Electricity Tracker and Visualizer

JID – 4343

Hy Minh Le, Paris Tran, Junseob Lee, Jacob Truong, Aaron Hung

Client: SLB – Somesh Saravanan, Yassine Bennani

Repository: https://github.com/hyminhle/JID-4343-Electricity-Tracker.git

Table of Contents

Table of Figures

Terminology

Term	Definition	Context
API	Application Programming Interface: A set of functions and protocols that allow software applications to communicate with each other and access features or data.	Used to describe how the Frontend (React) interacts with the Backend (Flask).
Cache	A mechanism for storing data temporarily to reduce access time and improve system performance by storing frequently used data.	Backend (Flask) caches data to improve response times for data retrieval.
CSV	Comma-Separated Values: A file format used to store tabular data in which each line represents a row, and each value is separated by a comma.	Frontend (React) sends a CSV file to the Backend (Flask) for processing.
Database (MySQL)	A structured set of data stored electronically in a computer system, often managed by a database management system (DBMS) like MySQL.	The Backend (Flask) stores electricity consumption data in the Database (MySQL).
Frontend (React)	A React component responsible for displaying the main graph and providing controls for data selection.	Renders a graph of electricity consumption, allowing users to select different parameters.
LineGraph.js	A React component that uses Chart.js to render a line graph to visualize electricity consumption data.	Displays the line graph for electricity consumption data on the user interface.
Machine Learning	A field of artificial intelligence that uses algorithms and statistical models to enable systems to improve their performance over time without explicit programming.	The system plans to integrate machine learning for future electricity consumption predictions.
MySQL	An open-source relational database management system that stores and manages structured data.	The Backend (Flask) connects to the MySQL database to store and retrieve data.
Prediction	The process of using models, such as machine learning models, to forecast future events or trends based on historical data.	Users will be able to click the “Predict” button for future electricity consumption estimates.
React.js	A JavaScript library used for building user interfaces, particularly single-page applications, by creating reusable UI components.	Frontend (React) is built using React.js to render interactive components.
Scheduler (APScheduler)	A Python library used to schedule periodic tasks, such as refreshing data in the cache at specific intervals.	The Backend (Flask) uses APScheduler to refresh the data cache periodically.

CSV Upload Endpoint	An API endpoint used to accept and process CSV file uploads from the Frontend (React) to the Backend (Flask) for storage and processing.	The user uploads a CSV file through this endpoint for data processing and storage.

Introduction

Background

Efficient energy management is critical for facilities like campuses, where tracking and understanding electricity usage can lead to significant savings and environmental benefits. Many campus managers and engineers currently rely on a combination of manual methods and disparate software to track electricity usage, which can be time-consuming and ineffective. Our electricity tracker application is designed to provide a streamlined, user-friendly solution for tracking and analyzing campus electricity usage data. This application incorporates visual tools, including a heat map, graphical data representations, a calendar view, and detailed statistics, to allow users to monitor usage patterns, pinpoint high-consumption areas, and make data-driven decisions to reduce energy consumption. The current implementation of the tracker provides powerful data analysis tools, but our goal is to ensure the interface remains accessible to a broad user base, including non-technical facility managers and environmental specialists. The existing interface of many energy-tracking solutions can overwhelm users with raw data and lacks cohesive navigation, hindering its usability for same-day analysis and decision-making.

Overall, the project involved developing a real-time Electricity Consumption Tracker for SLB, designed to help engineers, managers, and executives monitor and analyze energy usage effectively. The team focused on creating a user-friendly interface, integrating a graph visualizer for clear data representation, and ensuring backend and frontend connectivity using technologies like React.js, Node.js, and SQL, as well as a heat map data feature, calendar data visualization, and report generation. The goal was to deliver a functional project aligned with client needs while balancing technical feasibility and usability.

Document Summary

The System Architecture section outlines both the static and dynamic components of our system's frontend and backend. The static system architecture diagram illustrates the organization of the application's components and their interactions, while the dynamic system architecture diagram shows how the application executes tasks, following a specific scenario to demonstrate the flow of data and user interactions.

The Data Storage Design section outlines the structure of our system's database, including the use of a MySQL database for storing electricity consumption data, along with caching mechanisms to improve performance. We provide an entity-relationship diagram for the database and explain how data is exchanged between the frontend and backend, as well as the methods employed to secure application data.

The Component Design section offers a detailed overview of the specific components of our system architecture. It includes both static and dynamic diagrams that break the system into three parts: User Interface, Application Logic, and Data Management.

The UI Design section showcases the user interfaces that users will interact with, providing explanations for the design decisions made in the layout and functionality of the application.

System Architecture

Introduction

Our application is a web-based electricity tracker designed to provide users with an interactive interface to visualize and analyze electricity consumption data. We are building our system using a layered architecture, as it offers significant flexibility for simultaneous development. Each layer is designed to operate independently, enabling faster development cycles and simplifying the process of maintaining or extending the system in the future. Since this project will be handed over to other engineers upon completion, we've prioritized creating a modular, scalable, and reusable design. For instance, if changes to the database or API are required down the line, those updates can be made within their respective layers without disrupting the entire system. This separation of concerns is a key strength of our architectural approach.

In the following sections, we provide an overview of our system. We begin with a static system architecture diagram (Figure 1), which illustrates the relationships between the major components of our system. This diagram provides a high-level understanding of how each part of our application works in harmony. Additionally, we include a dynamic diagram (Figure 2), which demonstrates how these components interact during specific tasks, offering a practical example of the system's behavior.

Our system architecture consists of four main components: the Frontend, which encompasses the user interface and data visualization; the Backend, which manages business logic, API endpoints, and data processing; the Database, which stores electricity consumption data in a structured format; and the Cache, which optimizes performance by reducing database load. Each of these components plays a critical role in ensuring that our application is efficient, user-friendly, and ready for future enhancements.

Static System Architecture

Our static system architecture was designed based on the requirements and preferences provided by our client, SLB, to ensure seamless integration with their existing infrastructure while addressing their need for real-time electricity visualization and analysis. We implemented a layered architecture, starting with the frontend, where users interact with an intuitive React.js interface to upload data, visualize electricity consumption trends, and access predictive insights. The backend, built with Flask, processes user inputs, manages APIs, and communicates with the data layer, which uses a MySQL database to securely store and retrieve electricity data. To enhance performance, we incorporated a caching mechanism with SimpleCache, reducing database load and improving response times. Security and reliability were prioritized to protect SLB's sensitive data, ensuring the system delivers accurate and secure energy insights in real-time.

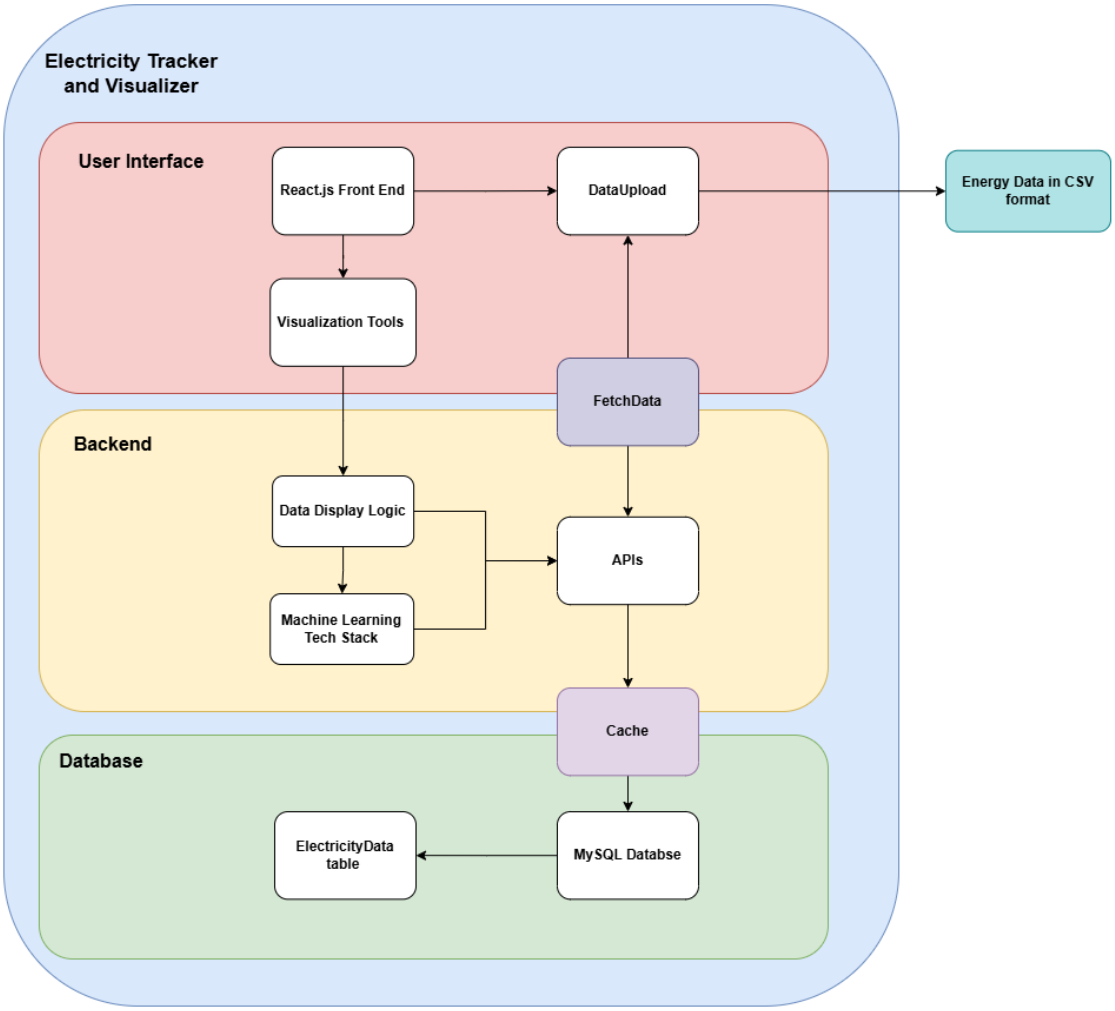

Figure 1 - Static System Architecture

Dynamic System Architecture

The dynamic system architecture diagram illustrates the flow of control and data within the electricity consumption tracker system. When a user uploads a CSV file, the Frontend (React) sends the data to the Backend (Flask), which processes it, stores it in the Database (MySQL), and caches it for quicker access. The user then interacts with the Frontend, selecting a building, year, and month to view the data, which is retrieved either from the Database or Cache and displayed in the LineGraph.js component using Chart.js.

The Backend periodically refreshes the cache to maintain performance, and in the future, machine learning predictions will be integrated when the user clicks the “Predict” button. This system ensures efficient, seamless interactions between the user interface, data processing, and storage, with optimized data retrieval and performance.

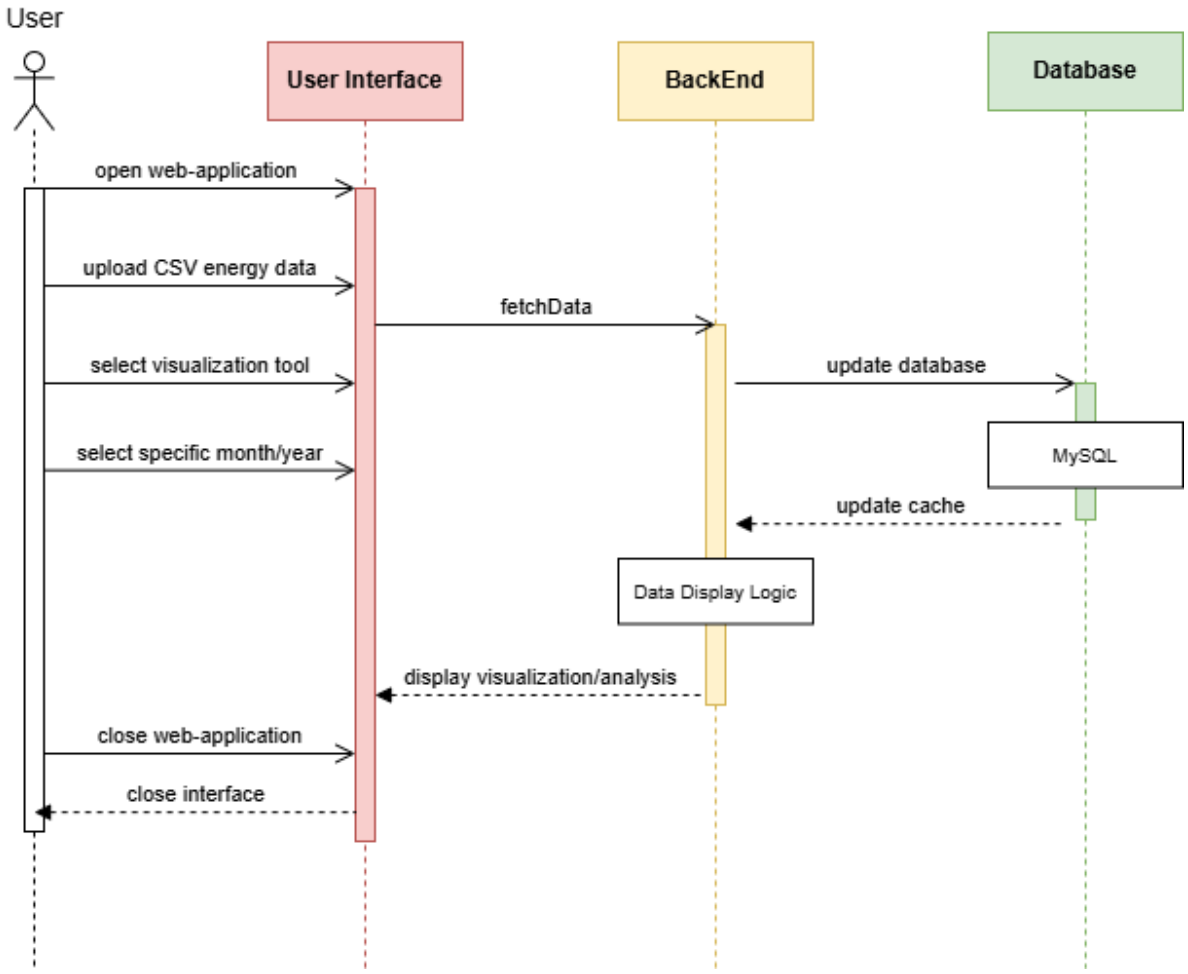

Figure 2- Dynamic System Architecture

Components Design

Introduction

Below are diagrams that help showcase the static and dynamic levels of our system architecture. The static design (Figure 3) showcases the direct realtionship between different components and classes. The design is also color-coded to show the three different components of the application. The green is the frontend application, the blue represents the backend of the application, and the orange signifies the data and ouputs the electricity usage of each building. We chose to use a sequence diagram (Figure 4) to show the dynamic level of our structure. The diagram starts at when the user inputs a csv file, and guides through the runtime of the backend interacting with the frontend. This shows the relationship and interaction between the frontend and backend of the application when the data is inputted. The two diagrams give a comprehensive view of how the system functions and how each application reacts with each other.

Static

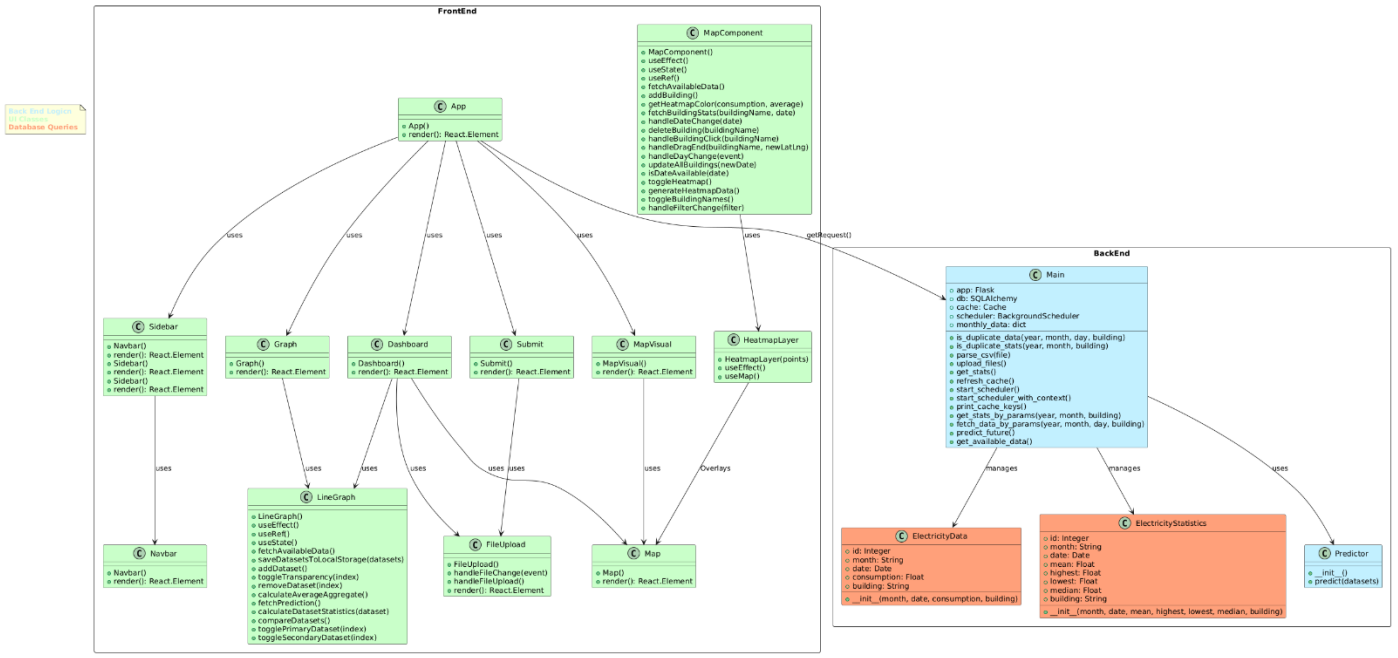

Figure 3 - Class Diagram

Dynamic

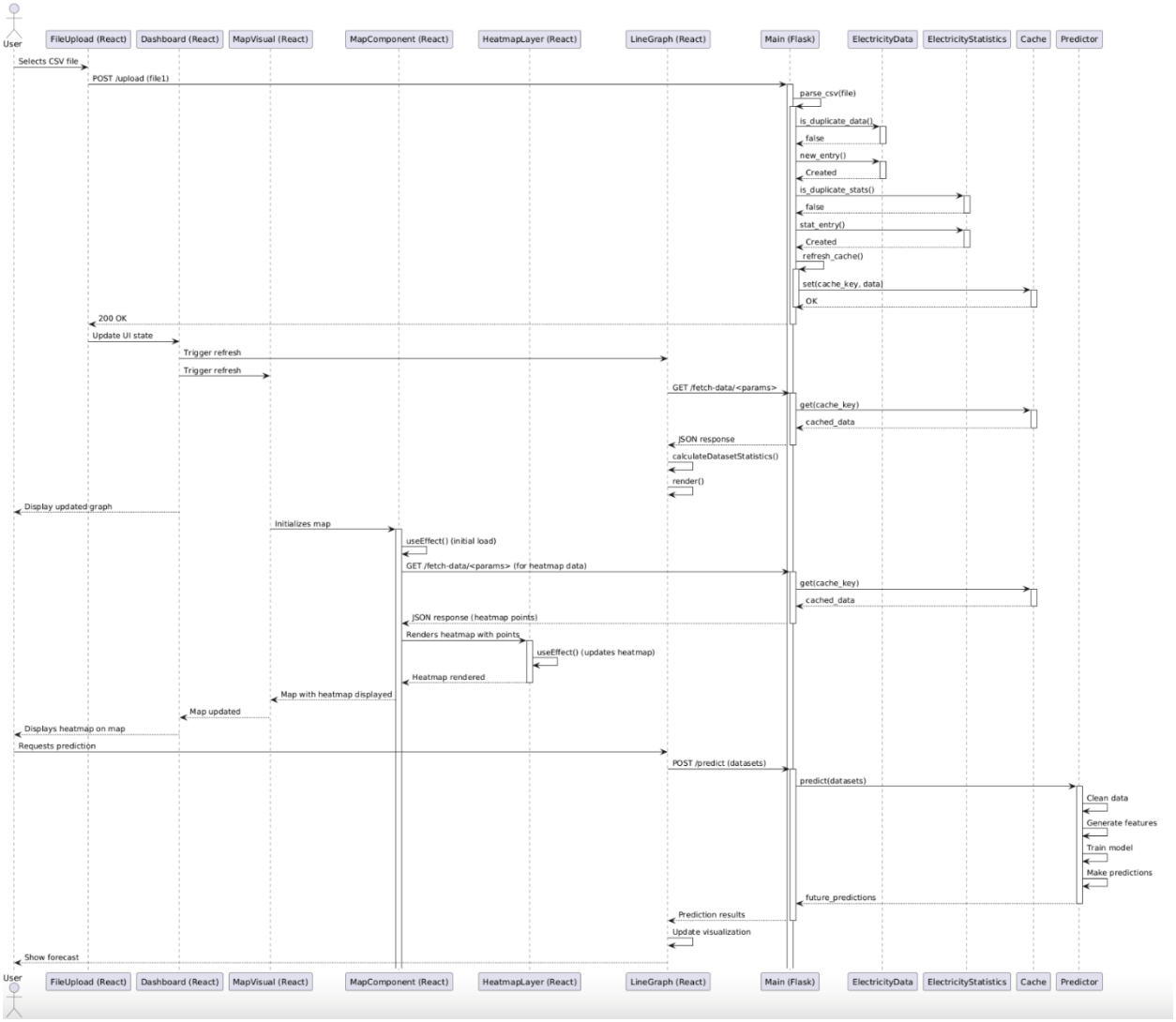

Figure 4 - Sequence Diagram

Data Storage Design

Introduction

The diagram below shows how we use a relational database to store and manage electricity consumption data. This method of data storage uses SQL and follows a structured schema with tables, primary keys, and foreign keys to define relationships between different entities. The data is stored in a MySQL database, which ensures data integrity and supports complex queries for data retrieval and manipulation.

We have a diagram representing our database structure and the relationships between tables. The filled-in diamonds represent the primary keys that uniquely identify each record in a table. The arrows indicate foreign key relationships, showing how tables are connected. The larger clear boxes represent the tables in our database, each containing columns that store specific attributes of the data.

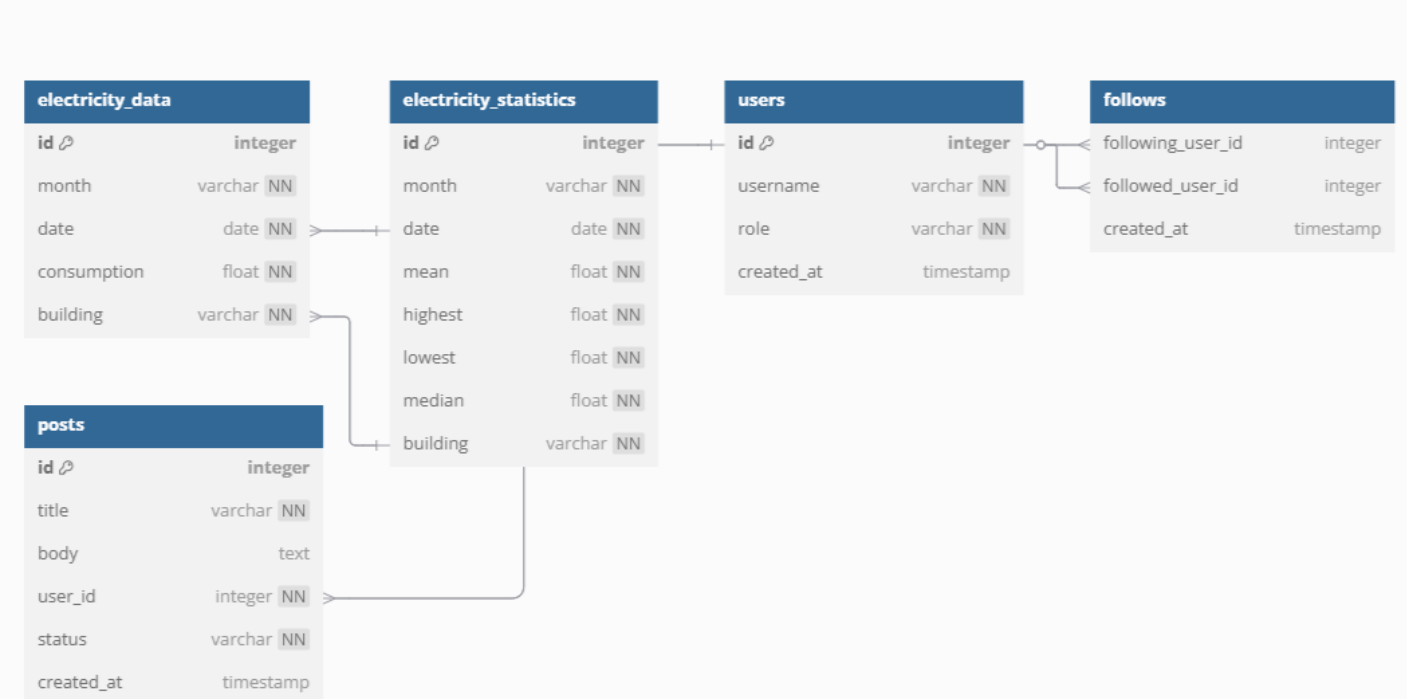

Figure 5 - Entity Relationship Diagram for Database

File Documentation

The CSV (Comma-Separated Values) file format is used to upload electricity consumption data into the application. Each CSV file contains data for a specific building and includes the following columns:

1. Group: This column contains information about the building group. The building number is extracted from this column using a regular expression.
2. Date Columns: These columns contain date and time information in the format MM/DD/YYYY HH:MM. Each date column represents a specific time interval (e.g., hourly) for which electricity consumption data is recorded.
3. Consumption Data: The rows under the date columns contain the electricity consumption values in kilowatt-hours (kWh) for the corresponding date and time.

Data Exchange

- CSV: The primary data format for uploading electricity consumption data is CSV. This format is used for its simplicity and ease of use.
- JSON: Data exchanged between the front end and backend of the application is typically in JSON (JavaScript Object Notation) format. JSON is used for its lightweight and easy-to-parse structure.
- HTTP/HTTPS: Data transfer between the frontend and backend uses HTTP/HTTPS protocols. HTTPS is preferred for secure data transfer to protect sensitive information.

Security Concerns

The primary piece of sensitive information contained within our application is the electricity consumption data for various buildings. This information is stored securely in a MySQL database with appropriate access controls. We also handle user authentication to restrict access to the application, ensuring that only authorized users can view and manage the data. User credentials are securely stored using hashing techniques, and we use HTTPS to encrypt data exchanged between the frontend and backend, preventing interception and tampering. While we do not store any medical or financial information, we are mindful of protecting the electricity consumption data and any personally identifiable information (PII) that may be associated with it. Additionally, we ensure that key pieces of data are securely transferred and stored, and we comply with relevant data protection regulations to address any privacy concerns.

UI Design

The dashboard provides an overview of electricity usage through customizable widgets. Users can view energy usage trends, heatmaps, calendar views, and reports. The drag-and-drop functionality allows users to rearrange widgets to suit their preferences.

Flexibility and Efficiency of Use:

- The drag-and-drop functionality supports both novice and expert users by allowing them to personalize the layout, improving task efficiency.

Aesthetic and Minimalist Design:

- The dashboard uses whitespace and grouping effectively, presenting multiple data types (graphs, maps, metrics) in a clean, non-overwhelming way.

Visibility of System Status:

- The alert box (e.g., unusual usage pattern alert on 10/9/24) immediately informs users of important system events in a visible, color-coded section.

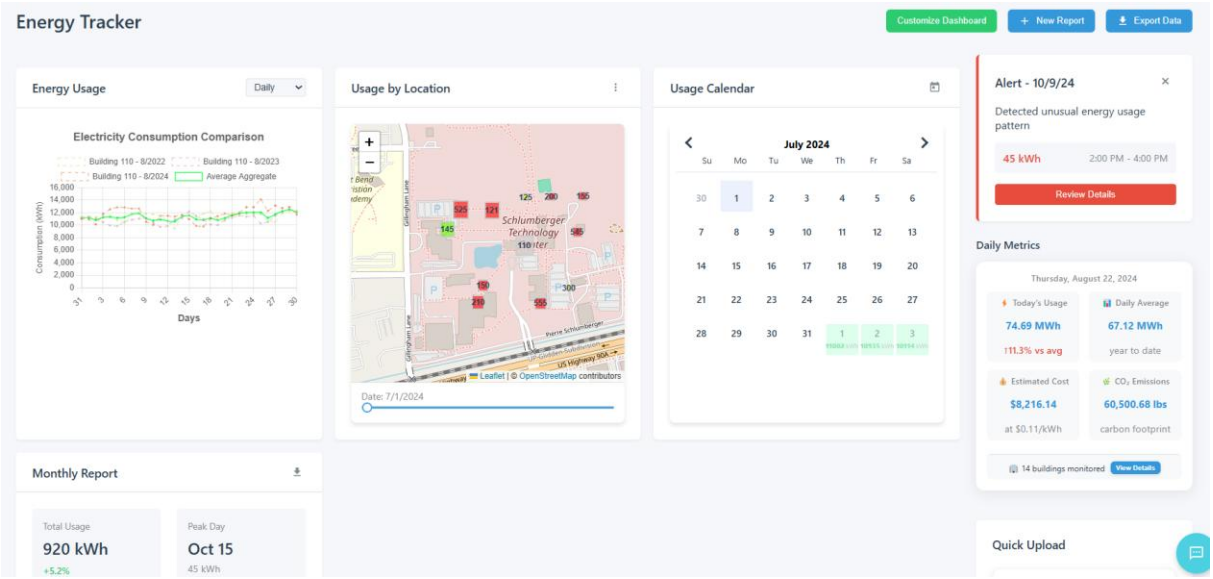

Figure 6 - Dashboard Page

The line graph feature allows users to compare electricity consumption across multiple datasets. Users can add datasets for specific buildings and months, calculate averages, and generate predictions for future usage.

Recognition Rather Than Recall:

- Users can select datasets from dropdowns and toggle between primary/secondary/average without remembering past inputs, enhancing usability.

Consistency and Standards:

- The graph uses standard chart conventions (e.g., color-coded lines, labeled axes), making interpretation intuitive and consistent across datasets.

Help Users Recognize, Diagnose, and Recover from Errors:

- Controls such as “Remove” buttons for each dataset allow users to quickly adjust or fix selections if incorrect data was chosen.

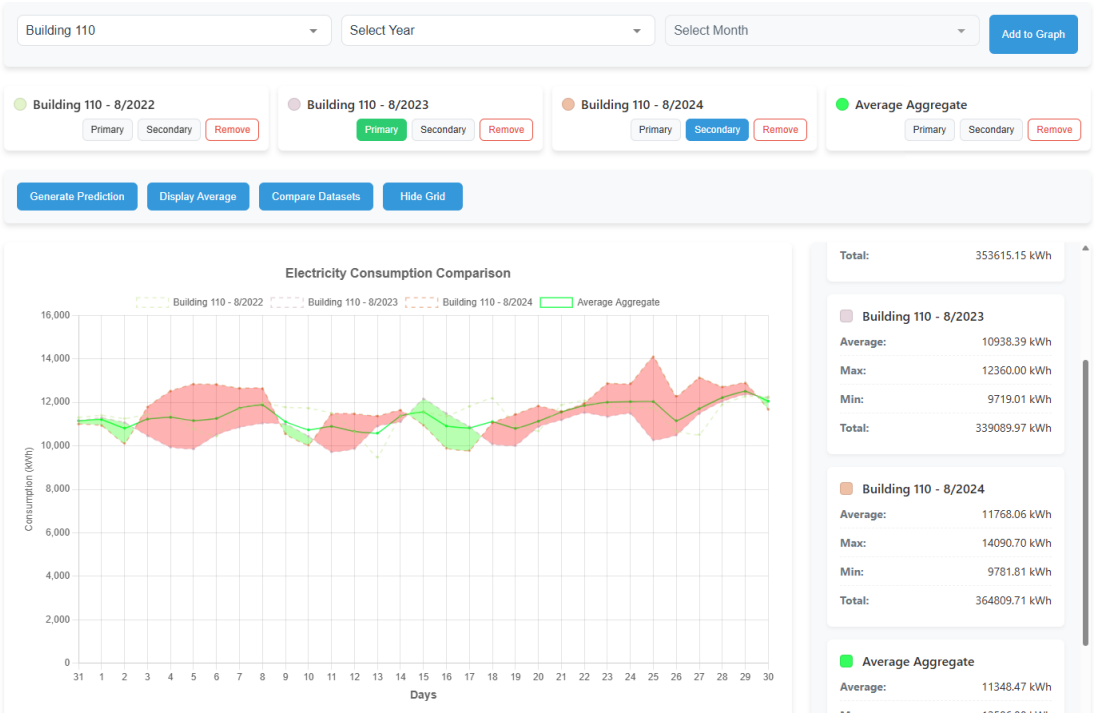

Figure 7 - Graph Visuals Page

The heatmap visualizes electricity usage across different locations. Users can identify areas with high or low consumption and analyze spatial trends.

Match Between System and the Real World:

- The map uses familiar real-world geographic layouts, with color-coded buildings to represent energy levels, aligning with users' natural understanding.

Error Prevention:

- Color indicators (red for high, green for low) prevent misinterpretation and guide the user to focus on critical areas needing attention.

User Control and Freedom:

- The user can easily select and deselect buildings and dates using dropdowns and sliders, enabling them to navigate without getting stuck in unwanted views.

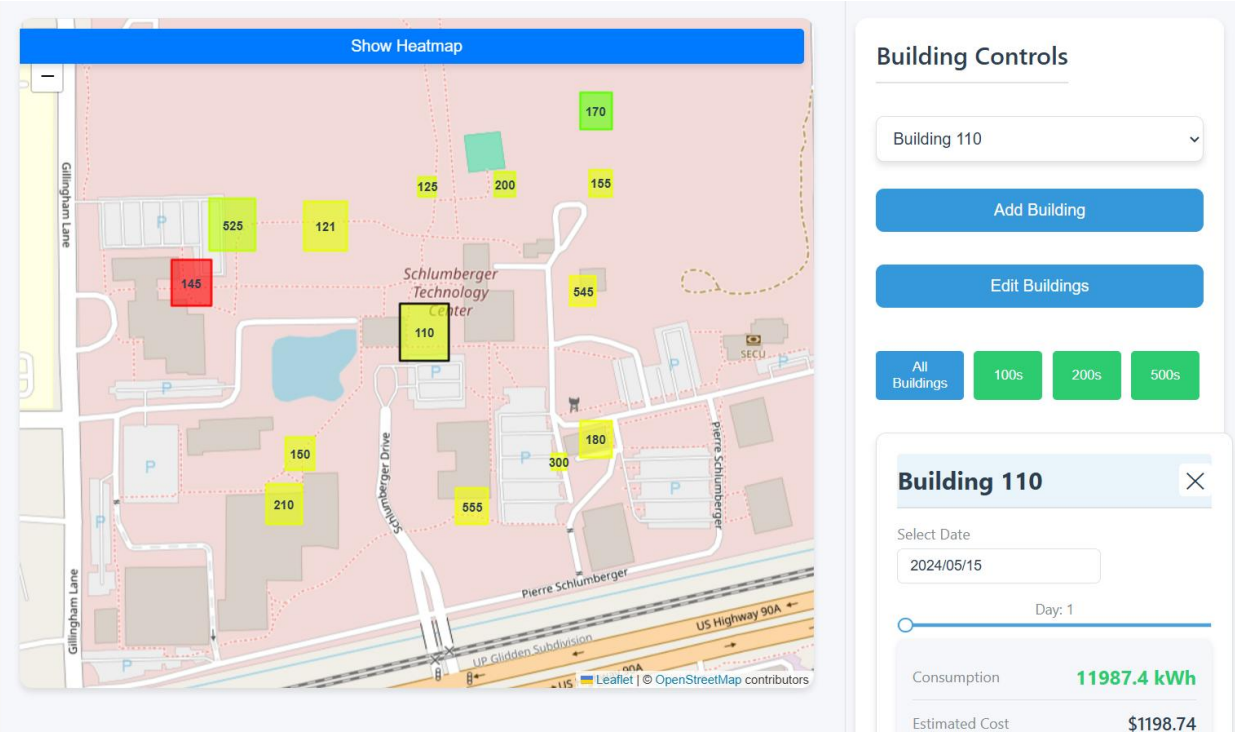

Figure 8 - Heat Map Page

The calendar view provides a monthly overview of electricity consumption. Users can select specific dates to view daily statistics and compare usage trends. The calendar also highlights days with unusually high or low consumption with red/green color respectively

Recognition Rather Than Recall:

- Color-coded calendar entries provide immediate recognition of abnormal consumption days without requiring memory.

Error Prevention:

- Highlighting high-usage days reduces the likelihood of users overlooking anomalies.

Consistency and Standards:

- The calendar follows standard monthly layouts and color conventions (e.g., red for alerts), ensuring user familiarity.

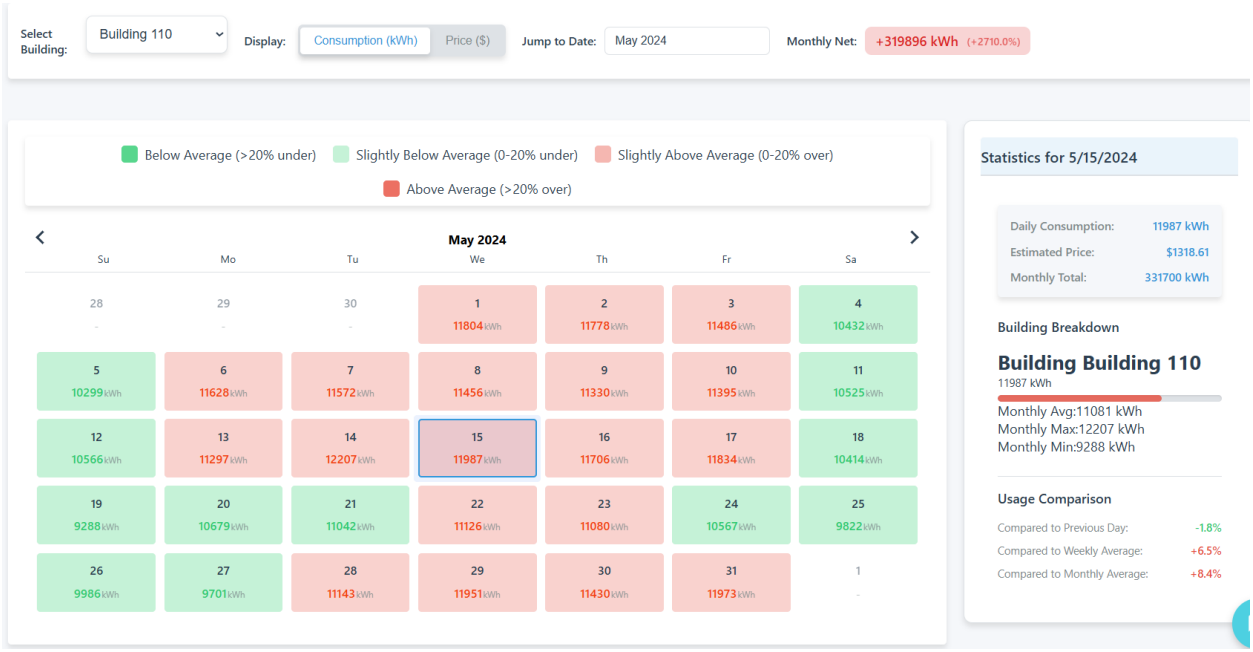

Figure 9 - Calendar Page

The report page offers detailed insights into electricity consumption for selected buildings and timeframes. Users can view metrics such as daily averages, monthly totals, and peak usage. The page also includes options to export reports as PDFs.

Help and Documentation:

- The inclusion of downloadable reports supports users who need external references or help with analysis.

Visibility of System Status:

- Clear display of metrics like cost, usage, and CO_2 emissions provides immediate insight into consumption levels.

Flexibility and Efficiency of Use:

- Advanced users can generate reports quickly using filters and export them in common formats like PDF.

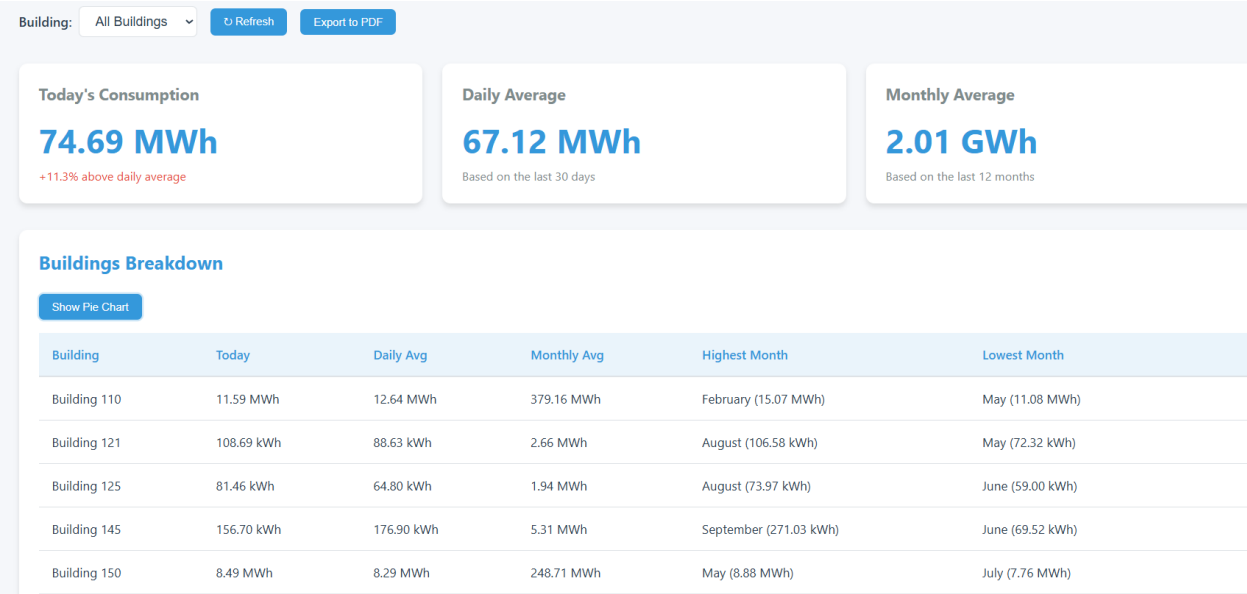

Figure 10 - Report Page

Appendix

API

Fetch Available Data:

Endpoint: GET /get-available-data

Description: Retrieves a list of available buildings and their associated data.

Response: JSON object with building names and metadata.

Fetch Consumption Data:

Endpoint: GET /fetch-data/{year}/{month}/{day}/{building}

Description: Retrieves electricity consumption data for a specific building and date.

Response: JSON object with daily consumption values.

Fetch Statistics:

Endpoint: GET /stats/{year}/{month}/{building}

Description: Retrieves monthly statistics (average, peak, lowest) for a building.

Response: JSON object with statistical metrics.

Example JSON Response: (For GET /stats/2025/3/Building%20110)


```
{
  "month": "March",
  "date": "2025-03-01",
  "mean": 120.45,
  "highest": 200.75,
  "lowest": 80.30,
  "median": 115.00,
  "building": "Building 110",
  "highestMonth": "March",
  "lowestMonth": "March"
```

}

CSV Format

Example format: “Group, Other Metadata ,01/01/2025 00:00,01/01/2025 01:00,01/01/2025 02:00

Building 110, Some Info,45.6,78.2,32.1”