right.
* **DeviceDetails.jsx**  
  This component shows the details of the selected device, including metadata and colored circles to visualize consumption data (kWh and Watts).
* **DeviceCreate.jsx**  
  A form for adding a new device, with fields styled using TailwindCSS. After submission, the user is redirected to the list with a confirmation message.
* **DeviceUpdate.jsx**  
  Similar to DeviceCreate, this pre-filled form allows easy editing of an existing device. The user can update the fields or cancel to return to the list.

## Web Application Pages

The application features a modern, intuitive, and responsive interface. It is designed to simplify the management of electrical devices by providing smooth navigation, a structured data display, and clear forms. The following visual elements illustrate the key pages of the application:

**Home Page**

Displays a summary of the main features and guides the user as soon as the application is launched.A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

**Device List and Device Details**

Displays all devices with a search bar to filter by name, type, or brand. It also shows detailed information for the selected device, including its consumption data visualized with colored circles.A screenshot of a computer

AI-generated content may be incorrect.

**A screenshot of a computer

AI-generated content may be incorrect.**

**Creation Form**

Allows users to add a new device using clear input fields with automatic validation.A screenshot of a computer

AI-generated content may be incorrect.

**A screenshot of a computer

AI-generated content may be incorrect.**

**Update Form**

A pre-filled form for editing an existing device, with a cancel option available.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

# Additional Details and Documentation

## Express.js Architecture

The Node.js server is structured around the Express framework. The application’s entry point is the index.js file, which configures the Express app, sets the port, connects to the MongoDB database using Mongoose, and mounts the routes defined in the routes folder.

The main router is in routes/devices.js, which contains all the necessary routes for managing devices (CRUD). The controllers are written directly in this file, making the project simple to understand and modify, especially for student or rapid development use cases.

## Mongoose Data Model

The models/Device.js file defines the schema for devices. This schema enforces the document structure within MongoDB, using types such as String, Number, and nested objects like averageConsumption and powerRating.

Mongoose enables server-side data validation and provides a convenient abstraction layer for interacting with MongoDB (queries, updates, deletions, etc.).

## API RESTful

The API follows REST standards, allowing for straightforward interaction via Postman, Axios, or Fetch.  
Each route corresponds to a CRUD operation:

* **GET /api/devices:** retrieves the full list of devices
* **GET /api/devices/:id:** returns a single device by ID
* **POST /api/devices:** creates a new device
* **PUT /api/devices/:id:** updates an existing device
* **DELETE /api/devices/:id:** deletes a device

Responses are returned in JSON format, and errors are handled using standard HTTP codes (404, 500, etc.).

## User Interface

The frontend uses React to build responsive components, with TailwindCSS for design. Special attention was given to visual consistency:

* Forms use rounded, borderless input fields, inspired by modern mobile UI trends.
* Dark mode is toggled dynamically from the bottom of the Sidebar, and preferences are stored in localStorage.
* DeviceList supports dynamic search and highlights the most recently created or updated device.
* DeviceDetails displays key metrics (daily kWh, standby W, etc.) in animated circular visualizations.

## User Experience (UX)

The app is designed to be simple and smooth to use:

* Redirection after form submission with success message
* Auto-selection of the created or updated device
* Visual feedback (green or red alerts) displayed at the top of the screen
* One-click dark mode toggle

## Suggestions for Future Improvements

* Add authentication (JWT, sessions)
* Enable PDF/CSV data export
* Include a consumption trend graph
* Develop an admin dashboard with advanced filters

# Conclusion

This project resulted in the development of a complete and functional web application dedicated to managing electrical devices. By leveraging modern technologies such as React for the frontend, TailwindCSS for styling, Node.js with Express for the server, and MongoDB as the database, the entire system is built on a robust, modular, and scalable architecture. The interface is clear, smooth, and accessible, allowing any user to easily navigate between pages, view data, add new devices, or modify existing ones. The use of colored circles to visualize statistics enhances the readability of technical information and improves the understanding of energy consumption.

This project meets the needs of equipment management in environments such as schools or businesses. It simplifies routine operations and centralizes key data in a single tool accessible from any connected workstation. It also serves as a solid foundation for future enhancements.

Indeed, several improvement paths could be considered to enrich the user experience and expand the application’s usability both functionally and administratively. For example, implementing an authentication system would allow differentiated access between regular users and administrators. This would strengthen security and provide a personalized experience based on user roles. Another valuable addition would be multilingual support, making the application accessible to an international audience by enabling easy language switching between French, English, or other languages. Furthermore, exporting data in various formats such as PDF or Excel would give managers an efficient way to generate reports for sharing or archiving. Finally, integrating a centralized dashboard would aggregate key information in the form of graphs or synthetic indicators, facilitating the overall analysis of energy consumption.

In summary, this application is a powerful and adaptable solution, with many opportunities for improvement to make it even more complete, intelligent, and aligned with the practical needs of users.