11 avril 2025

AHMED-FATEH GUEZZANE

**SmartPower Hub**

Gestion d’appareils éléctroniques

Table des matières

[Introduction 2](#_Toc195525796)

[Rapport 3](#_Toc195525797)

[Technologies utilisées 3](#_Toc195525798)

[Structure de la base de données 4](#_Toc195525799)

[Architecture backend 5](#_Toc195525800)

[Fonctionnalités frontend 6](#_Toc195525801)

[Les pages de l’application web 7](#_Toc195525802)

[Détails complémentaires et documentation 12](#_Toc195525803)

[Architecture Express.js 12](#_Toc195525804)

[Modèle de données avec Mongoose 12](#_Toc195525805)

[API RESTful 12](#_Toc195525806)

[Interface utilisateur 13](#_Toc195525807)

[Expérience utilisateur (UX) 13](#_Toc195525808)

[Suggestions pour l’amélioration future 13](#_Toc195525809)

[Conclusion 14](#_Toc195525810)

# Introduction

Ce projet est une application web moderne conçue pour la **gestion centralisée d'appareils électriques**. Elle permet à l’utilisateur d’ajouter de nouveaux appareils, de consulter leurs détails, de les modifier ou de les supprimer. Chaque appareil est accompagné de données précises telles que sa consommation énergétique quotidienne, mensuelle et annuelle, son mode veille, sa catégorie, sa marque et son type. Ces informations sont visualisées de façon claire et intuitive, à l’aide d’éléments graphiques comme des cercles interactifs représentant les différentes mesures de consommation.

L’application se distingue par son interface réactive et conviviale, qui facilite la navigation et l’interaction avec les données. Grâce à un **système de recherche dynamique**, l’utilisateur peut filtrer rapidement les appareils affichés. Le design est moderne, compatible avec un **mode clair et sombre**, et pensé pour offrir une expérience utilisateur fluide et agréable.

Le projet est particulièrement adapté à un **contexte éducatif ou professionnel**, par exemple dans une école, un laboratoire, une entreprise ou tout autre espace nécessitant une gestion rigoureuse de plusieurs équipements électroniques. Les étudiants en développement peuvent également l'utiliser comme base pour un projet académique, un portfolio ou un déploiement local.

Ce projet met en avant une architecture solide et bien structurée, avec une séparation claire entre le **frontend en React** et le **backend en Express.js**, tout en utilisant une base de données **MongoDB** pour stocker les données des appareils. Il constitue un bon exemple d'application web complète, fonctionnelle et extensible.

# Rapport

## Technologies utilisées

Le principal langage utilisé dans ce projet est **JavaScript**, reconnu comme l’un des langages les plus populaires et polyvalents dans le développement web moderne. JavaScript est ici utilisé sur l’ensemble de la stack, à la fois **côté client (frontend)** et **côté serveur (backend)**, ce qui facilite l’unification du code et la communication entre les différentes couches de l’application.

**Frontend**

Le frontend est développé avec **React**, une bibliothèque JavaScript permettant de construire des interfaces utilisateur dynamiques et réactives sous forme de composants modulaires. React facilite la gestion de l’état, du routage (via React Router) et des interactions avec l’API backend.

Le design visuel est assuré par **TailwindCSS**, un framework CSS utilitaire qui permet de styliser les composants directement dans les classes HTML. Cela offre une grande rapidité de développement, un design cohérent, et une prise en charge native du mode sombre (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.

**Base de données**

Les données des appareils sont stockées dans une base de données **MongoDB**, une base NoSQL orientée documents, idéale pour des structures de données flexibles. La bibliothèque **Mongoose** est utilisée pour modéliser les données, définir des schémas, et effectuer les opérations CRUD de manière structurée. Elle facilite également la validation et la manipulation des objets MongoDB.

**Outils de développement**

Le projet utilise **Vite** comme outil de build et de développement. Vite offre un environnement de développement ultra-rapide avec rechargement à chaud (Hot Module Replacement), ce qui permet un cycle de développement fluide et réactif.

Enfin, **MongoDB Compass** est utilisé pour explorer visuellement la base de données, visualiser les collections et documents, effectuer des requêtes, et surveiller l’état du stockage sans passer par la ligne de commande.

## Structure de la base de données

Le modèle Mongoose utilisé pour interfacer avec MongoDB définit les champs suivants :

* **id** : une chaîne de caractères unique pour identifier chaque appareil. Ce champ est utilisé en plus de l’identifiant MongoDB (\_id) afin de garantir un identifiant personnalisé dans l'application.
* **name**, **brand**, **type**, **category** : des champs textuels servant à décrire l'appareil. Par exemple, un appareil pourrait s'appeler *"Imprimante Canon"*, être de type *"imprimante"*, de marque *"Canon"* et appartenir à la catégorie *"Bureau"*.
* **energyLabel** : une étiquette standardisée (ex. : A++, A+, B, etc.) permettant de classer les appareils selon leur efficacité énergétique. Cela aide à identifier rapidement les appareils économes.
* **description** : une zone de texte libre qui permet à l’utilisateur d’ajouter des détails supplémentaires sur l'appareil (fonctionnalité, emplacement, remarques...).
* **averageConsumption** : un objet contenant trois valeurs numériques
  + dailyKWh : consommation moyenne quotidienne en kilowattheures.
  + monthlyKWh : consommation mensuelle.
  + annualKWh : consommation annuelle.
* **powerRating** : un objet représentant la puissance électrique de l’appareil
  + **minWatts** : puissance minimale (en watts).
  + **maxWatts** : puissance maximale (en watts).
* **standbyConsumption** : indique la consommation de l’appareil lorsqu’il est en veille. C’est un critère important dans les dispositifs connectés ou laissés allumés en permanence.

## Architecture backend

Le **backend** de cette application est construit avec **Express.js**, un framework léger et puissant basé sur **Node.js**. Express permet de créer des serveurs web performants et structurés, tout en facilitant la mise en place d’une **API REST** claire et maintenable.

Fonction principale : exposer une API REST

Le rôle principal du backend est de fournir une **interface de communication entre le frontend et la base de données MongoDB**. Pour cela, il expose une série de **routes HTTP** qui suivent la logique d’une API REST, c’est-à-dire un ensemble d’URL et de méthodes (GET, POST, PUT, DELETE) permettant d’effectuer toutes les opérations CRUD (Create, Read, Update, Delete).

Voici les routes disponibles :

* **GET /api/devices** : récupère la liste de tous les appareils enregistrés.
* **GET /api/devices/:id** : retourne un seul appareil correspondant à l’identifiant spécifié.
* **POST /api/devices** : permet de créer un nouvel appareil en envoyant ses données via le corps de la requête (format JSON).
* **PUT /api/devices/:id** : met à jour les informations d’un appareil existant.
* **DELETE /api/devices/:id** : supprime définitivement un appareil de la base de données.

Ces routes sont regroupées de manière logique dans un fichier dédié, **routes/devices.js**, ce qui permet de garder une structure de projet claire et modulaire.

Le fichier **index.js** constitue le **point d’entrée principal** du serveur backend. Il est responsable de la configuration de l'application Express, de l’établissement de la connexion à la base de données, et de l’intégration des routes de l’API.

## Fonctionnalités frontend

Le **frontend** de l’application est développé en **React**, une bibliothèque JavaScript très populaire pour la création d’interfaces utilisateur dynamiques. Il est associé à **TailwindCSS**, un framework CSS utilitaire moderne qui permet de styliser les composants rapidement avec des classes directement dans le JSX.

L’ensemble de l’interface est **responsif** (adaptée à différents écrans), **thémée en clair/sombre**, et organisée de manière intuitive autour d’une **navigation latérale fixe** (sidebar). La navigation entre les différentes pages se fait grâce à **React Router**, une bibliothèque dédiée au routage dans les applications React monopage (SPA).

**Structure des composants et pages**

Voici les principaux composants et pages du frontend :

* **Home.jsx**

La page d’accueil présente un aperçu des principales fonctionnalités de l’application (ajout, visualisation, modification d’appareils) et introduit les différentes sections de l’interface. Elle sert de point d’entrée pour orienter l’utilisateur.

* **Sidebar.jsx**  
  Un menu vertical fixe à gauche de l’écran avec des icônes interactives pour naviguer entre l’accueil, la liste et l’ajout d’appareils. Il inclut aussi un bouton pour activer le mode sombre, avec changement d’icône au survol.
* **DeviceList.jsx**  
  Cette page affiche la liste des appareils avec une barre de recherche pour filtrer par nom, marque ou type. En sélectionnant un appareil, ses détails s’affichent à droite.
* **DeviceDetails.jsx**  
  Ce composant affiche les détails d’un appareil sélectionné, avec ses métadonnées et des cercles colorés pour visualiser les données de consommation (kWh et Watts).
* **DeviceCreate.jsx**  
  Formulaire permettant d’ajouter un nouvel appareil, avec champs stylisés via TailwindCSS. Après soumission, l’utilisateur est redirigé vers la liste avec un message de confirmation.
* **DeviceUpdate.jsx**  
  Similaire à DeviceCreate, ce formulaire pré-rempli permet de modifier facilement un appareil. L’utilisateur peut éditer les champs ou annuler pour revenir à la liste.

## Les pages de l’application web

L’application dispose d’une interface moderne, intuitive et réactive. Elle a été conçue pour simplifier la gestion des appareils électriques, en offrant une navigation fluide, un affichage structuré des données, et des formulaires clairs. Les éléments visuels suivants illustrent les principales pages de l’application.

**Page d’accueil**

Présente un résumé des fonctionnalités et oriente l’utilisateur dès l’ouverture de l’application.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

**Liste des appareils et détail d’un appareil**

Affiche tous les appareils avec une barre de recherche pour filtrer par nom, type ou marque. Montre également les informations d’un appareil sélectionné, avec ses données de consommation en cercles colorés.

**A screenshot of a computer

AI-generated content may be incorrect.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Formulaire de création**

Permet d’ajouter un nouvel appareil avec des champs clairs et validés automatiquement.

**A screenshot of a computer

AI-generated content may be incorrect.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Formulaire de mise à jour**

Formulaire pré-rempli pour modifier un appareil existant, avec option d’annulation.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

# Détails complémentaires et documentation

## Architecture Express.js

Le serveur Node.js est structuré autour du framework Express. Le point d’entrée de l’application est le fichier `index.js`, qui configure l’application Express, définit le port, connecte la base de données MongoDB via Mongoose et monte les routes définies dans le dossier `routes`.

Le routeur principal est `routes/devices.js`. Ce fichier contient toutes les routes nécessaires à la gestion des appareils (CRUD). Les contrôleurs sont directement intégrés dans ce fichier, ce qui rend le projet simple à comprendre et à modifier pour des projets étudiants ou en développement rapide.

## Modèle de données avec Mongoose

Le fichier `models/Device.js` contient la définition du schéma des appareils. Ce schéma impose la structure des documents dans la base MongoDB, avec des types comme `String`, `Number`, et des objets imbriqués comme `averageConsumption` et `powerRating`. L'utilisation de Mongoose permet une validation des données côté serveur, ainsi qu'une abstraction pratique pour interagir avec MongoDB (requêtes, mises à jour, suppression, etc.).

## API RESTful

L’API exposée est conforme aux standards REST, permettant des appels simples via Postman, Axios ou Fetch.

Chaque route correspond à une opération :

* `GET /api/devices` : récupère la liste complète des appareils.
* `GET /api/devices/:id` : retourne un seul appareil selon son identifiant.
* `POST /api/devices` : crée un nouvel appareil.
* `PUT /api/devices/:id` : modifie un appareil existant.
* `DELETE /api/devices/:id` : supprime un appareil.

Les réponses sont retournées au format JSON et les erreurs sont gérées par des codes HTTP standards (404, 500, etc.).

## Interface utilisateur

Le frontend utilise React pour créer des composants réactifs, et TailwindCSS pour le design. Un effort particulier a été fait sur le design :

* Les formulaires utilisent des champs arrondis, sans bordure apparente, inspirés des interfaces modernes mobiles.
* Le thème sombre est géré dynamiquement à l’aide d’un toggle en bas de la `Sidebar`, et la préférence est mémorisée dans le localStorage.
* La `DeviceList` permet la recherche dynamique d’un appareil, avec surbrillance automatique du dernier appareil modifié ou créé.
* La `DeviceDetails` utilise des cercles pour afficher les métriques principales (daily kWh, standby W, …), avec des animations au survol.

## Expérience utilisateur (UX)

* Le site est pensé pour être simple et fluide :
* Redirections après soumission de formulaire avec message de succès
* Sélection automatique de l’appareil modifié ou ajouté
* Feedback visuel (alertes vertes ou rouges) affiché en haut de page
* Dark mode activable d’un simple clic

## Suggestions pour l’amélioration future

* Ajout d’une authentification (JWT, sessions)
* Export PDF/CSV des données
* Ajout d’un graphique d’évolution de consummation
* Interface d’administration avec filtres plus complexes

# Conclusion

Ce projet a permis de développer une application web complète et fonctionnelle dédiée à la gestion d’appareils électriques. Grâce à l’utilisation de technologies modernes telles que React pour le frontend, TailwindCSS pour le design, Node.js avec Express pour le serveur et MongoDB comme base de données, l’ensemble du système repose sur une architecture robuste, modulaire et évolutive. L’interface est claire, fluide et accessible, permettant à tout utilisateur de naviguer facilement entre les différentes pages, de consulter les données, d’ajouter de nouveaux appareils ou de modifier ceux existants. La visualisation des statistiques sous forme de cercles colorés améliore la lisibilité des informations techniques et offre une meilleure compréhension des consommations énergétiques.

Ce projet répond aux besoins de gestion d’équipements dans des contextes comme des établissements scolaires ou des entreprises. Il simplifie les opérations courantes et centralise les données utiles dans un outil unique, accessible depuis n’importe quel poste connecté. Il constitue également une base solide pour des évolutions futures.

En effet, plusieurs pistes d’amélioration peuvent être envisagées afin d’enrichir l’expérience utilisateur et d’ouvrir l’application à un usage plus large, tant sur le plan fonctionnel qu’administratif. La mise en place d’un système d’authentification permettrait par exemple de distinguer les accès entre simples utilisateurs et administrateurs. Cela renforcerait la sécurité et offrirait une expérience personnalisée selon les rôles. Une autre amélioration intéressante serait l’ajout d’un système multilingue pour rendre l’application accessible à un public international, en permettant notamment de basculer facilement entre le français, l’anglais ou toute autre langue. De plus, la possibilité d’exporter les données sous différents formats, comme PDF ou Excel, offrirait aux gestionnaires un moyen efficace de produire des rapports à partager ou à archiver. Enfin, l’intégration d’un tableau de bord centraliserait les informations clés sous forme de graphiques ou d’indicateurs synthétiques, facilitant ainsi l’analyse globale de la consommation énergétique.

En résumé, cette application constitue une solution performante et adaptable, et les perspectives d’amélioration sont nombreuses pour en faire un outil encore plus complet, intelligent et orienté vers les besoins concrets des utilisateurs.