AN INTELLIGENT ELECTRICITY MANAGEMENT UNIT: AI-DRIVEN POWER FORECASTING AND PERSONALIZED CONSUMPTION INSIGHTS WITH APPLICATION INTEGRATION

Pivithuru N.H.A.S.
IT21389160

B.Sc. (Hons) Degree in Information Technology Specializing in Information Technology

Department of Information Technology
Sri Lanka Institute of Information Technology
Sri Lanka

August 2025

POWER MONITORING PROTAL – INTELLIGENT ENERGY MONITORING, VISUALIZATION, AND ANALYTICS DASHBOARD

Pivithuru N.H.A.S.

IT21389160

The dissertation was submitted in partial fulfilment of the requirements for the Bachelor of Science (Hons) in Information Technology specializing in Information Technology

Department of Information Technology

Sri Lanka Institute of Information Technology

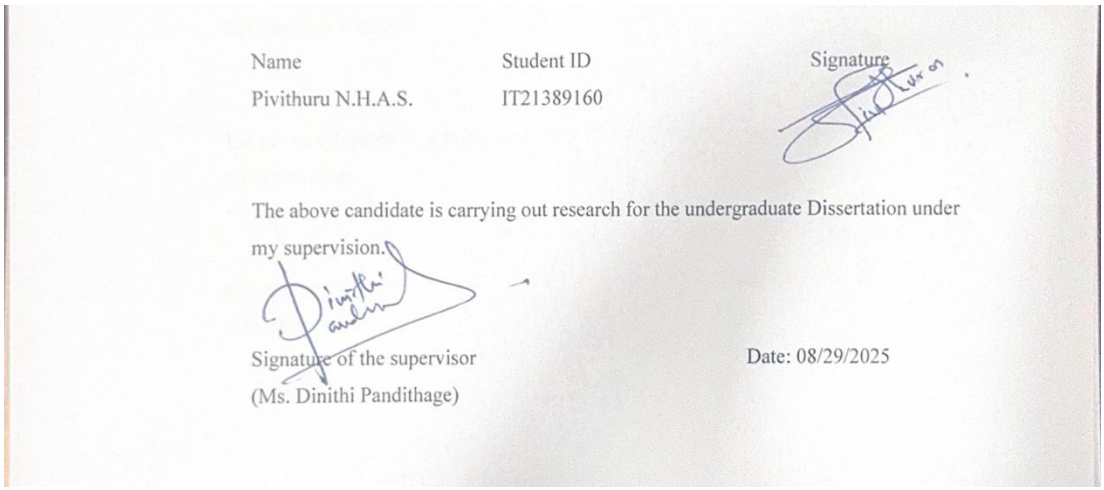
Sri Lanka

August 2025

DECLARATION

"I declare that this is my own work, and this proposal does not incorporate without acknowledgement any material previously submitted for a degree or diploma in any other university or Institute of higher learning and to the best of my knowledge and belief it does not contain any material previously published or written by another person except where the acknowledgement is made in the text.

Also, I hereby grant to Sri Lanka Institute of Information Technology, the nonexclusive right to reproduce and distribute my dissertation, in whole or in part in print, electronic or other medium. I retain the right to use this content in whole or part in future works (such as articles or books)."



Name	Student ID	Signature
Pivithuru N.H.A.S.	IT21389160	

The above candidate is carrying out research for the undergraduate Dissertation under my supervision.

Signature of the supervisor

(Ms. Dinithi Pandithage)

Date: 08/29/2025

ABSTRACT

The increasing demand for energy-efficient solutions has emphasized the need for real-time monitoring and intelligent visualization in modern energy management systems. This research presents the development of a web-based energy data visualization and analytics platform that integrates Internet of Things (IoT) devices with Amazon Web Services (AWS) cloud infrastructure to provide actionable insights into household and device-level energy consumption. The system enables real-time monitoring, forecasting of electricity bills using time-series models, and detailed device-wise cost breakdowns through an interactive React-based dashboard powered by a Laravel backend. Real-time data is transmitted via WebSocket communication and processed for both current and predictive analysis, enhancing decision-making and cost awareness. Furthermore, the inclusion of accessibility features—such as color-blind-friendly themes for users with protanopia and deuteranopia—addresses inclusivity gaps found in most existing platforms. Evaluation results indicate that the proposed solution accurately visualizes energy data, enhances user engagement, and supports sustainable energy consumption through actionable insights.

Keywords: IoT, Energy Management, Data Visualization, Forecasting, Accessibility, WebSocket, Cloud Computing, AWS, Smart Home, Laravel, React.

ACKNOWLEDGEMENT

I would like to express my sincere gratitude to all those who supported and guided me throughout the course of this research project.

First and foremost, I extend my heartfelt thanks to my supervisor Ms. Dinithi Pandithage and co supervisor Mr. Ashvinda Iddamalgoda, for their invaluable guidance, constructive feedback, and continuous encouragement at every stage of the research. Their expertise and insights were instrumental in shaping the direction and outcome of this work.

I am also grateful to the academic staff of the Department of Information Technology, SLIIT, for providing the resources and knowledge foundation necessary to successfully complete this project. Special thanks are due to my colleagues and peers, whose discussions, suggestions, and collaboration helped refine many aspects of the work.

I would also like to acknowledge my family for their unwavering support, patience, and motivation during this journey. Their encouragement gave me the strength to stay focused and dedicated.

Finally, I wish to thank everyone who directly or indirectly contributed to the success of this research. Without their support and cooperation, the completion of this project would not have been possible.

Table of Contents

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

ABBREVIATION	DESCRIPTION
IoT	Internet of Things
ML	Machine Learning
LLM	Large Language Model
HEM	Home Energy Management

INTRODUCTION

Background and Literature Survey

Smart homes and energy management systems have become an important field of research in recent years. The increasing popularity of the Internet of Things (IoT) has enabled homes, buildings, and industries to adopt connected devices that can sense, process, and communicate data in real time. These technologies not only improve comfort and convenience but also play a critical role in reducing unnecessary energy consumption and promoting sustainability. This section reviews the existing literature in the domain of IoT-based smart home applications, energy optimization strategies, mobile applications for smart ecosystems, and energy management platforms.

Several studies have explored this area in different ways. For example, Prathyusha and Bhowmik [1] presented an IoT-based smart home system with integrated energy management features. Their work highlighted that interconnected smart devices have become an essential part of everyday life, offering convenience, safety, and energy efficiency. However, they also identified a challenge: while smart appliances improve quality of life, they can sometimes add to the total energy costs if not managed properly. To address this, their research proposed a model where each room in a smart home is equipped with selected smart devices, and energy consumption is tracked and analyzed. The benefit of this work is that it emphasizes the importance of room-level monitoring and the ability to simulate energy use in order to identify opportunities for saving. This aligns directly with the broader goal of reducing waste and encouraging sustainable practices in modern households.

In another important study, Khan and Mouftah [2] investigated energy optimization and management in smart homes by using web services as the communication framework. Their proposed approach assumed a smart home equipped with a wireless

sensor network, based on Zigbee technology, and connected to a central computer. They designed a system where users could remotely interact with smart home elements such as sensors, appliances, HVAC systems, and thermostats. The central computer acted as the control hub, and Business Process Execution Language (BPEL) was used to implement web services. One of the most innovative aspects of their research was the introduction of an algorithm to sell energy back to the grid, allowing households to benefit financially when excess energy was available. They also applied dynamic programming techniques to optimize energy usage. Their findings demonstrated the strong potential of web services in providing flexible, scalable communication between users and smart home devices. This research provided a foundation for integrating smart homes into smart grid environments, making energy systems more interactive and responsive. [6]

The importance of user interaction in smart home systems was addressed by Argyros et al. [3]. Their research focused on designing a mobile application that enables users to interact with smart home ecosystems more effectively. Unlike some earlier systems that were tightly bound to specific devices or architectures, their application was designed to adapt to any smart home configuration, regardless of the hardware used. To achieve this, they followed a user-first design thinking methodology, which placed the user experience at the center of the development process. The application allowed users to monitor sensors and devices, define personalized routine preferences, and view data related to energy consumption. Furthermore, the team tested their application in a real-world environment known as the Ambient Assisted Living (AAL) House. To evaluate acceptance and performance, they conducted surveys using questionnaires. The results suggested that end users appreciated the flexibility and usability of the application. This study is significant because it demonstrates the value of designing smart home systems from a user perspective rather than focusing only on technical capabilities.[7], [8]

Another contribution in this field came from Chen Li, Logenthiran, and Woo [4], who developed a mobile application for smart home energy management known as

iSHome. Their work was conducted in Singapore, which has ambitions to become a smart nation with widespread adoption of smart homes and smart buildings. The researchers developed two versions of the iSHome application. The first version could communicate with smart power plugs using Bluetooth, while the second version was able to interact with a smart home management server. Both versions were implemented on Android using the Eclipse development environment. Although the system did not include advanced energy-saving algorithms at that stage, it still demonstrated a reduction in energy wastage through improved monitoring and control. The authors emphasized that smart homes would not only need to interact with devices inside the home but also with the larger power grid. This approach shows how smart home applications can contribute to a sustainable and efficient electricity system at a national scale.

Finally, Stolojescu-Crisan and Gal [5] explored the design of a home energy management system in the context of global climate change and the urgent need for energy reduction. Their work introduced a monitoring system that forms part of a larger smart home automation platform called qToggle. The key contribution of this research is the emphasis on next-generation energy saving solutions, where monitoring is not just about displaying usage data but about enabling actionable decisions. Their system could be remotely accessed and controlled using an Android application, giving users the flexibility to monitor their household energy consumption anytime and anywhere. The authors also positioned their work within the broader discussion of climate change, noting that energy management systems are essential tools in helping households transition toward renewable energy adoption and more responsible energy usage. [11]

Taken together, these studies reveal both the progress and the gaps in the area of smart home energy management. Prathyusha and Bhowmik [1] provided detailed insights into room-level modeling and simulation of energy usage, while Khan and Mouftah [2] demonstrated the value of integrating homes with the smart grid through web services. Argyros et al. [3] highlighted the importance of user-friendly mobile

applications that adapt to diverse environments, and Chen Li et al. [4] showed the potential for Bluetooth-based and server-based mobile applications in controlling smart appliances. Meanwhile, Stolojescu-Crisan and Gal [5] emphasized energy monitoring as part of a wider global challenge of climate change. [10], [11]

Despite these contributions, there are still important areas for improvement. Many of the existing systems focus on monitoring and basic control but do not combine advanced forecasting with real-time analytics. Some applications are limited to specific communication technologies such as Bluetooth or Zigbee, which can restrict scalability. Moreover, accessibility is rarely considered; for example, features like color-blind-friendly interfaces are missing in most platforms. There is also a lack of solutions that provide device-level cost breakdowns alongside predictive billing logic. These gaps highlight the need for a more inclusive, scalable, and intelligent visualization system that not only tracks real-time energy use but also forecasts future consumption, provides accessibility features, and generates reports that empower both residential and commercial users to make informed decisions.

Research gap

Although significant research has been conducted on smart home energy management systems (SHEMS), several crucial limitations remain unaddressed. The reviewed literature demonstrates that existing systems have made important contributions in terms of monitoring, optimization, and user interaction, but there are still gaps that limit their effectiveness, inclusivity, and long-term usability. These gaps are discussed below in detail.

1. Forecasting Integration

Most existing systems either focus on monitoring real-time energy consumption or on providing remote control of devices. For example, Prathyusha and Bhowmik [1]

developed an IoT-based smart home environment that models household rooms with smart appliances and calculates their energy consumption. Their system successfully estimates energy usage but does not extend to predictive analysis of future bills or consumption. Similarly, Khan and Mouftah [2] proposed a smart grid-enabled system that allowed remote device interaction and even implemented an algorithm to sell energy back to the grid. While their work advanced optimization using dynamic programming, it remained centered on present energy states without providing AI-driven forecasting of future consumption patterns. This reveals a gap where real-time monitoring exists in isolation, while forecasting models—though present in academic research—are rarely integrated into user-facing applications. Without this integration, users only see what is happening now but cannot plan ahead or make proactive decisions about upcoming energy costs.

2. Device-Level Insights

Another notable limitation lies in the granularity of energy data provided to users. Several studies, such as the mobile application by Argyros et al. [3], have focused on enabling users to view overall energy usage, set preferences, and monitor devices. However, in most cases, energy data is presented at a household or aggregated level, rather than broken down per appliance. As a result, users may know the total energy cost of their home but lack the ability to pinpoint which devices are consuming the most energy. In modern households where appliances like refrigerators, washing machines, and air conditioners have very different consumption patterns, this limitation reduces the usefulness of the system. Chen et al. [4] designed iSHome, a mobile application that could interact with power plugs and a smart home management server, but it primarily emphasized device control rather than detailed consumption analytics. The lack of fine-grained, device-wise energy analysis means that families cannot effectively identify high-energy-consuming appliances, schedule loads

efficiently, or replace inefficient devices. This gap highlights the need for systems that go beyond aggregate data and deliver actionable, device-level insights.

3. Accessibility and Inclusive Design

A further research gap lies in the lack of accessibility features in most energy management platforms. While the reviewed studies [1–5] make significant contributions in terms of system functionality and optimization, none of them directly address the needs of users with accessibility challenges such as color vision deficiencies. For instance, dashboards and mobile applications often use colors as the primary medium for displaying energy trends and charts. This creates barriers for users with protanopia or deuteranopia, who may struggle to distinguish between certain color-coded elements. Accessibility is rarely considered a core design requirement in the reviewed systems, which instead prioritize technological functions. However, for an energy management platform to truly serve diverse households, it must adopt inclusive design practices such as color-blind-friendly themes, alternative data representations, and customizable visualization modes. The absence of such features in current research reveals a substantial gap in terms of usability for all users.

4. User Engagement and Interpretability

Several systems reviewed demonstrate strong technical foundations but fall short in terms of user engagement and interpretability. For example, the system presented by Stolojescu-Crisan and Gal [5] emphasizes energy monitoring within a smart home automation environment, with remote control supported through an Android application. While technically sound, the focus remains on data display rather than on ensuring that data is easily understood by non-technical users. Similarly, Khan and Mouftah [2] provided sophisticated energy optimization mechanisms but did not prioritize user-friendly interfaces that translate technical information into actionable

recommendations. As Argyros et al. [3] highlight, end-user experience is crucial for adoption. Yet, most studies emphasize device connectivity and control rather than intuitive visualization. This gap underlines that visualization is not just about presenting data, but about making that data meaningful, engaging, and capable of driving behavioral change.

5. Report Generation and Long-Term Usability

Another underexplored area is structured report generation. Most existing solutions, such as those by Prathyusha and Bhowmik [1], Argyros et al. [3], and Chen et al. [4], focus on real-time display of energy data through dashboards or mobile applications. However, few provide users with the ability to generate downloadable reports (e.g., monthly summaries, device-level comparisons, or long-term consumption records). Such reports are essential for tasks like budgeting, tracking progress over time, or sharing data with landlords, auditors, or energy providers. The lack of this feature confines users to viewing data only within the app interface, reducing the long-term value of the system. Structured reporting could transform these platforms from monitoring tools into decision-support systems. Its absence marks a significant research gap.

Area	Findings from Literature	Identified Gap
Forecasting Integration	Most works [1][2][4][5] focused on monitoring or controlling energy usage in real time. Forecasting was rarely combined with visualization.	Lack of systems that integrate AI-driven forecasting with real-time dashboards for better planning.
Device-Level Insights	Existing studies often consider overall household consumption [1][2][5]. Device-wise analytics is either missing or very limited.	Users cannot identify which specific devices consume the most energy.
Accessibility (Color Blindness)	Reviewed systems [3][4][5] did not address inclusivity for visually impaired or color-blind users.	Lack of accessible dashboards (example: color-blind-friendly themes).
User Engagement & Visualization	Some applications provide raw data or control features [2][4], but with limited focus on user-friendly visualization.	Absence of clear, actionable visualizations that ordinary users can easily understand.
Report Generation	Many systems show consumption data on screen only [1][3][5]. Structured reporting features are rare.	No support for downloadable reports for tracking, analysis, or sharing.

Table 1. 1: Summary of Research Gaps in Existing Energy Management Systems

Feature / Aspect	[1]	[2]	[3]	[4]	[5]	Proposed System
Real-time Monitoring	✓	✓	✓	✓	✓	✓
AI-based Forecasting	✗	✗	✗	✗	✗	✓
Device-level Analytics	✗	✗	✗	✗	✗	✓
Accessibility (Color-blind support)	✗	✗	✗	✗	✗	✓
Report Generation	✗	✗	✗	✗	✗	✓
Cloud Integration (AWS, scalable)	✗	✗	✗	✗	✗	✓

Table 1. 2: Comparison of Existing Research and Identified Gaps

In summary, the literature reveals several interrelated gaps:

(i) limited integration of real-time monitoring with forecasting capabilities [1,2]
(ii) insufficient focus on device-level energy analytics [3,4]
(iii) absence of accessibility and inclusive visualization features across platforms [1–5]
(iv) weak emphasis on user engagement and interpretability of data [2,3,5], and
(v) lack of structured report generation for long-term usability [1,3,4]. Addressing these shortcomings would enable the development of a more advanced energy visualization and analytics system—one that is predictive, device-aware, inclusive, user-friendly, and supportive of long-term decision-making.

Research Problem

The rapid growth of the Internet of Things (IoT) and smart home technologies has enabled the development of numerous energy management systems aimed at reducing household electricity consumption and improving efficiency. Systems such as those proposed by Prathyusha and Bhowmik [1] and Stolojescu-Crisan and Gal [5] demonstrate that smart environments can successfully monitor consumption patterns and provide users with real-time data on household energy usage. Similarly, Khan and Mouftah [2] extended the functionality of smart grid-based systems to include energy optimization algorithms, while Argyros et al. [3] and Chen et al. [4] presented mobile applications that allow users to interact with smart devices in their homes. Despite these advancements, a major problem remains: users still lack an inclusive and user-friendly platform that integrates real-time monitoring with forecasting, device-level insights, meaningful visualizations, and accessibility features.

This problem manifests in several practical challenges that affect both usability and effectiveness:

1. Inability to Forecast Energy Costs

Most existing platforms are limited to displaying current consumption without projecting future energy usage or costs. For instance, Prathyusha and Bhowmik [1] developed a system capable of estimating consumption patterns, but their work stopped short of forecasting future bills. Similarly, Khan and Mouftah [2] optimized consumption through smart grid communication but did not provide predictive tools to help users plan financially. Without forecasting, households are left unaware of their likely expenses until bills arrive, making proactive energy-saving decisions more difficult. This lack of predictive insight reduces the overall utility of existing systems, especially for users who want to budget effectively.

2. Difficulty in Identifying Device-Level Consumption

Another pressing issue is that many systems present energy data at an aggregate household level. While this provides a general overview, it does not help users identify which specific appliances are driving consumption. Argyros et al. [3], for example, enabled users to monitor sensors and devices through a mobile app, but the emphasis remained on overall household monitoring rather than detailed per-device analytics. Chen et al. [4] introduced applications capable of controlling devices via Bluetooth or through a home management server, but once again, the focus was on control rather than analysis. In practice, this makes it hard for users to identify high-consumption appliances like refrigerators or air conditioners, which could otherwise be replaced, scheduled, or used more efficiently.

3. Lack of Accessibility and Inclusive Features

Accessibility is a major issue that has been largely overlooked in energy management research. None of the reviewed studies [1–5] specifically address the needs of users with accessibility challenges, such as individuals with color vision deficiencies. Most dashboards and mobile applications rely heavily on color-coded graphs and charts to communicate information, which makes them difficult to interpret for users with protanopia or deuteranopia. In an era where inclusivity is central to technology design, the absence of such features highlights a critical shortcoming. Without accessibility-focused adaptations—such as color-blind-friendly themes, customizable visualizations, or alternative data formats—existing systems risk excluding a significant group of users.

4. Limited Engagement and Actionability of Data

Even when data is presented, it is often not in a format that ordinary users can easily understand or act upon. Stolojescu-Crisan and Gal [5] presented qToggle, a system that successfully integrates monitoring with remote control, but their focus remained on technical implementation rather than on ensuring user interpretability of the data. Likewise, Khan and Mouftah [2] introduced advanced optimization features but did not simplify the results for end-users in a non-technical format. As Argyros et al. [3] point out, user-centered design is critical for adoption, yet many existing systems fall short in translating numbers into meaningful insights or actionable recommendations. This gap leaves users with raw figures but little guidance on how to change behavior for cost or energy savings.

5. Absence of Structured Report Generation

Finally, most systems are designed to display real-time energy usage through dashboards or mobile interfaces, but they lack functionality for structured report generation. Prathyusha and Bhowmik [1], Argyros et al. [3], and Chen et al. [4] all presented applications that provide real-time data display, but none offered tools to generate monthly summaries, device-level comparisons, or long-term tracking reports. Such features are critical for households that want to evaluate their progress over time, compare consumption across different periods, or share structured data with landlords, auditors, or energy providers. Without reporting capabilities, users are confined to short-term snapshots of data and cannot use the system as a long-term decision-support tool.

In conclusion, while the literature demonstrates substantial progress in smart home energy management, existing systems fall short of delivering a comprehensive, inclusive, and user-friendly platform. The main problem is that current solutions do not combine real-time monitoring with forecasting, device-level insights, accessible visualizations, and structured reporting. As a result, users face the following challenges:

- They cannot clearly see or plan for future electricity costs.
- They struggle to identify which devices are consuming the most energy.
- Color-blind users and others with accessibility needs cannot fully engage with dashboards.
- Numerical data is presented without sufficient interpretability or recommendations for action.
- Long-term usability is limited due to the absence of structured report generation.

Addressing these challenges requires a next-generation platform that integrates forecasting, device-level tracking, inclusive design, and actionable visualizations to empower users to make informed energy decisions.

Research Objective

The primary objective of this research is to design and implement an intelligent, accessible, and user-friendly web-based energy management dashboard that integrates real-time monitoring, predictive analytics, and device-level insights for households, bridging existing research gaps by offering comprehensive analytics, inclusive accessibility features, and forecasting capabilities beyond traditional aggregate consumption or basic device control solutions.

1. To design and implement a web-based dashboard that visualizes real-time and forecasted electricity usage data at both household and device levels.

The dashboard will act as the central user interface where data collected from IoT devices is presented in an interactive and visually engaging format. Unlike traditional monitoring systems that display only total household consumption, the proposed dashboard will include both real-time readings and AI-driven forecasts. This enables users not only to monitor what is happening now but also to anticipate future consumption trends and costs. At the household level, users will be able to see their overall usage patterns, while at the device level, they can analyze the contribution of specific appliances (such as refrigerators, washing machines, or air conditioners). Forecast integration will allow families to plan their energy usage in advance, which is particularly useful for managing monthly budgets and avoiding unexpected high bills.

2. To apply mathematical models for bill calculation based on local electricity pricing slabs, using both actual consumption and predicted values.

Energy bills in most regions are calculated using slab-based pricing models where higher consumption is charged at higher rates. A major limitation in existing dashboards is that they only show kilowatt-hour (kWh) usage without translating this into actual monetary costs under local tariff rules. This research will implement mathematical algorithms that dynamically compute costs based on real-time usage and forecasted consumption. By simulating the local tariff structures, the system will provide users with an estimated bill projection. This will empower households to manage their energy usage more effectively, since users will not only see their current usage in units but also understand the financial implications of continuing at their present rate of consumption.

3. To create device-level analytics, including cost so far, estimated bill, average usage, and threshold monitoring.

Many households contain multiple appliances that consume electricity at vastly different rates, yet existing platforms often aggregate all data into a single household figure. This makes it difficult for users to identify which devices are responsible for higher costs. The proposed system will therefore focus on device-level granularity, providing analytics such as:

- Cost so far: The accumulated financial cost of running a device up to the current time.

- Estimated bill contribution: A projection of how much each device will add to the total bill if current trends continue.
- Average usage analysis: Identifying peak hours and average energy demand per device.
- Threshold monitoring: Allowing users to set personalized consumption thresholds and receive alerts when a device approaches or exceeds this limit.

By presenting this information in clear dashboards and visualizations, users will be able to identify energy-hungry devices and make informed decisions such as turning off unnecessary loads or replacing inefficient appliances.

4. To integrate data sources from IoT devices and cloud databases (DynamoDB, RDS, Athena) using real-time Web Sockets.

A technically challenging objective of this research is to develop an infrastructure that supports real-time data collection and transmission. IoT devices deployed within the household will record energy parameters (such as voltage, current, power, and energy consumption), which will be transmitted to the cloud. The backend will leverage AWS cloud services including DynamoDB for real-time storage, RDS for relational queries, and Athena for analytical queries over historical data. To ensure low-latency updates, the dashboard will connect to the cloud through Web Sockets, allowing immediate streaming of new values to the user interface without the need for manual refreshes. This integration will ensure the system remains scalable, responsive, and suitable for multi-user environments, which is crucial for modern smart home ecosystems.

5. To implement accessibility support for color-blind users by providing protanopia and deuteranopia-friendly themes.

Accessibility remains an often-overlooked area in energy monitoring platforms. Individuals with color vision deficiencies, such as protanopia (difficulty distinguishing red tones) and deuteranopia (difficulty distinguishing green tones), often face challenges interpreting standard color-coded charts and graphs. To address this gap, the dashboard will provide alternative visual themes that adapt color palettes to ensure clarity for these user groups. Tailwind-based styling and conditional theming will be applied, ensuring that all charts, alerts, and visual elements are legible and meaningful regardless of visual impairment. This not only promotes inclusivity but also aligns with universal design principles, making the system accessible to a wider demographic of users.

6. To enable report generation so users can download structured summaries of their consumption.

While dashboards are useful for interactive monitoring, users often need to retain records of their energy usage for long-term comparisons, billing disputes, or household management. Many existing systems fail to provide a convenient method for report exporting. This research aims to address this by implementing automated report generation in formats such as PDF or Excel, summarizing both household-level and device-level insights. Reports will include graphs, usage statistics, forecast data, and cost breakdowns, enabling users to keep an offline record or share with relevant stakeholders (e.g., utility companies or household members). This feature enhances the practical usability of the system and ensures that data is not limited to on-screen viewing.

7. To provide interactive charts (power-time, current-time) for better understanding of usage patterns.

Data visualization is central to this research. Instead of static tables or charts, the system will feature interactive time-series graphs that plot parameters such as power vs. time and current vs. time. These charts will allow zooming, filtering, and time range selection, enabling users to identify daily peaks, unusual spikes, and seasonal variations. Interactivity transforms raw data into actionable insights, making it easier for non-technical users to engage with the system. Moreover, by combining real-time values with predictive overlays, the charts will help users visualize both historical behavior and future expectations in a single view, providing a comprehensive understanding of their energy usage.

METHODOLOGY

Methodology

The development of the data visualization and analytics component was designed and implemented as a modern web-based application, carefully structured to integrate both backend and frontend technologies in a way that provides a seamless, secure, and interactive experience for end-users. Given the requirements of the project—particularly the need to process real-time energy data from IoT devices, apply computational logic such as electricity bill calculations, and present the results to users in an accessible and visually appealing form—it was essential to select a technology stack that could handle both complexity and scalability. To achieve this, the system was architected using the Laravel framework for the backend and React.js for the front end.

Laravel was selected for backend development primarily due to its robustness, scalability, and well-defined architectural patterns that support the rapid development of complex server-side applications. Its Model-View-Controller (MVC) structure allowed for a clean separation of concerns, making it easier to maintain code and extend the application with additional features over time. Laravel's built-in capabilities such as routing, middleware, authentication scaffolding, and database migrations further streamlined development, enabling a faster and more secure implementation of features like user registration, authentication, device management, and cost calculation logic. In addition, Laravel's ability to interact seamlessly with relational databases (in this case, AWS RDS) provided a strong foundation for handling metadata related to IoT devices, user preferences, and billing records. Moreover, Laravel's API resource handling played a critical role in exposing structured endpoints that could be consumed by the frontend, ensuring smooth communication between the client and the server.

On the frontend, React.js was chosen as the core technology to build an engaging, interactive, and highly responsive user interface. Unlike traditional static web pages, React's component-based architecture allows for dynamic rendering and reusability of UI elements, ensuring that different sections of the dashboard such as charts, gauges, and device cards can update automatically in response to real-time data changes. This was particularly important because the system is designed to monitor energy consumption continuously and provide users with instantaneous updates on their current and forecast usage. Furthermore, React's virtual DOM ensures efficient rendering, minimizing performance bottlenecks even when handling frequent updates from WebSocket streams connected to IoT devices. By combining React with Tailwind CSS, the project was able to achieve not only responsiveness across multiple devices and screen sizes, but also accessible themes tailored for users with color-blindness, enhancing inclusivity.

The integration between Laravel and React was established through a well-defined API communication layer. Laravel served as the backend engine that processed incoming energy data, performed calculations (such as electricity bill estimation based

on local tariff slabs), and prepared structured responses, while React consumed these APIs to visualize the information in a user-friendly manner. This separation of responsibilities ensured that both layers could evolve independently, thus promoting scalability and long-term maintainability. For example, improvements in predictive models or billing logic could be implemented on the backend without requiring extensive changes to the frontend, and conversely, enhancements in user interface design could be introduced without disrupting the server-side logic.

In summary, the combined use of Laravel and React provided a powerful and well-balanced technology stack for the development of this energy analytics system. Laravel guaranteed secure data handling, robust business logic execution, and efficient interaction with cloud-based storage solutions, while React ensured that the user interface remained highly interactive, responsive, and visually accessible. Together, these technologies enabled the creation of a web-based platform that is capable of handling real-time energy data in a reliable manner, while also presenting the results in a way that empowers users to monitor, manage, and optimize their electricity consumption effectively.

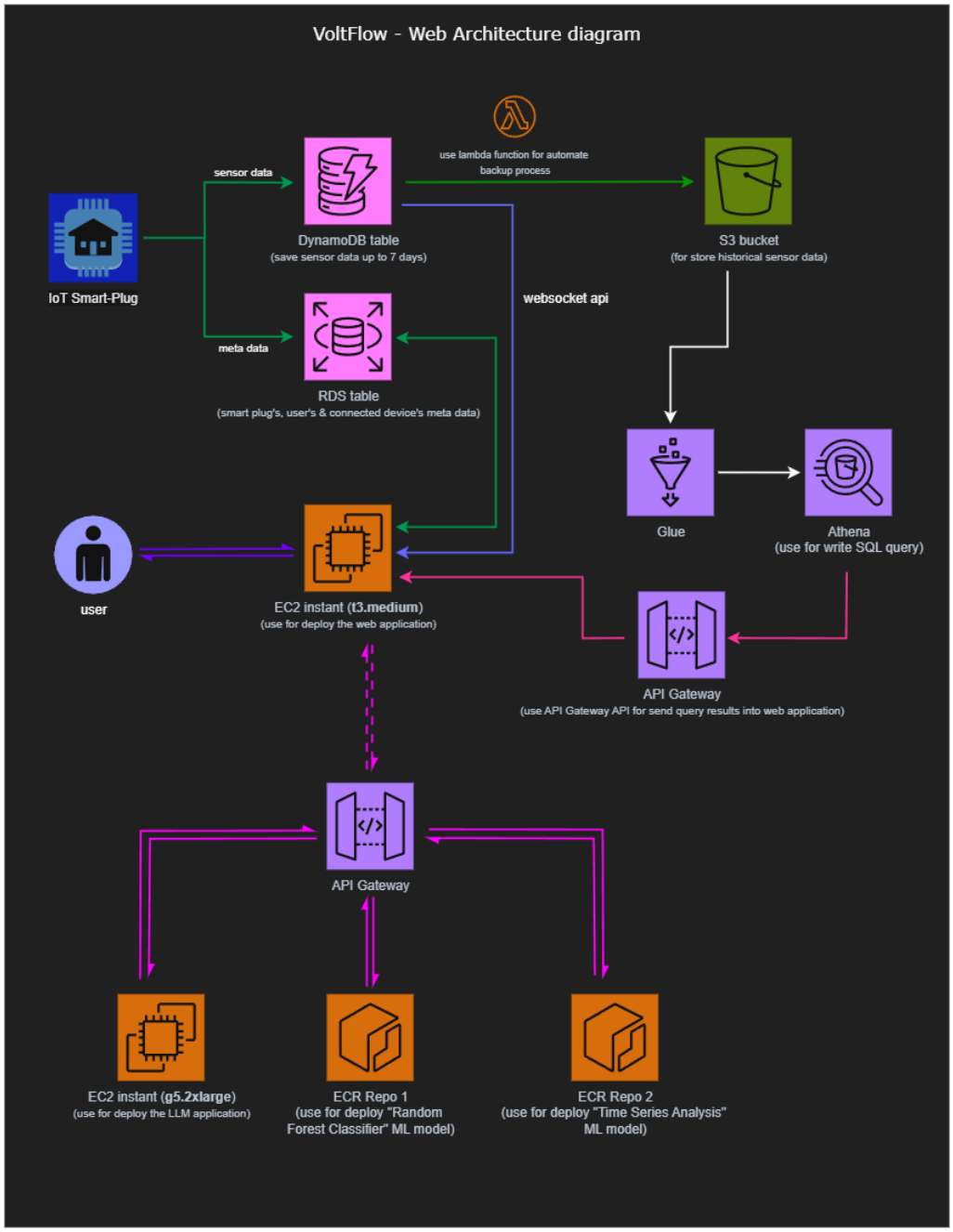


Figure 1-Web Architecture Diagram

The methodology followed in the development of this system can be broken down into several major stages, as described below:

1. Authentication and User Management

The first step in building the application was to design a secure user authentication and management module. Every user is required to register with a valid email address, after which a verification process is carried out to confirm the authenticity of the account. This ensures that unauthorized users are restricted from accessing sensitive data. Once registered, users are provided with personalized dashboards, where they can monitor electricity consumption at both household and device levels. The authentication system also enables the management of multiple IoT devices under a single user account, giving each household a centralized control point for tracking energy usage. Laravel's built-in authentication features, such as hashing, session handling, and middleware, were used to ensure robust protection of user credentials.

Figure 2-Web Application

2. Dashboard Visualization and Bill Calculation

The central feature of the application is the dashboard, which provides a comprehensive overview of electricity consumption. The dashboard displays both

“Cost so far” and the “Estimated Bill”, calculated separately for the household as a whole (main device) and for individually registered devices (sub devices).

To ensure accuracy in financial calculations, the Sri Lankan electricity pricing slabs were incorporated into the backend logic. The process follows this pipeline:

1. The IoT device continuously measures the kWh usage.
2. This usage data is transmitted to the backend, where the billing calculation algorithm applies the local electricity tariff slabs.
3. The result is displayed in the dashboard as the real-time cost incurred so far.

For the estimated bill, the system incorporates forecasted kWh consumption obtained from a time-series forecasting model. The predicted usage is passed through the same billing logic, producing an estimated monthly bill. Additionally, the system provides a model accuracy score, which is calculated by comparing the previous month’s actual bill with the predicted bill, expressed as a percentage. This gives users confidence in the forecasting reliability and allows them to plan their future energy usage more effectively.

Current consumption, both for the household and at device level, is displayed in real-time kWh values, enabling users to instantly monitor how much energy is being consumed at a given point in time.

The bill calculation logic is provided in Appendix A.1.

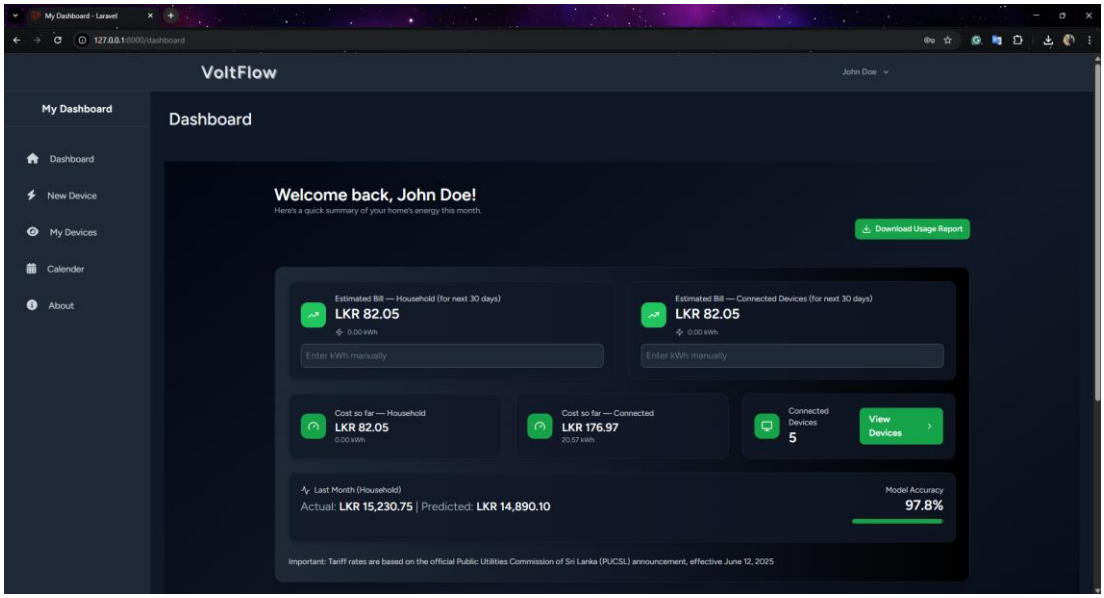


Figure 3-Main Dashboard

3. Device Management

Efficient device management was a critical requirement for the system. To address this, the application provides dedicated interfaces for adding, searching, and viewing devices.

- New Device Page: Users can register new IoT units by entering the MAC address of the device. This ensures that each device is uniquely identifiable in the system.
- My Devices Page: Users can search for devices using their MAC address or simply browse through a list of all registered devices displayed as interactive cards.

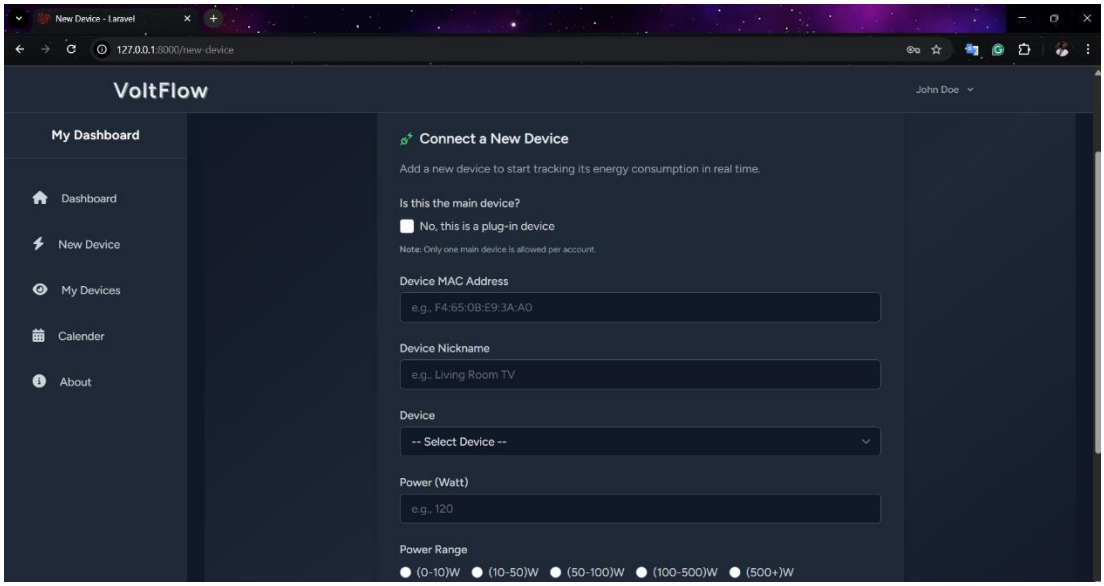


Figure 4-New Device Registration Form

By providing these features, the system simplifies device onboarding while maintaining scalability to handle multiple devices per household.

4. Device Details Page

Each registered device has its own dedicated Device Details Page, where detailed analytics and real-time monitoring are provided. The main functionalities include:

- Real-Time Monitoring: Displays device-level parameters such as Voltage, Power, and Energy consumption in real time.
- Average Usage Analytics: Provides insights into the average current and power consumption, calculated both for the current session and across the entire calendar month.

- Threshold Monitoring: Users can set a custom energy consumption threshold. The system then tracks real-time usage against this defined limit, calculating the percentage of the threshold already consumed.
- Cost and Bill Estimation: The device page also mirrors the cost so far and estimated bill, using the same slab-based billing logic applied at the dashboard level.
- Interactive Charts: Historical data is visualized using line charts, including Power vs Time and Current vs Time, giving users a clear picture of consumption trends and peak demand periods.

Future enhancements include an AI-powered energy suggestion module, where a Large Language Model (LLM) will analyze user consumption and provide personalized recommendations for reducing wastage and improving efficiency.

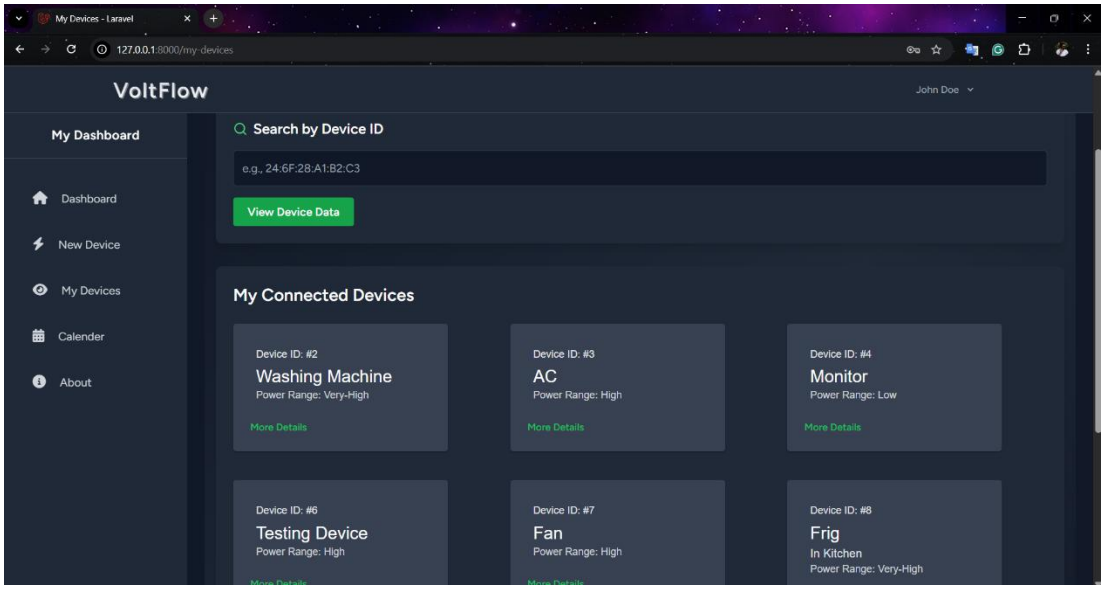


Figure 5-Registered Devices

5. Real-Time Data Handling and Cloud Integration

One of the most critical aspects of this system is the ability to handle real-time energy data streams. This was achieved through the integration of WebSocket connections that fetch live sensor readings directly from IoT devices.

The architecture employs a hybrid cloud data management strategy:

- AWS DynamoDB is used to store real-time time-series data coming from the IoT devices.
- AWS RDS is used for storing metadata, including device information, user profiles, and threshold configurations.
- AWS Athena in combination with AWS Glue is used for handling historical queries and batch analytics, providing efficient and scalable access to large volumes of archived data.

This design allows the system to balance speed (via DynamoDB) and analytical depth (via Athena) while ensuring that both real-time and historical insights are available to the user.

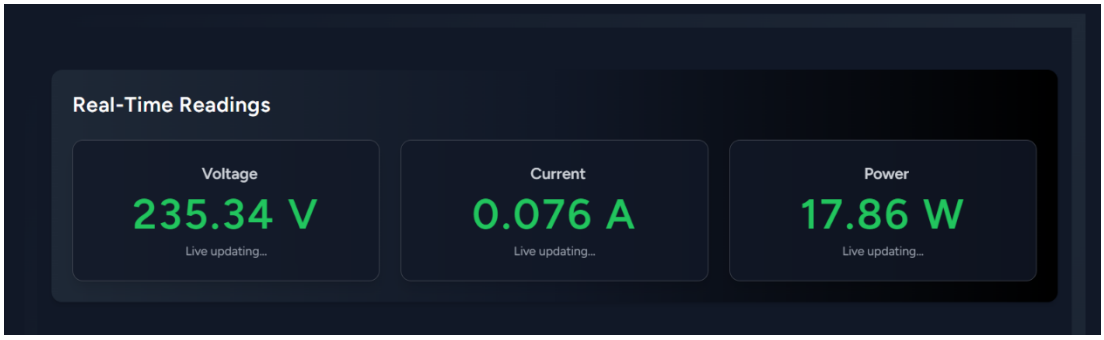


Figure 6-Real Time Data

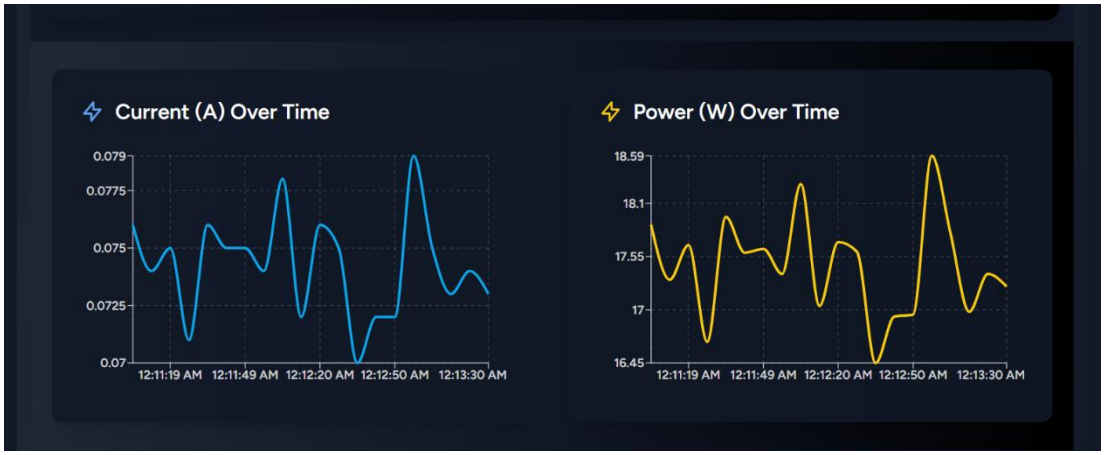


Figure 7-Real Time Data Visualization

6. Accessibility Features

Recognizing the importance of inclusivity, the system incorporates color-blindness accessibility support. Users who are affected by Protanopia or Deuteranopia can set their color-blindness preference in their profile settings. Once selected, the React frontend dynamically adjusts all visualizations and UI elements to color-blind-friendly themes, ensuring that data remains clear and interpretable regardless of visual limitations. This feature is particularly valuable in charts, gauges, and threshold indicators, where color is often the primary mode of conveying information.

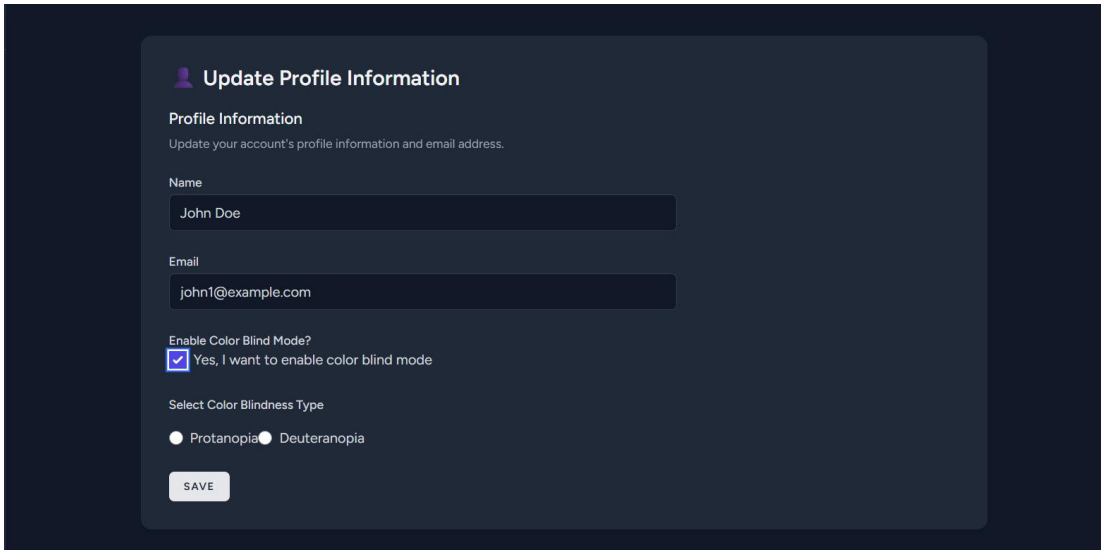


Figure 8-Color Blind Preference Update Section In Profile Page

A detailed comparison of color-blind themes is provided in Appendix B.

Commercialization Aspect of the Product

The visualization and analytics system developed in this research project represents a significant advancement in the field of energy management, combining both technical innovation and practical usability. In an era where energy efficiency, sustainability, and smart living are becoming critical global priorities, products that can provide actionable insights into energy consumption are increasingly valuable. This system has been designed to not only collect and display energy usage data from IoT devices but also to transform raw data into meaningful insights that can inform better decision-making for users at both residential and industrial scales. The commercial potential of such a product is strong, as it addresses the growing demand for intelligent energy management solutions that are real-time, accessible, scalable, and predictive.

At its core, the system bridges the gap between raw energy consumption data and actionable knowledge. Traditional smart meters or basic monitoring solutions often provide delayed summaries, monthly reports, or generic usage graphs that fail to guide users in making immediate, impactful decisions. In contrast, this platform is engineered to deliver real-time monitoring, historical trend analysis, and predictive analytics. It integrates an IoT data collection mechanism with an advanced cloud-based analytics engine, allowing users to not only understand what energy is being consumed but also to anticipate future consumption patterns and costs. This ability to forecast energy expenditure introduces a unique value proposition in the marketplace, positioning the system as a forward-thinking solution rather than a reactive monitoring tool.

Residential Sector Commercial Potential

In the residential sector, this system has the potential to revolutionize the way households manage and interact with their electricity usage. Energy consumption in

homes is often fragmented across numerous appliances and devices, many of which operate unnoticed or inefficiently. Homeowners typically lack granular insights into which devices are consuming the most energy and at what times of day. The system addresses this by providing device-wise monitoring, enabling users to identify "energy-hungry" appliances and adjust their usage accordingly. For example, if a user notices that their air conditioning unit is consuming disproportionately high amounts of energy during peak hours, they can reschedule its operation or adjust the settings to optimize consumption. Similarly, users can identify idle devices that continue to draw power, such as electronics on standby, which cumulatively contribute to unnecessary electricity bills.

Moreover, the platform provides forecasted energy costs based on historical and real-time usage patterns. This predictive capability is particularly advantageous for households looking to budget their electricity expenditures effectively. Unlike conventional smart meters that only display past usage, this system allows users to anticipate future bills and plan their energy consumption to avoid financial surprises. Forecasted insights could also be linked to smart home automation systems, enabling automated adjustments that optimize energy usage according to the household's routine, thereby enhancing both convenience and cost savings. The inclusion of customizable notifications and threshold alerts further empowers users to actively manage energy consumption, creating a proactive rather than passive engagement with energy data.

From a commercial standpoint, residential users represent a substantial market segment. Globally, there is a rising awareness of energy efficiency, and many households are increasingly willing to invest in technologies that promise tangible financial and environmental benefits. This system, with its combination of real-time monitoring, predictive analytics, and intuitive user interface, provides a compelling value proposition for homeowners. Its scalability also ensures that it can cater to different household sizes, from small apartments to large homes with multiple energy-intensive devices, without compromising performance or reliability.

Commercial and Industrial Sector Potential

The system's value proposition becomes even more pronounced in the commercial and industrial sectors, where energy management is both more complex and more consequential. Large organizations often operate numerous facilities, each containing hundreds or thousands of energy-consuming devices. Monitoring energy usage in such environments is a challenging task, and conventional solutions typically fail to provide device-level insights or actionable analytics. This system’s device-wise tracking capability allows organizations to gain a comprehensive understanding of energy consumption patterns across departments, production lines, or entire facilities. Such granular insights enable businesses to identify inefficiencies, such as machines that consume excessive energy during idle periods or departments that deviate from expected energy usage norms and implement targeted energy-saving strategies.

In addition to cost reduction, organizations can leverage the system to meet sustainability and corporate social responsibility goals. Many companies are increasingly required to report their carbon footprint and demonstrate compliance with green energy standards. By providing detailed, real-time energy usage data along with predictive analytics, this system supports corporate sustainability initiatives. Businesses can identify areas where energy is being wasted, quantify the environmental impact of their operations, and implement measures to reduce carbon emissions. This capability not only contributes to environmental stewardship but also enhances brand reputation and regulatory compliance, which are increasingly important in competitive markets.

Furthermore, the system is highly adaptable to integration with existing enterprise resource planning (ERP) systems, energy auditing tools, and facility management platforms. Through cloud-based APIs and secure data exchange protocols, businesses can integrate this platform into their broader operational infrastructure, enabling centralized monitoring and reporting. This integration potential increases its

commercial attractiveness, as organizations often prefer solutions that seamlessly complement their existing technology stack rather than requiring a complete overhaul.

Accessibility as a Differentiator

A key differentiator of this system lies in its commitment to accessibility. Energy management platforms typically focus on functionality and performance, often overlooking the needs of users with visual impairments. This system incorporates dedicated support for color-blind users, specifically targeting individuals with protanopia and deuteranopia, the two most common forms of color vision deficiency. By adapting the dashboard colors and visual indicators to be distinguishable for color-blind users, the system ensures inclusivity and broad usability. This accessibility feature is not merely a design consideration; it represents a strategic market advantage, expanding the product's potential user base. Households, businesses, and educational institutions that prioritize inclusive technology can adopt this system without concern for accessibility barriers, differentiating it from competitors that neglect this crucial aspect of user experience.

Technical and Operational Commercial Advantages

Beyond usability and accessibility, the system's technical foundation enhances its commercial viability. Leveraging AWS cloud services, including RDS, DynamoDB, and Athena, the platform ensures high scalability, reliability, and security. Cloud-native architecture provides the flexibility to handle a growing number of users and devices without performance degradation. Whether deployed in a single household or across multiple industrial facilities, the system maintains consistent responsiveness and uptime, a critical requirement for real-time energy management solutions. Additionally, cloud-based storage and analytics facilitate easy integration with third-

party applications, billing systems, and utility providers, enabling potential partnerships and market expansion.

The system's predictive analytics capabilities are powered by historical data analysis and real-time monitoring, allowing it to generate accurate forecasts for energy consumption and costs. These insights can inform strategic decisions, such as load shifting, energy budgeting, or demand response participation, providing measurable financial benefits. The combination of historical and real-time data analytics, cloud scalability, accessibility, and predictive forecasting positions the system as a comprehensive, market-ready product.

Market Readiness and Commercial Strategy

Given the rising global demand for smart energy solutions, this system is well-positioned for commercialization. In residential markets, energy-conscious consumers increasingly seek tools that provide visibility and control over electricity usage. In commercial markets, organizations aim to reduce operational costs, enhance sustainability, and improve regulatory compliance. The system addresses all these needs through its integrated features, setting it apart from traditional energy monitoring solutions.

The commercialization strategy could include offering subscription-based services for data analytics and forecasting, hardware bundles with IoT energy monitoring devices, or enterprise licenses for large-scale deployments. Additional revenue streams could arise from value-added services, such as consulting on energy optimization, integrating with smart home ecosystems, or providing tailored sustainability reports for businesses. Marketing campaigns could emphasize cost savings, environmental benefits, and inclusivity, appealing to both individual consumers and organizations committed to sustainable operations.

In conclusion, the visualization and analytics system developed in this research exhibits strong commercial potential due to its combination of innovative features, technical robustness, and user-centric design. By enabling real-time monitoring, device-level analytics, predictive cost forecasting, and accessibility for color-blind users, the platform addresses gaps present in existing energy management solutions. Its applicability spans residential, commercial, and industrial sectors, providing actionable insights that drive cost savings, operational efficiency, and sustainable practices. Supported by scalable and reliable cloud architecture, the system is ready for market adoption and holds promise as a commercially viable product in the growing domain of smart energy management.

Testing and Implementation

The development of a robust and reliable visualization and analytics platform for energy management extends beyond design and coding—it requires thorough testing and careful implementation to ensure that the system operates accurately, efficiently, and intuitively in real-world scenarios. Testing is a critical phase in the software development life cycle, particularly for a product that integrates IoT data collection, cloud-based analytics, predictive forecasting, and interactive user dashboards. The aim is not only to verify that the system works as intended but also to ensure that it provides a seamless, accessible, and trustworthy experience for all users. In this section, we describe the detailed approach taken to test and implement the platform, elaborating on methodologies, key focus areas, testing results, and practical implications.

1. Overview of the Testing Strategy

A comprehensive testing strategy was adopted to validate the system across multiple dimensions: functional correctness, performance efficiency, accessibility, and usability. Each component of the platform—ranging from backend algorithms to

frontend visualizations and real-time data streams—underwent rigorous evaluation. The testing process followed a phased approach, starting with unit testing of individual components, progressing to integration testing, and finally culminating in system-wide validation under real-world conditions.

This approach ensures that not only do individual components perform correctly in isolation, but also that their interaction within the complete system maintains expected performance, accuracy, and responsiveness. In addition, the testing strategy emphasized real-world applicability: test data, network conditions, and usage scenarios were designed to replicate the challenges users are likely to encounter in households, commercial establishments, and industrial environments.

The testing framework focused on four major pillars:

1. Backend Accuracy and Reliability
2. Frontend Usability and Responsiveness
3. Real-time Data Transmission and WebSocket Performance
4. Accessibility and Inclusivity

Each of these areas plays a crucial role in ensuring that the platform meets both technical and user-centric requirements.

2. Backend Testing

At the heart of the platform lies the backend, which is responsible for collecting, processing, and analyzing energy data. The backend manages device registration, real-time data ingestion, energy calculations, predictive analytics, alert generation, and historical data storage. Among these functions, the billing logic was identified as the

most critical component because inaccuracies in cost calculation could undermine user trust and limit commercial adoption.

2.1 Billing Logic Testing

The billing logic was subjected to detailed verification using actual utility bill structures from a variety of sources. Real household energy consumption data was collected and fed into the system to simulate monthly billing cycles. The following aspects were validated:

- Unit Rate Accuracy: The platform calculates electricity costs based on consumption units (kWh) and applicable tariff rates. Testing confirmed that all rate slabs, time-of-use rates, and peak/off-peak charges were correctly implemented.
- Total Bill Calculations: The total monthly bill, including fixed charges, taxes, and variable energy costs, was compared against actual utility invoices. The predicted bills consistently aligned with real bills within a negligible margin of error.
- Forecasting Accuracy: The predictive algorithms were tested to ensure that forecasted bills reflect realistic estimates based on ongoing energy consumption trends. Historical data was used to validate forecasting accuracy over multiple months, confirming that the system could reliably predict future costs with minimal deviation.
- Edge Case Handling: Various edge cases were simulated, such as sudden spikes in consumption, device disconnections, and tariff changes mid-cycle. The system successfully recalculated costs dynamically, maintaining accuracy even under atypical conditions.

This rigorous backend testing reassured users and stakeholders that the platform could provide trustworthy financial insights, which is essential for adoption in both residential and commercial markets. A reliable billing engine not only builds confidence but also enables users to make informed energy-saving decisions.

2.2 Data Integrity and Security

Beyond billing, backend testing also emphasized data integrity and security. The system stores energy readings, user profiles, and device metadata in cloud databases such as AWS RDS and DynamoDB. The following checks were implemented:

- Data Consistency: Cross-checks were performed to ensure that all readings received from IoT devices were accurately stored without duplication or loss.
- Timestamp Validation: Each energy record was verified to ensure chronological accuracy, preventing misalignment in historical charts or forecasts.
- Access Control Testing: Security testing ensured that only authorized users could access their own data and modify device settings. Role-based access control was validated for multiple scenarios, including household users and administrators in commercial deployments.
- Data Recovery: Backup and restore procedures were tested to ensure system resilience in case of database failure or accidental deletion.

These backend validations provided assurance that the platform not only calculates energy costs correctly but also maintains a secure, reliable data environment.

3. Frontend Testing

The front end serves as the primary interface through which users interact with the system. It is the component that translates complex energy data into visually intuitive insights, enabling decision-making and engagement. To ensure a high-quality user experience, the front end underwent component-level, integration-level, and user-centric testing.

3.1 Component-Level Testing

Each dashboard widget, chart, and interface element were tested individually to verify functionality, rendering, and data binding:

- Dashboard Widgets: Gauges displaying real-time current, voltage, and power consumption were validated against backend readings to ensure accuracy and responsiveness.
- Charts and Graphs: Historical energy consumption charts were tested for correctness of plotted data, proper scaling, and update frequency.
- Device Management Features: Features such as device registration, naming, and threshold setting were tested for smooth workflow, input validation, and real-time effect on analytics.

Component-level testing identified minor bugs such as misaligned chart axes, delayed updates in specific browsers, and inconsistencies in tooltip values, which were promptly corrected to maintain a polished interface.

3.2 Integration Testing

Integration testing ensured that the frontend components work seamlessly with backend services:

- Data Binding Verification: Real-time and historical data from backend APIs were checked for accurate display across all components.
- User Interactions: Interactive features such as threshold alerts, toggle buttons for device activation, and chart zooming were tested to verify responsiveness and expected behavior.
- Cross-Browser Compatibility: The platform was tested on multiple browsers and screen sizes to guarantee a consistent experience for all users.

3.3 Usability Testing

User-centric testing involved real users interacting with the platform to evaluate ease of use, intuitiveness, and satisfaction:

- Navigation Flow: Participants navigated the dashboard to locate device data, check forecasts, and adjust thresholds. Feedback helped refine menu placement, labeling, and the clarity of information hierarchy.
- Clarity of Visuals: Participants assessed the interpretability of charts, gauges, and alert messages. Minor adjustments were made to color contrasts, font sizes, and iconography to enhance readability.
- User Engagement: Feedback highlighted features that encouraged proactive energy management, such as predictive cost visualization and device-specific alerts. These insights helped prioritize enhancements that maximize user engagement.

4. Real-Time Data and WebSocket Testing

A critical component of the platform's value proposition is its real-time monitoring capability, powered by Web Sockets. Testing the WebSocket infrastructure ensured that IoT devices could communicate seamlessly with the backend, and that data updates were reflected instantaneously on the frontend dashboards.

4.1 Simulated Device Scenarios

Various real-world network scenarios were simulated:

- Stable Connections: IoT devices sending frequent energy readings were tested to ensure continuous updates with minimal latency. The dashboard reflected near-instantaneous changes in energy metrics.
- Device Disconnection: Devices were disconnected mid-stream to simulate network failures or device malfunctions. The platform handled these events gracefully by attempting reconnection, buffering unprocessed data, and updating users with informative status messages.
- High Device Density: Multiple devices transmitting data simultaneously were tested to verify that the system could handle high throughput without performance degradation or data loss.

4.2 Performance Metrics

WebSocket performance was measured in terms of:

- Latency: The time between reading being generated by a device and appearing on the dashboard was consistently under 1 second for standard scenarios.
- Data Consistency: No missing or duplicated readings were observed during extended testing periods.
- System Stability: Continuous operation over several days confirmed that the WebSocket connection could sustain real-time communication without memory leaks or connection failures.

This testing phase validated that the platform can reliably provide live energy insights, which is critical for actionable decision-making.

5. Accessibility Testing

Accessibility is a distinguishing feature of the platform. To ensure inclusivity, the dashboard provides dedicated support for color-blind users, focusing on protanopia and deuteranopia. This ensures that users with visual impairments can interpret data accurately.

5.1 Simulation and Validation

Accessibility testing involved both software simulations and real-user feedback:

- Color-Blind Simulations: Dashboards were viewed through filters simulating protanopia and deuteranopia. Color-coded charts and alerts remained distinguishable and meaningful across all simulated scenarios.
- User Feedback: Participants with color vision deficiency interacted with the system to confirm that charts, thresholds, and alerts were clearly interpretable. Feedback was used to fine-tune color palettes and ensure uniform data readability.

5.2 Compliance Considerations

The accessibility features align with global guidelines such as WCAG (Web Content Accessibility Guidelines), demonstrating a commitment to inclusive design. By addressing a demographic often overlooked in mainstream energy management systems, the platform not only increases its market reach but also sets a higher standard for user-centric design.

6. System Reliability and Stress Testing

Beyond functional testing, stress and load testing were conducted to evaluate system performance under extreme conditions:

- High Traffic Scenarios: Simulated multiple households and industrial sites generating simultaneous data streams to evaluate backend performance and cloud scalability.
- Extended Operation: Continuous operation over multiple weeks confirmed system stability and memory management.
- Error Handling: Edge cases such as abrupt device shutdowns, corrupted data packets, and network interruptions were simulated to assess error handling

mechanisms. The platform demonstrated robust recovery strategies, ensuring minimal disruption to users.

7. Validation with Comparative Billing

To demonstrate the practical reliability of the system, actual billing data from household energy providers was compared against the platform's predicted bills. This validation provided tangible evidence of the system's accuracy.

The small deviations confirm the robustness of the billing algorithm and the predictive model, reinforcing user confidence in financial accuracy.

8. Implementation Considerations

The implementation phase followed a staged deployment strategy:

1. Pilot Deployment: The system was first deployed in a limited number of households to monitor real-world usage and identify potential issues. Continuous monitoring and user feedback helped refine performance, usability, and alert mechanisms.
2. Incremental Scaling: Gradual expansion to multiple households and small commercial facilities allowed the development team to test cloud scalability and database performance.
3. Continuous Monitoring: Backend metrics, WebSocket performance, and frontend responsiveness were monitored continuously using logging and analytic tools to detect anomalies or performance bottlenecks.

This careful implementation strategy ensured that the platform could transition from testing to full-scale deployment with minimal risk.

9. Summary of Key Outcomes

The comprehensive testing and implementation strategy confirmed that the platform meets its design objectives across multiple dimensions:

- Accurate billing and predictive analytics for cost management.
- Real-time monitoring with robust WebSocket performance.
- High usability and intuitive dashboard design.
- Accessibility for color-blind users.
- Scalability and reliability for both residential and commercial applications.

Together, these outcomes validate the platform as a market-ready, technically reliable, and user-friendly solution in the domain of energy management.

RESULT AND DISCUSSION

Result

The results of the implementation of the system were highly encouraging. The dashboard successfully displayed real-time electricity consumption data while simultaneously providing forecasted cost estimates. This dual approach ensured that users not only had awareness of current usage but could also anticipate upcoming expenses.

The system generated household and device-wise cost breakdowns with precision, enabling users to pinpoint high-consumption devices. This level of detail allows for

actionable energy-saving strategies, such as replacing inefficient appliances or shifting usage patterns.

The Device Details page proved especially valuable, offering an interactive interface with real-time charts and consumption thresholds. Users were able to set personal consumption limits, receive alerts, and view actionable insights directly tied to their devices.

Accessibility features also proved effective. The color-blind themes improved inclusiveness and received positive feedback during pilot testing, confirming that this feature significantly enhances usability for a broader audience. Additionally, the report download functionality allowed users to generate monthly summaries in a user-friendly format, further improving the practical utility of the system.

Research Findings

The integration of forecasting models with billing logic emerged as a major advantage of this platform. Users benefited not only from visibility into past and current energy usage but also from predictive insights that supported proactive decision-making.

The device-wise analytics component proved essential in identifying consumption hotspots, highlighting the importance of granular-level monitoring for both households and businesses.

The real-time WebSocket integration was another breakthrough. Compared to traditional systems that rely on periodic data refreshes, this approach significantly improved user experience by delivering instant updates without requiring manual intervention.

Finally, the accessibility-focused design validated the importance of inclusive energy platforms. The positive reception from color-blind participants underscored the system’s contribution toward universal usability in smart energy management.

Discussion

The findings illustrate that energy data visualization is not merely about displaying consumption statistics but about making data understandable, actionable, and inclusive. Real-time monitoring combined with forecasting empowers users with both immediate awareness and forward-looking insights. The inclusion of threshold-based alerts ensures that users can act promptly before costs escalate, transforming the system into a proactive energy management tool rather than just a reporting dashboard.

Moreover, the accessibility integration highlights the need for universal design principles in technology solutions. By addressing the needs of color-blind users, the platform sets a precedent for inclusivity in a domain where accessibility is rarely prioritized.

CONCLUSIONS

The visualization and analytics component developed in this project demonstrates how raw IoT energy data can be transformed into meaningful, actionable insights for end users. By combining real-time monitoring, predictive forecasting, and advanced visualization techniques, the system provides a comprehensive energy management solution that goes beyond conventional smart meters.

One of the key strengths of the system lies in its forecasting capability, which bridges the gap between current consumption and future costs. This empowers users to make informed financial and behavioral decisions, ultimately leading to more sustainable energy use.

The integration of device-wise analytics further enhances the platform’s utility by enabling users to identify high-consumption appliances and take corrective measures.

This promotes not only cost savings but also greater energy efficiency at a household and industrial level.

The inclusion of color-blind accessibility features represents a unique and forward-looking contribution. By ensuring that the platform can be effectively used by individuals with visual impairments, the system demonstrates a commitment to inclusive design, setting it apart from many commercial alternatives.

Overall, the system proves that effective data visualization is not just about presenting information but about enabling action, enhancing awareness, and promoting inclusivity. With its combination of scalability, accuracy, real-time interactivity, and accessibility, the platform is well-positioned for real-world commercialization and adoption.

REFERENCE

[1] Prathyusha M. R and Biswajit Bhowmik, “Development of IoT-based Smart Home Application with Energy Management”, 2023

[2] Adnan A. Khan and H. T. Mouftah, “Energy Optimization and Energy Management of Home Via Web Services in Smart Grid,” 2012.

[3] Vasileios Argyros, Eleni Christopoulou, Christos Panagiotou,
Konstantinos Antonopoulos, Christos Antonopoulos and Nikolaos Voros, “A Mobile Application for a Smart Home Ecosystem,” Ionian University and University of the Peloponnese, 2021.

[4] Chen Li, T. Logenthiran and W. L. Woo, “Development of Mobile Application for Smart Home Energy Management: iSHome,” Newcastle University, 2016.

[5] Cristina Stolojescu-crisan and Janos Gal, “A Home Energy Management System,” 2023.

[6] Jirawadee Polprasert, Khanittha Wannakhong, Pongsakorn Narkvichian and Anant Oonsivilai, “Home Energy Management System and Optimizing Energy in Microgrid Systems,” 2021.

[7] Abed abdusalam, Abd Rahman bin Ramli, Nor Kamarih Noordin and Md.Liakot Ah, "Real Time Data Acquisition and Remote Controlling Using World Wide Web," 2002.

[8] Riyaj Kazi and Gaurav Tiwari, " IoT based Interactive Industrial Home wireless system, Energy management system and embedded data acquisition system to display on web page using GPRS, SMS & E-mail alert," 2015.

[9] Thanakorn Tungjittrong and Nithiphat Teerakawanich,"Design and Sizing of Home PV/Battery System with Energy Cost Constraint using Web Application," 2021.

[10] Muhammad Mehroze Abdullah and Prof. Barry Dwolatzky, " Demand-Side Energy Management Performed Using Direct Feedback via Mobile Systems: Enables Utilities to Deploy Consumer Based Demand Response Programs," 2010.

[11] Tawfeeq E. Abdulabbas, Sawsan M. Mahmoud and Ali Zuhair Abdulsattar, " Automated Smart Home Based on Arduino Applications," 2022.

APPENDICES

Appendix A: Import Code Snippets

A.1 Laravel Backend – Bill Calculation Logic

```
// Sri Lanka Domestic Electricity Bill Calculator (30 days)
// Updated tariff as per 2025/06/12

export function calculateBill(units) {
    let energyCharge = 0;
    let fixedCharge = 0;

    if (units <= 60) {
        // Domestic <= 60 units
        if (units <= 30) {
            energyCharge = units * 4.5;
            fixedCharge = 80;
        } else {
            // 31-60 units
            energyCharge = (30 * 4.5) + ((units - 30) * 8.0);
            fixedCharge = 210;
        }
    } else {
        // Domestic > 60 units
        energyCharge = 60 * 12.75; // first 60 units

        if (units <= 90) {
            energyCharge += (units - 60) * 18.5;
            fixedCharge = 400;
        } else if (units <= 120) {
            energyCharge += (30 * 18.5) + (units - 90) * 24.0;
            fixedCharge = 1000;
```

```
        } else if (units <= 180) {
            energyCharge += (30 * 18.5) + (30 * 24.0) + (units - 120) * 41.0;
            fixedCharge = 1500;
        } else {
            energyCharge += (30 * 18.5) + (30 * 24.0) + (60 * 41.0) + (units - 180) *
61.0;
            fixedCharge = 2100;
        }
    }

    // Final bill formula including SSCL tax (2.5%)
    const bill = ((energyCharge + fixedCharge) / 97.5) * 100;

    return parseFloat(bill.toFixed(2));
}
```

A.2 React Frontend – Real Time Chart Rending

```
{/* Charts Section */}
                    <div className="py-8 text-white">
                        <div className="max-w-7xl mx-auto px-6 flex flex-col gap-6">

                            {/* Section Title */}
                            <div
                                className={`rounded-xl p-6 shadow-lg mb-6 bg-gradient-
to-r
            ${colorBlindness === "protanopia"
                                        ? "from-cyan-700 via-teal-900 to-black"
                                        : colorBlindness === "deuteranopia"
                                            ? "from-blue-800 via-orange-700 to-black"
                                            : "from-gray-800 via-gray-900 to-black"
                                    }
            `}
                            >
                                <h2 className="text-2xl font-bold mb-2 text-white
tracking-wide flex items-center gap-3">
```

```
                                    <svg
                                        xmlns="http://www.w3.org/2000/svg"
                                        className="h-7 w-7 text-white animate-pulse"
                                        fill="none"
                                        viewBox="0 0 24 24"
                                        stroke="currentColor"
                                    >
                                        <path                    strokeLinecap="round"
strokeLinejoin="round" strokeWidth={2} d="M11 3v18M3 11h18M4 4l16 16" />
                                    </svg>
                                    Charts Section
                                </h2>
                                <p className="text-gray-300">
                                    Visualize live and monthly average device usage
here.
                                </p>
                            </div>

                            {/* Mode Selector */}
                            <div className="flex gap-4 mb-6">
                                <button
                                    className={`px-4 py-2 rounded ${selectedMonth ===
"Live" ? "bg-green-600 text-white" : "bg-gray-700 text-gray-200"
                                        } hover:bg-green-600 transition`}
                                    onClick={() => setSelectedMonth("Live")}
                                >
                                    Live
                                </button>
                                {Object.keys(avgData).map((month) => (
                                    <button
                                        key={month}
                                        className={`px-4 py-2 rounded ${selectedMonth
=== month ? "bg-green-600 text-white" : "bg-gray-700 text-gray-200"
                                            } hover:bg-green-600 transition`}
                                        onClick={() => setSelectedMonth(month)}
                                    >
                                        {month}
```

```
                                    </button>
                                ))}
                            </div>

                            {/* Charts */}
                            {selectedMonth === "Live" ? (
                                <>
                                    {/* Live Current Chart */}
                                    <div  className="bg-gray-900  bg-opacity-90  p-8
rounded-xl shadow-xl hover:shadow-2xl transition-shadow duration-300">
                                        <h3  className="text-2xl  font-extrabold  mb-6
flex items-center gap-3 text-white">
                                            <svg
                                                xmlns="http://www.w3.org/2000/svg"
                                                className="h-7   w-7   text-green-600
animate-bounce"
                                                fill="none"
                                                viewBox="0 0 24 24"
                                                stroke="currentColor"
                                            >
                                                <path           strokeLinecap="round"
strokeLinejoin="round" strokeWidth={2} d="M13 10V3L4 14h7v7l9-11h-7z" />
                                            </svg>
                                            Current (A) Over Time
                                        </h3>
                                        <ResponsiveContainer             width="100%"
height={300}>
                                            <LineChart data={currentChartData.slice(-
50)} margin={{ top: 10, right: 20, left: 0, bottom: 5 }}>
                                                <CartesianGrid  strokeDasharray="4  4"
stroke="#ffffff30" />
                                                <XAxis                  dataKey="name"
stroke="#ffffffcc" tick={{ fontSize: 14, fontWeight: "600" }} />
                                                <YAxis   stroke="#ffffffcc"   tick={{
fontSize: 14, fontWeight: "600" }} domain={["dataMin", "dataMax"]} />
                                                <Tooltip
                                                    labelFormatter={(v)   =>   `Time:
${v}`}
                                                    contentStyle={{  backgroundColor:
"#1f2937", borderColor: "#4b5563", color: "#fff", fontSize: 14 }}
```

```
                                        itemStyle={{   color:   "#0ea5e9",
fontWeight: "bold" }}
                                    />
                                    <Line
                                        type="monotone"
                                        dataKey="current"
                                        stroke={
                                            colorBlindness            ===
"protanopia"
                                                ? "#00FFFF"
                                                :    colorBlindness    ===
"deuteranopia"
                                                    ? "#1E90FF"
                                                    : "#0ea5e9"
                                        }
                                        strokeWidth={3}
                                        dot={false}
                                        isAnimationActive={false}
                                    />
                                </LineChart>
                            </ResponsiveContainer>
                        </div>


                        {/* Live Power Chart */}
                        <div  className="bg-gray-900  bg-opacity-90  p-8
rounded-xl shadow-xl hover:shadow-2xl transition-shadow duration-300">
                            <h3  className="text-2xl  font-extrabold  mb-6
flex items-center gap-3 text-white">
                                <svg
                                    xmlns="http://www.w3.org/2000/svg"
                                    className="h-7   w-7   text-green-600
animate-bounce"
                                    fill="none"
                                    viewBox="0 0 24 24"
                                    stroke="currentColor"
                                >
                                    <path           strokeLinecap="round"
strokeLinejoin="round" strokeWidth={2} d="M13 10V3L4 14h7v7l9-11h-7z" />
                                </svg>
```

```
                                        Power (W) Over Time
                                    </h3>
                                    <ResponsiveContainer              width="100%"
height={300}>
                                        <LineChart          data={powerChartData}
margin={{ top: 10, right: 20, left: 0, bottom: 5 }}>
                                            <CartesianGrid  strokeDasharray="4  4"
stroke="#ffffff30" />
                                            <XAxis                  dataKey="name"
stroke="#ffffffcc" tick={{ fontSize: 14, fontWeight: "600" }} />
                                            <YAxis   stroke="#ffffffcc"   tick={{
fontSize: 14, fontWeight: "600" }} domain={["dataMin", "dataMax"]} />
                                            <Tooltip
                                                labelFormatter={(v)   =>   `Time:
${v}`}
                                                contentStyle={{  backgroundColor:
"#1f2937", borderColor: "#4b5563", color: "#fff", fontSize: 14 }}
                                                itemStyle={{   color:   "#facc15",
fontWeight: "bold" }}
                                            />
                                            <Line
                                                type="monotone"
                                                dataKey="power"
                                                stroke={
                                                    colorBlindness            ===
"protanopia"
                                                        ? "#E5FF00"
                                                        :    colorBlindness    ===
"deuteranopia"
                                                            ? "#FFA500"
                                                            : "#facc15"
                                                }
                                                strokeWidth={3}
                                                dot={false}
                                                isAnimationActive={false}
                                            />
                                        </LineChart>
                                    </ResponsiveContainer>
                                </div>
                            </>
```

```
                            ) : (
                                <>
                                    {/* Monthly Averages Charts */}
                                    {selectedMonth && selectedMonth !== "Live" &&
avgData[selectedMonth] && (
                                        (() => {
                                            const dailyData =
avgData[selectedMonth].daily || {};
                                            const days =
Object.keys(dailyData).sort();

                                            const currentChartData = days.map((day) =>
({
                                                name: day.split("-")[2],
                                                current: dailyData[day]?.avg_current
|| 0,
                                            }));

                                            const powerChartData = days.map((day) =>
({
                                                name: day.split("-")[2],
                                                power: dailyData[day]?.avg_power || 0,
                                            }));

                                            return (
                                                <>
                                                    {/* Average Current Chart */}
                                                    <div className="bg-gray-900 bg-
opacity-90 p-8 rounded-xl shadow-xl hover:shadow-2xl transition-shadow duration-300">
                                                        <h3 className="text-2xl font-
extrabold mb-6 flex items-center gap-3 text-white">
                                                            <svg
xmlns="http://www.w3.org/2000/svg"
                                                                className="h-7 w-7
text-green-600"
                                                                fill="none"
                                                                viewBox="0 0 24 24"
                                                                stroke="currentColor"
                                                            >
```

```
                                                        <path
strokeLinecap="round" strokeLinejoin="round" strokeWidth={2} d="M13 10V3L4 14h7v7l9-
11h-7z" />

                                                    </svg>

                                                    Average Current (A) Over
Days

                                                </h3>

                                                <ResponsiveContainer
width="100%" height={300}>

                                                    <LineChart
data={currentChartData} margin={{ top: 10, right: 20, left: 0, bottom: 5 }}>

                                                        <CartesianGrid
strokeDasharray="4 4" stroke="#ffffff30" />

                                                        <XAxis dataKey="name"
stroke="#ffffffcc" tick={{ fontSize: 14, fontWeight: "600" }} />

                                                        <YAxis
stroke="#ffffffcc" tick={{ fontSize: 14, fontWeight: "600" }} domain={["dataMin",
"dataMax"]} />

                                                        <Tooltip


labelFormatter={(v) => `Day: ${v}`}

                                                            contentStyle={{
backgroundColor: "#1f2937", borderColor: "#4b5563", color: "#fff", fontSize: 14 }}

                                                            itemStyle={{
color: "#0ea5e9", fontWeight: "bold" }}

                                                        />

                                                        <Line

                                                            type="monotone"

                                                            dataKey="current"

                                                            stroke={


colorBlindness === "protanopia"

                                                                    ?
"#00FFFF"

                                                                    :
colorBlindness === "deuteranopia"

                                                                        ?
"#1E90FF"

                                                                        :
"#0ea5e9"

                                                            }

                                                            strokeWidth={3}

                                                            dot={false}
```

```
isAnimationActive={false}
                                                        />
                                                    </LineChart>
                                                </ResponsiveContainer>
                                            </div>

                                            {/* Average Power Chart */}
                                            <div  className="bg-gray-900  bg-
opacity-90 p-8 rounded-xl shadow-xl hover:shadow-2xl transition-shadow duration-300">
                                                <h3 className="text-2xl font-
extrabold mb-6 flex items-center gap-3 text-white">
                                                    <svg
xmlns="http://www.w3.org/2000/svg"
                                                        className="h-7    w-7
text-green-600"
                                                        fill="none"
                                                        viewBox="0 0 24 24"
                                                        stroke="currentColor"
                                                    >
                                                        <path
strokeLinecap="round" strokeLinejoin="round" strokeWidth={2} d="M13 10V3L4 14h7v7l9-
11h-7z" />
                                                    </svg>
                                                    Average  Power  (W)  Over
Days
                                                </h3>
                                                <ResponsiveContainer
width="100%" height={300}>
                                                    <LineChart
data={powerChartData} margin={{ top: 10, right: 20, left: 0, bottom: 5 }}>
                                                        <CartesianGrid
strokeDasharray="4 4" stroke="#ffffff30" />
                                                        <XAxis dataKey="name"
stroke="#ffffffcc" tick={{ fontSize: 14, fontWeight: "600" }} />
                                                        <YAxis
stroke="#ffffffcc"  tick={{  fontSize:  14,  fontWeight:  "600"  }}  domain={["dataMin",
"dataMax"]} />
                                                        <Tooltip
labelFormatter={(v) => `Day: ${v}`}
```

```
                                                                    contentStyle={{
backgroundColor: "#1f2937", borderColor: "#4b5563", color: "#fff", fontSize: 14 }}
                                                                    itemStyle={{
color: "#facc15", fontWeight: "bold" }}
                                                                />
                                                                <Line
                                                                    type="monotone"
                                                                    dataKey="power"
                                                                    stroke={
colorBlindness === "protanopia"
                                                                            ?
"#E5FF00"
                                                                            :
colorBlindness === "deuteranopia"
                                                                                ?
"#FFA500"
                                                                                :
"#facc15"
                                                                    }
                                                                    strokeWidth={3}
                                                                    dot={false}
isAnimationActive={false}
                                                                />
                                                            </LineChart>
                                                        </ResponsiveContainer>
                                                    </div>
                                                </>
                                            );
                                        })()
                                    )}
                                </>
                            )}
                        </div>
                    </div>
```

A.3 WebSocket Data Handling

```
useEffect(() => {
        axios.get(`/api/device/${macAddress}`)
            .then(res => setDevice(res.data))
            .catch(err => console.error(err));

        // Fetch websocket data
        let socket;
        const connectWebSocket = () => {
            const ws = new WebSocket(
                "wss://olqh07fhd5.execute-api.ap-southeast-2.amazonaws.com/dev"
            );
            socket = ws;

            ws.onopen = () => {
                console.log("WebSocket connected");

                // Send macAddress to register
                ws.send(
                    JSON.stringify({
                        action: "sendMessage",
                        macAddress: macAddress,
                    })
                );
                setIsConnected(false); // still no data yet

            };

            let connectionTimeout;

            ws.onmessage = (event) => {
                try {
                    const raw = JSON.parse(event.data);
```

```
                    // Only consider real device data as connected
                    const hasValidData =
                        raw &&
                        (raw.cur !== undefined || raw.vol !== undefined || raw.pwr !==
undefined);

                    if (hasValidData) {
                        setIsConnected(true);

                        // Reset the timeout on each valid payload
                        clearTimeout(connectionTimeout);
                        connectionTimeout = setTimeout(() => {
                            setIsConnected(false); // No payload received in X seconds
                        }, 9000); // 7 seconds timeout

                       // Map incoming keys (cur, vol, pwr) to your internal structure
                        const mappedData = {
                            current: raw.cur,
                            voltage: raw.vol,
                            power: raw.pwr,
                            energy: raw.e_tot,
                            runtime: raw.run,
                            internal_temp: raw.int_t,
                            external_temp: raw.ext_t,
                        };

                        setDeviceData(mappedData);
                        handleWebSocketData(event.data);
                    }

                } catch (err) {
                    console.error("Error parsing WebSocket message:", err);
                    setIsConnected(false); // if parsing fails, show disconnected
                }
            };
```

```
    ws.onclose = (event) => {
        console.log("🔌 WebSocket closed", event.code);
        setIsConnected(false); // mark as disconnected
        // Optional: Reconnect on unexpected close
        if (event.code !== 1000) {
            setTimeout(connectWebSocket, 3000); //reconnect
        }
    };

    ws.onerror = (err) => {
        console.error("WebSocket error:", err);
        setIsConnected(false);
        ws.close();
    };
};

const fetchThreshold = async () => {
    try {
        const response = await axios.get(
            `/device/${macAddress}/threshold`
        );
        setThreshold(response.data.threshold_kWh);
        setNewThreshold(response.data.threshold_kWh);
    } catch (error) {
        console.error("Error fetching threshold:", error);
    }
};

fetchThreshold();
connectWebSocket();

return () => {
    if (socket && socket.readyState === WebSocket.OPEN) {
        socket.close(1000, "Page closed");
    }
```

```
        };

    }, [macAddress]);


useEffect(() => {
        let sysSocket; // keep reference

        const connectSystemWS = () => {
            const ws = new WebSocket(
                "wss://7hybghy3aj.execute-api.ap-southeast-2.amazonaws.com/dev/"
            );
            sysSocket = ws;

            ws.onopen = () => {
                console.log("System WebSocket connected");
                // Register macAddress
                ws.send(JSON.stringify({ action: "sendMessage", macAddress }));
            };

            ws.onmessage = (event) => {
                try {
                    const raw = JSON.parse(event.data);

                    // only process if it belongs to this mac
                    if (raw.id === macAddress) {
                        setSystemData({
                            free_heap: raw.fh,
                            min_free_heap: raw.mfh,
                            max_alloc_heap: raw.mah,
                            flash_chip_size_mb: raw.fcsm,
                            sketch_size_kb: raw.ssk,
                            free_sketch_space_kb: raw.fssk,
                            cpu_mhz: raw.cm,
                            chip_model: raw.cmdl,
```

```
                    chip_revision: raw.cr,
                    chip_cores: raw.cc,
                    uptime_sec: raw.us,
                    ping: raw.rssi,
                });
                setIsSystemConnected(true);
            }
        } catch (err) {
            console.error("Error parsing system WebSocket:", err);
        }
    };

    ws.onclose = () => {
        console.log("System WebSocket closed");
        setIsSystemConnected(false);
        // only reconnect if component is still mounted
        if (!ws._shouldClose) setTimeout(connectSystemWS, 3000);
    };

    ws.onerror = (err) => {
        console.error("System WS error:", err);
        ws.close();
    };

    // custom flag to prevent reconnection on unmount
    ws._shouldClose = false;
};

connectSystemWS();

return () => {
    if (sysSocket) {
        sysSocket._shouldClose = true; // prevent reconnection
        if (sysSocket.readyState === WebSocket.OPEN) {
            sysSocket.close(1000, "Page closed");
```

```
            }
        }
    };
}, [macAddress]);
```

Appendix B: Color themes change on User Interfaces

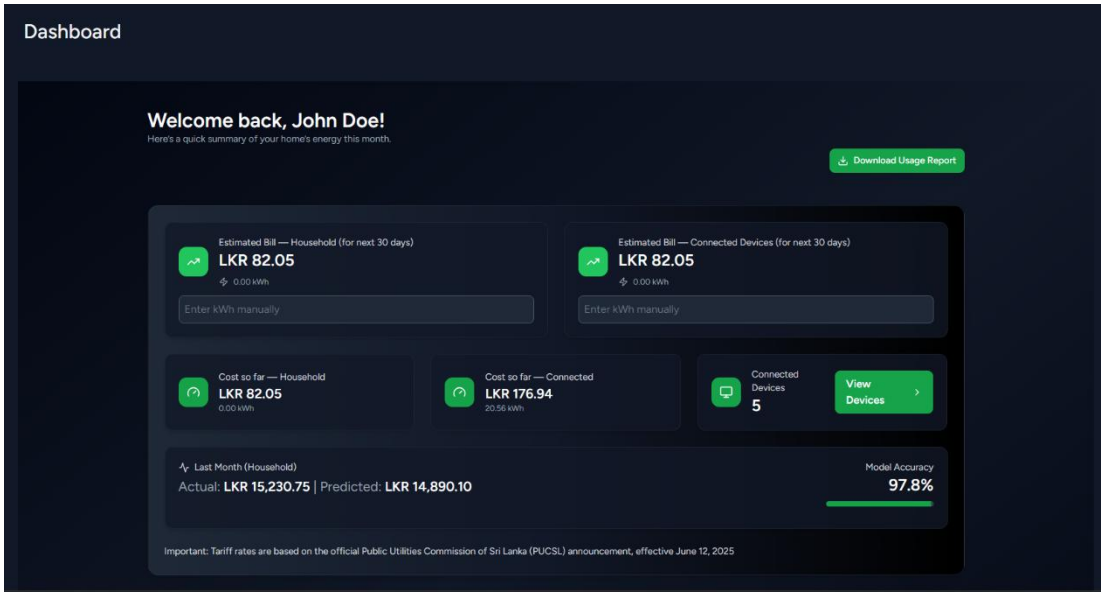


Figure 9: Default theme in Dashboard page.

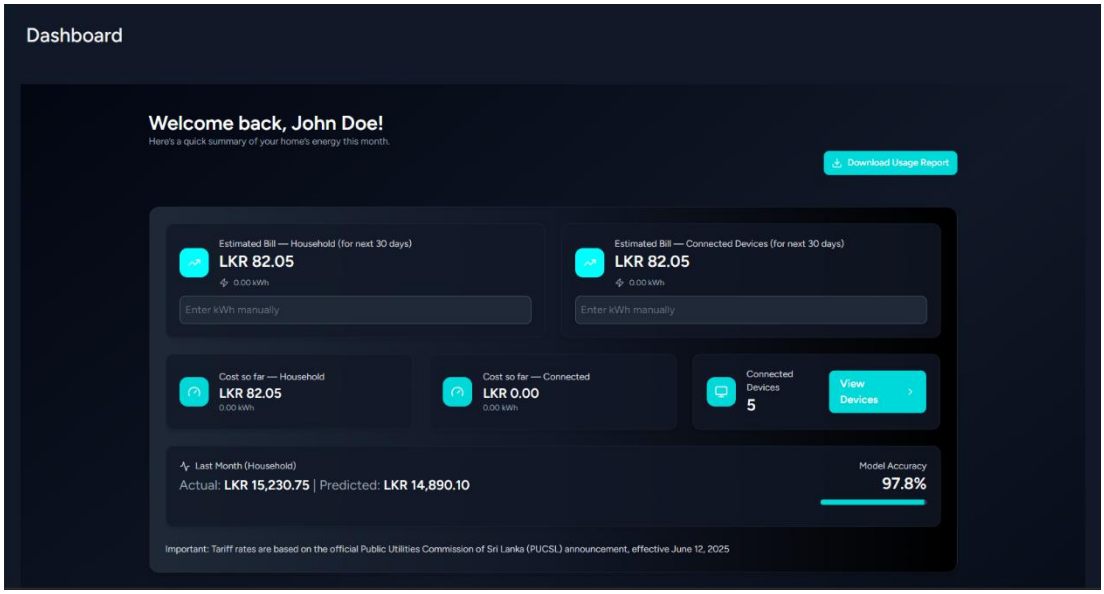


Figure 10: Protanopia color blind theme

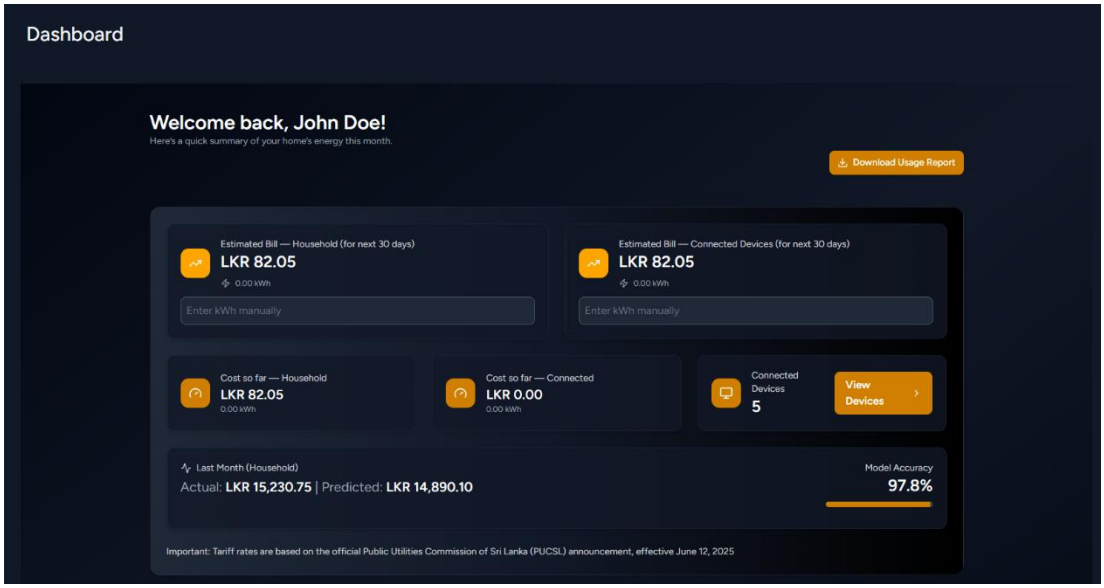


Figure 11: Deuteranopia color blind theme

Appendix C: Lambda functions

C.1 – Query handle in DynamoDB for a single device

```
import json
import boto3
from boto3.dynamodb.conditions import Key
from datetime import datetime
from zoneinfo import ZoneInfo
from decimal import Decimal
import calendar

dynamodb = boto3.resource("dynamodb")
table = dynamodb.Table("SmartPlug_6D_DataSet_Tbl")

TZ = ZoneInfo("Asia/Colombo")



def month_bounds(year: int, month: int):
    """Return start_ts, end_ts (epoch seconds) for a given month in Asia/Colombo."""
    start = datetime(year, month, 1, 0, 0, 0, tzinfo=TZ)
    _, last_day = calendar.monthrange(year, month)
    end = datetime(year, month, last_day, 23, 59, 59, tzinfo=TZ)
    return int(start.timestamp()), int(end.timestamp())



def decimal_to_float(val):
    if isinstance(val, Decimal):
        return float(val)
    try:
        return float(val)
    except Exception:
        return 0.0
```

```
def collect_cycles(mac: str, start_ts: int = None, end_ts: int = None):
    """
    Query records in chronological order and split into cycles.

    A new cycle is started when:
      - current int == 1 (explicit start),
      - OR current int <= previous int (reset / rollover),
      - OR this is the first record (we start the first cycle).
    This avoids treating '3' as start if it appears after 1 (i.e., mid-cycle).
    """
    params = {
        "TableName": table.name,
        "KeyConditionExpression": Key("id").eq(mac),
        "ProjectionExpression": "ts, cur, pwr, e_tot, #i",
        "ExpressionAttributeNames": {"#i": "int"},
        "ScanIndexForward": True,  # chronological
    }

    if start_ts is not None and end_ts is not None:
        params["KeyConditionExpression"] = Key("id").eq(mac) & Key("ts").between(start_ts, end_ts)

    last_key = None
    cycles = []
    current_cycle = []
    prev_int = None

    while True:
        if last_key:
            params["ExclusiveStartKey"] = last_key
        resp = table.meta.client.query(**params)
        items = resp.get("Items", [])

        for it in items:
            # defensive conversions
```

```
    try:
        ts = int(it.get("ts", 0))
    except Exception:
        ts = 0
    try:
        intval = int(it.get("int", 0))
    except Exception:
        intval = 0

    rec = {
        "ts": ts,
        "int": intval,
        "cur": decimal_to_float(it.get("cur", 0)),
        "pwr": decimal_to_float(it.get("pwr", 0)),
        "e_tot": decimal_to_float(it.get("e_tot", 0)),
    }

    if not current_cycle:
        # start the first cycle
        current_cycle.append(rec)
    else:
        # decide if this record begins a new cycle
        # start if explicit marker 1 OR int decreased/reset (<= prev_int)
        if intval == 1 or (prev_int is not None and intval <= prev_int):
            cycles.append(current_cycle)
            current_cycle = [rec]
        else:
            current_cycle.append(rec)

    prev_int = intval

last_key = resp.get("LastEvaluatedKey")
if not last_key:
    break
```

```
    # push last buffer
    if current_cycle:
        cycles.append(current_cycle)

    return cycles


def get_previous_cycle_averages(mac: str):
    """
    Return averages for the cycle BEFORE the most recent one (second-to-last).
    """
    cycles = collect_cycles(mac)

    if len(cycles) < 2:
        return {
            "average_current": 0,
            "average_power": 0,
            "samples": 0,
            "start_ts": None,
            "end_ts": None,
            "note": "Not enough cycles (need at least 2).",
        }

    prev_cycle = cycles[-2]  # second-to-last cycle

    count = len(prev_cycle)
    total_cur = sum(r["cur"] for r in prev_cycle)
    total_pwr = sum(r["pwr"] for r in prev_cycle)

    return {
        "average_current": round(total_cur / count, 5) if count else 0,
        "average_power": round(total_pwr / count, 7) if count else 0,
        "samples": count,
        "start_ts": prev_cycle[0]["ts"],
        "end_ts": prev_cycle[-1]["ts"],
```

```
    }


def get_monthly_energy(mac: str):
    """
    For the current calendar month, split cycles and sum the last e_tot of each cycle.
    """
    now = datetime.now(TZ)
    start_ts, end_ts = month_bounds(now.year, now.month)

    cycles = collect_cycles(mac, start_ts=start_ts, end_ts=end_ts)

    cycle_etots = []
    for c in cycles:
        if c:
            last_etot = c[-1].get("e_tot", 0)
            # only include positive/meaningful e_tot values
            if last_etot:
                cycle_etots.append(decimal_to_float(last_etot))

    total_energy = sum(cycle_etots)

    return {
        "cycles": len(cycle_etots),
        "cycle_etots": cycle_etots,
        "total_energy": round(total_energy, 6),
    }


def get_monthly_averages(mac: str):
    """
    For the current calendar month, return average current and power.
    """
    now = datetime.now(TZ)
    start_ts, end_ts = month_bounds(now.year, now.month)
```

```
    records = []
    params = {
        "TableName": table.name,
        "KeyConditionExpression":  Key("id").eq(mac)  &  Key("ts").between(start_ts,
end_ts),
        "ProjectionExpression": "ts, cur, pwr",
        "ScanIndexForward": True,
    }

    last_key = None
    while True:
        if last_key:
            params["ExclusiveStartKey"] = last_key
        resp = table.meta.client.query(**params)
        items = resp.get("Items", [])

        for it in items:
            records.append({
                "cur": decimal_to_float(it.get("cur", 0)),
                "pwr": decimal_to_float(it.get("pwr", 0)),
            })

        last_key = resp.get("LastEvaluatedKey")
        if not last_key:
            break

    count = len(records)
    if count == 0:
        return {
            "average_current": 0,
            "average_power": 0,
            "samples": 0,
        }

    total_cur = sum(r["cur"] for r in records)
```

```
    total_pwr = sum(r["pwr"] for r in records)

    return {
        "average_current": round(total_cur / count, 5),
        "average_power": round(total_pwr / count, 7),
        "samples": count,
    }


def lambda_handler(event, context):
    mac = (event.get("pathParameters", {}) or {}).get("mac")
    if not mac:
        return {
            "statusCode": 400,
            "headers": {"Content-Type": "application/json"},
            "body": json.dumps({"error": "MAC address is required"}),
        }

    prev_cycle = get_previous_cycle_averages(mac)
    monthly_energy = get_monthly_energy(mac)
    monthly_avg = get_monthly_averages(mac)

    body = {
        "mac": mac,
        "timezone": "Asia/Colombo",
        "previous_cycle": prev_cycle,
        "monthly_energy": monthly_energy,
        "monthly_average": monthly_avg,
    }

    return {
        "statusCode": 200,
        "headers": {"Content-Type": "application/json"},
        "body": json.dumps(body),
    }
```

C.2 – Query handle in Athena for a single device

```
import boto3
import os
import time
import json
from datetime import datetime, timedelta

ATHENA_DATABASE = os.environ["ATHENA_DATABASE"]
ATHENA_TABLE = os.environ["ATHENA_TABLE"]
ATHENA_OUTPUT = os.environ["ATHENA_OUTPUT"]

athena = boto3.client("athena")

def run_athena_query(query):
    """Run Athena query and return all rows (excluding header)"""
    response = athena.start_query_execution(
        QueryString=query,
        QueryExecutionContext={"Database": ATHENA_DATABASE},
        ResultConfiguration={"OutputLocation": ATHENA_OUTPUT},
    )
    query_execution_id = response["QueryExecutionId"]

    # Wait for completion
    while True:
        status = athena.get_query_execution(QueryExecutionId=query_execution_id)
        state = status["QueryExecution"]["Status"]["State"]
        if state in ["SUCCEEDED", "FAILED", "CANCELLED"]:
            break
        time.sleep(1)

    if state != "SUCCEEDED":
        reason = status["QueryExecution"]["Status"].get("StateChangeReason", "Unknown
reason")
```

```
            raise Exception(f"Query failed: {state} - {reason}")

    # Fetch results
    result = athena.get_query_results(QueryExecutionId=query_execution_id)
    rows = result["ResultSet"]["Rows"]

    # Convert rows to list of dicts
    headers = [col["VarCharValue"] for col in rows[0]["Data"]]
    data = []
    for row in rows[1:]:
        values = [col.get("VarCharValue", None) for col in row["Data"]]
        data.append(dict(zip(headers, values)))

    return data

def lambda_handler(event, context):
    mac = event.get("pathParameters", {}).get("mac")
    if not mac:
        return {"statusCode": 400, "body": "MAC address required"}

    ###  Monthly Averages Query (for month selection dropdown)
    month_query = f"""
        SELECT
              date_trunc('month', from_unixtime(ts) AT TIME ZONE 'Asia/Colombo') AS
month,
            AVG(CAST(cur AS DOUBLE)) AS avg_current,
            AVG(CAST(pwr AS DOUBLE)) AS avg_power
        FROM {ATHENA_TABLE}
        WHERE id = '{mac}'
          AND from_unixtime(ts) AT TIME ZONE 'Asia/Colombo' >= date_add('month', -3,
date_trunc('month', current_timestamp AT TIME ZONE 'Asia/Colombo'))
        GROUP BY 1
        ORDER BY month DESC
    """
    monthly_results = run_athena_query(month_query)
    months_data = [
```

```
        {
            "month": row["month"],
            "avg_current": float(row["avg_current"]) if row["avg_current"] else 0,
            "avg_power": float(row["avg_power"]) if row["avg_power"] else 0,
        }
        for row in monthly_results
    ]

    ###  Daily Averages Query (for charts over days)
    daily_query = f"""
        SELECT
            date_format(from_unixtime(ts) AT TIME ZONE 'Asia/Colombo', '%Y-%m-%d') AS
day,
            AVG(CAST(cur AS DOUBLE)) AS avg_current,
            AVG(CAST(pwr AS DOUBLE)) AS avg_power
        FROM {ATHENA_TABLE}
        WHERE id = '{mac}'
          AND from_unixtime(ts) AT TIME ZONE 'Asia/Colombo' >= date_add('month', -4,
date_trunc('month', current_timestamp AT TIME ZONE 'Asia/Colombo'))
        GROUP BY 1
        ORDER BY day ASC
    """
    daily_results = run_athena_query(daily_query)

    # Build daily dictionary
    data_dict = {
        row["day"]: {
            "avg_current": float(row["avg_current"]) if row["avg_current"] else 0,
            "avg_power": float(row["avg_power"]) if row["avg_power"] else 0,
        }
        for row in daily_results
    }

    # Generate last 3 full months daily data (excluding current month)
    today = datetime.now()
    daily_avg_data = {}
```

```
    for m in range(1, 4):  # 1 to 3 → skip current month
        # Get the first day of the month 'm' months ago
          month_date = (today.replace(day=1) - timedelta(days=1)).replace(day=1) -
timedelta(days=30*(m-1))
        month_str = month_date.strftime("%Y-%m")

        # Calculate number of days in the month correctly
        next_month = (month_date.replace(day=28) + timedelta(days=4)).replace(day=1)
        days_in_month = (next_month - timedelta(days=1)).day

        daily_avg_data[month_str] = {}
        for day in range(1, days_in_month + 1):
            day_key = f"{month_str}-{str(day).zfill(2)}"
            if day_key in data_dict:
                daily_avg_data[month_str][day_key] = data_dict[day_key]
            else:
                 daily_avg_data[month_str][day_key] = {"avg_current": 0, "avg_power":
0}

    # Final response
    response_body = {
        "mac": mac,
        "months": months_data,               # For dropdown / last month charts
       "daily_avg_data": daily_avg_data     # For daily charts over last 3 full months
    }

    return {
        "statusCode": 200,
        "headers": {"Content-Type": "application/json"},
        "body": json.dumps(response_body, default=str),
    }
```

C.3 – Query handle in DynamoDB for a multiple device

```
import json
import boto3
from boto3.dynamodb.conditions import Key
from datetime import datetime
from zoneinfo import ZoneInfo
from decimal import Decimal
import calendar

# DynamoDB setup
dynamodb = boto3.resource('dynamodb')
table = dynamodb.Table('SmartPlug_6D_DataSet_Tbl')

TZ = ZoneInfo("Asia/Colombo")


def decimal_to_float(val):
    if isinstance(val, Decimal):
        return float(val)
    try:
        return float(val)
    except Exception:
        return 0.0


def month_bounds(year: int, month: int):
    start = datetime(year, month, 1, 0, 0, 0, tzinfo=TZ)
    _, last_day = calendar.monthrange(year, month)
    end = datetime(year, month, last_day, 23, 59, 59, tzinfo=TZ)
    return int(start.timestamp()), int(end.timestamp())


def collect_cycles(mac: str, start_ts=None, end_ts=None):
    """
    Fetch records and split into cycles.
    Cycle starts when:
```

```
      - int == 1 (explicit start)
      - int <= previous int (rollover/reset)
      - first record
    """
    params = {
        "TableName": table.name,
        "KeyConditionExpression": Key("id").eq(mac),
        "ProjectionExpression": "ts, e_tot, #i",
        "ExpressionAttributeNames": {"#i": "int"},
        "ScanIndexForward": True
    }

    if start_ts is not None and end_ts is not None:
        params["KeyConditionExpression"] = Key("id").eq(mac) &
Key("ts").between(start_ts, end_ts)

    last_key = None
    cycles = []
    current_cycle = []
    prev_int = None

    while True:
        if last_key:
            params["ExclusiveStartKey"] = last_key
        resp = table.meta.client.query(**params)
        items = resp.get("Items", [])

        for it in items:
            ts = int(it.get("ts", 0))
            intval = int(it.get("int", 0))
            e_tot = decimal_to_float(it.get("e_tot", 0))
            rec = {"ts": ts, "int": intval, "e_tot": e_tot}

            if not current_cycle:
                current_cycle.append(rec)
            else:
```

```
                if intval == 1 or (prev_int is not None and intval <= prev_int):
                    cycles.append(current_cycle)
                    current_cycle = [rec]
                else:
                    current_cycle.append(rec)

            prev_int = intval

        last_key = resp.get("LastEvaluatedKey")
        if not last_key:
            break

    if current_cycle:
        cycles.append(current_cycle)

    return cycles


def calculate_cycles_energy(cycles):
    """Sum last e_tot of each cycle."""
    total_energy = sum(cycle[-1]["e_tot"] for cycle in cycles if cycle)
    return round(total_energy, 6)


def lambda_handler(event, context):
    # Parse input JSON
    macs = []
    try:
        if "body" in event and isinstance(event["body"], str):
            body = json.loads(event["body"])
            macs = body.get("macs", [])
        elif "macs" in event and isinstance(event["macs"], list):
            macs = event["macs"]
    except Exception as e:
        print("Error parsing input:", e)
```

```
    return {
        "statusCode": 400,
        "headers": {"Content-Type": "application/json"},
        "body": json.dumps({"error": "Invalid input"})
    }

if not macs:
    return {
        "statusCode": 400,
        "headers": {"Content-Type": "application/json"},
        "body": json.dumps({"error": "No MAC addresses provided"})
    }

now = datetime.now(TZ)
start_ts, end_ts = month_bounds(now.year, now.month)

results = {}
overall_total = 0

for mac in macs:
    cycles = collect_cycles(mac, start_ts=start_ts, end_ts=end_ts)
    total_energy = calculate_cycles_energy(cycles)
    results[mac] = total_energy
    overall_total += total_energy

return {
    "statusCode": 200,
    "headers": {"Content-Type": "application/json"},
    "body": json.dumps({
        "per_mac_total_energy": results,
        "overall_total_energy": round(overall_total, 6)
    })
}
```

C.4 – Query handle in Athena for a multiple device

```
import boto3
import os
import time
import json

ATHENA_DATABASE = os.environ["ATHENA_DATABASE"]
ATHENA_TABLE = os.environ["ATHENA_TABLE"]
ATHENA_OUTPUT = os.environ["ATHENA_OUTPUT"]

athena = boto3.client("athena")

def run_athena_query(query, poll_interval=1):
    # Start query execution
    resp = athena.start_query_execution(
        QueryString=query,
        QueryExecutionContext={"Database": ATHENA_DATABASE},
        ResultConfiguration={"OutputLocation": ATHENA_OUTPUT},
    )
    qid = resp["QueryExecutionId"]

    # Wait until query completes
    while True:
        status = athena.get_query_execution(QueryExecutionId=qid)
        state = status["QueryExecution"]["Status"]["State"]
        if state in ("SUCCEEDED", "FAILED", "CANCELLED"):
            break
        time.sleep(poll_interval)

    if state != "SUCCEEDED":
        reason = status["QueryExecution"]["Status"].get("StateChangeReason",
"Unknown")
        raise Exception(f"Athena query failed: {state} - {reason}")
```

```
    # Fetch results
    result = athena.get_query_results(QueryExecutionId=qid)
    rows = result["ResultSet"]["Rows"]

    # First row is header
    header = [c.get("VarCharValue") for c in rows[0]["Data"]]
    data_rows = []
    for row in rows[1:]:
        values = [c.get("VarCharValue") for c in row["Data"]]
        data_rows.append(dict(zip(header, values)))

    return data_rows


def lambda_handler(event, context):
    print("Received event:", event)  # Debug log

    # Handle API Gateway v2 (body as string) OR direct Lambda Test JSON
    body = event
    if "body" in event and event["body"]:
        try:
            body = json.loads(event["body"])
        except Exception as e:
            return {
                "statusCode": 400,
                "body": json.dumps({"message": f"Invalid JSON body: {str(e)}"})
            }

    macs = body.get("macs", [])
    month = body.get("month")

    if not macs:
        return {
            "statusCode": 400,
            "body": json.dumps({"message": "Please provide 'macs' in event JSON"})
```

```
    }

# Build SQL IN clause
mac_list_sql = ",".join(f"'{m}'" for m in macs)

query = f"""
WITH month_data AS (
    SELECT
        id,
        cur,
        from_unixtime(ts) AS ts_dt
    FROM {ATHENA_TABLE}
    WHERE id IN ({mac_list_sql})
    AND month(from_unixtime(ts)) = {month}
)
SELECT
    id,
    avg(cur) AS avg_current
FROM month_data
WHERE year(ts_dt) = (
    SELECT max(year(from_unixtime(ts)))
    FROM {ATHENA_TABLE}
    WHERE month(from_unixtime(ts)) = {month}
)
GROUP BY id
"""

try:
    results = run_athena_query(query)
    return {
        "statusCode": 200,
        "headers": {"Content-Type": "application/json"},
        "body": json.dumps(results, default=str)
    }
except Exception as e:
```

```
return {
    "statusCode": 500,
    "body": json.dumps({"error": str(e)})
}
```