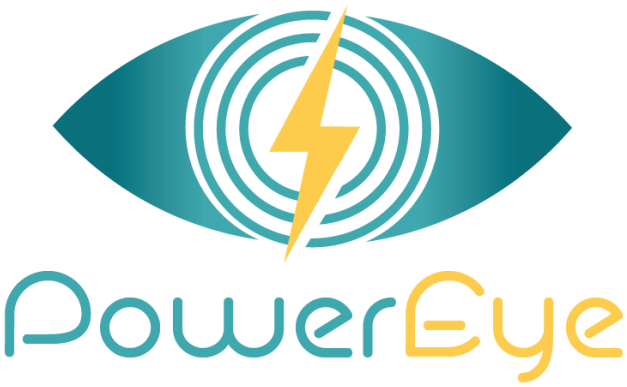
Final Report



Home Energy Monitoring System (HEMS)

Prince Sultan University

College of Computer & Information Science

Senior Project 2 (CS 499)

First Semester, term 231

Supervised By:

Dr. Layla Alfawzan & Dr. Sofianiza Abd Malik

Group Members:

SE Team

Maya Emar	219410523
Ward Najjar	220410019
Joody Koujan	219410402
Nour Al Akhras	220410351
Renad Almahmoud	219410575

CS Team

Reem Hejazi	219410002
Sarah Alshagawi	219410319
Raneem Balharith	219410305

Tables of Contents:

• List of Topics:

• List of Tables:

• List of Figures:

ACKNOWLEDGMENTS:

Our heartfelt thanks go out to each of these individuals for their unwavering support and expertise, which have been integral to the success of the PowerEye project:

First and foremost, we would like to express our sincere appreciation to Dr. Sofianiza Abd Malik and Dr. Layla Alfawzan, our dedicated supervisors. Their invaluable continuous guidance, unwavering support, and insightful feedback have played a crucial role throughout the development of this project.

A special note of thanks to Dr. Anees Ara and Prof. Mohammed Akour, our esteemed examiners, for generously dedicating their time and efforts to evaluate this paper.

Additional recognition goes Dr. Anees Ara for her indispensable assistance in system design. Her crucial insights significantly contributed to the overall success of this paper, and her continuous support during this semester played a pivotal role in navigating challenges and ensuring progress.

We would like to convey our heartfelt appreciation to Dr. Suad AlRamouni, whose contributions to the Software Engineering (SE) team played a pivotal role in the success of this paper. We are grateful for her expertise and collaborative spirit.

Special thanks to Dr. Roohi Jan and Ms. Ishrat Khan for their expertise and contributions in the database workshops. Their efforts enriched our understanding and implementation of robust NoSQL database structures within the system.

Finally, sincere acknowledgement goes to Dr. Abrar Wafa for her valuable feedback on the Machine Learning (ML) notebook. Her insights played a crucial role in enhancing the accuracy and effectiveness of our predictive models.

Thank you all for your unwavering support and expertise.

PROPOSAL

Introduction to the project with problem definition and background information

Introduction:

Home Energy Monitoring System (HEMS) is a system that collects energy data from various household appliances, processes and analyzes them, and presents the findings to users to give them insights about their energy consumption for better energy sustainability. HEMS is categorized into two main types: Non-Intrusive Load Monitoring (NILM) systems and Intrusive Load Monitoring (ILM) systems. NILM systems allow the identification and tracking of individual electrical appliances within a household without requiring additional sensors or intrusive measurements. On the other hand, ILM systems involve the installation of dedicated sensors directly on individual appliances or loads to monitor their energy usage.

Problem Statement:

Due to the increase in the usage of devices that need energy at home nowadays, effective energy monitoring has become a crucial concern. Although there are a lot of energy monitoring systems that help in controlling usage, most of them need professional support for the configuration like Sense, Neurio, etc. Also regarding smart plugs apps, most of them include only the basic features. In our case, we are going to develop an app that you can set up by yourself with some advanced features that most smart plugs apps don't have.

Project Objectives:

Our main objective is to develop an application that helps people to monitor their energy consumption. Also, presenting a summary of the usage through different periods and giving alerts and recommendations based on that usage. We aim to make the app clear, smooth, and easy to use for the majority of users.

Proposed Solution:

We aim to build a user-friendly mobile application that empowers people to effortlessly monitor their energy usage on an appliance level. With our app, users can set energy consumption goals and easily manage their devices and rooms. With real-time energy usage data at their fingertips, users gain valuable insights to make informed decisions about their energy usage. Our app also provides notifications to alert users of when they exceed their set limits, along with personalized energy-saving recommendations to encourage sustainable practices. All people, even with minimal technical knowledge, will have the ability to set up the plugs and devices with no need for technical support, and with interactive charts, the app offers an engaging and effortless experience in achieving energy efficiency.

Target Audience:

The target audience for our system is diverse and includes individuals with a range of interests, from cost savings and energy efficiency to environmental stewardship and technological innovation. These systems play a crucial role in empowering homeowners to take control of their energy usage and contribute to a more sustainable and eco-friendly living environment.

Related Works:

Sense and Neurio are examples of NILM-based HEMSs. These systems leverage AI algorithms to disaggregate the appliances, although it can be time-consuming and yield less accurate results (Altrabalsi et al, 2016). Both systems offer advanced features such as energy usage visualization, personalized recommendations, and cost estimation. However, professional assistance may be required to set up the monitor and the gateway. Emporia follows a similar approach, but they also place current transformers on the electrical sublines, allowing for more detailed monitoring.

On the other hand, commercial smart plugs like Kasa, Meross, and Wemo are examples of ILM-based solutions. While they don't offer advanced energy monitoring features, they provide scheduling and control capabilities for appliances and are easy to set up. However, unlike NILM systems, they cannot monitor the total energy consumption of an entire household.

In the Arabian Gulf, finding specific HEMS solutions can be challenging. However, companies like Legrand, Alfanar, and Circuits offer monitoring solutions not explicitly focused on HEMS. These companies focus on monitoring data centers, utility metering, and energy efficiency. This highlights the limited availability of dedicated HEMS solutions within the local market, making consumers seek international providers or alternative approaches.

Table 1: *Similar Works*

App	Appliance level monitoring	Real time energy monitoring	Historical Energy Usage Data	Data visualization	Cost estimation	No need for gateway/hub	No need for technical support	Scheduling and control	Personalized Recommendation
Sense	x[1]	x	x	x	x				x
Neurio	x[2]	x	x	x	x				x
Emporia		x	x	x					x
Kasa	x	x	x	x		x	x	x	
Meross	x	x	x	x		x	x	x	
Wemo	x	x	x		x	x	x	x	
Legrand		x	x	x					
Alfanar		x	x	x					
Circuit		x	x	x	x			x	x

Our Contribution:

Our system will utilize smart plugs, making it an ILM system. While most of the existing ILM systems in the market lack advanced energy monitoring features, our system will be a unique combination of the key features from both ILM and NILM. It will offer an easy setup, and appliance control, as well as advanced features typically found in NILM systems, such as data visualization, cost estimation, and personalized recommendations.

In addition to the mentioned key features, our system will also offer the following distinctive features:

- **Energy Recommendations:** Our system will provide users with personalized recommendations and tips about energy consumption and CO2 emissions, promoting energy sustainability and empowering users to make informed choices.
- **AI Features:** Our system will incorporate AI-driven capabilities for advanced data analysis, including anomaly detection, and personalized recommendations.

[1] Sense and Neurio use load disaggregation to identify appliances, however not every appliance is detected.

- **Local brand:** As the first of its kind in the region, we are the pioneering local brand for HEMS.
- **User-Friendly Interface:** Our system features a user-friendly interface that simplifies energy consumption insights, making energy information easily accessible even to those with limited knowledge about energy.

Scope of the Product:

- **User Account Management:**
  - Account signup, login, logout, deletion.
  - Display/edit profile information.
  - Set/track energy/cost goals monthly.
- **Device Management:**
  - Add/delete smart plugs and specify their appliances.
  - Add/delete rooms.
  - Customize names for the appliances/rooms.
  - Define appliance types.
  - Switch on/off appliances individually, or by room.
  - Indication of device connection status.
  - Display the status of the connected appliances (on/off).
- **Energy Consumption Analysis:**
  - Real-time power usage per appliance.
  - Present different historical data types (accumulated energy-cost-CO_2) for total, per room, and per appliance.
  - Display historical data by timeframe (daily-weekly-monthly-yearly) for total, per room, and per appliance.
- **Notifications and Alerts:**
  - Alerts for surpassing the specified energy consumption target and its percentage.
  - Personalized recommendations for energy optimization (e.g., alerts for devices in phantom mode or higher energy usage).
- **Information and Support:**
  - Easy-to-follow user guide for common issues.
- **User Interface:**
  - Implement intuitive and user-friendly interface elements with minimal text.
  - Ensure readability of energy analysis for most users.
- **Data Visualization:**

- Integration of interactive graphs featuring filter buttons, enabling users to refine the historical data display by duration, timeframe, or data category(accumulated energy-cost-CO2).
- Utilization of various graphical formats such as pie, bar, and line charts.

- **Security:**
  - Implement secure user and device authentication protocols (token-based).
  - Hash passwords.
- **System Features:**
  - Incorporate an error-handling mechanism.
  - Utilize caching mechanisms in the mobile app for temporary offline usage.
  - Meross Cloud integration.

Project Plan:

Table 2: *Project Planning*

Phase	Iteration	Task	Expected Deliverables
Planning	1	Define the problem, conduct product comparison, and define project scope and objectives	Project proposal
Requirement	1	Collect initial requirements through surveys and interviews.	Software Requirements Specification document.
	2	Refine and prioritize the requirements based on feedback, and document the finalized list of requirements.	
Design	1	Develop an architectural design, and refine it based on feedback.	Software Architectural and Detailed Design document.
	2	Select design patterns, and make necessary refinements.	
Implementation	1	Develop the highest priority features determined by the stakeholders.	Prototype, working software application, and code documentation.
	2	Develop the lower priority features determined by the stakeholders.	
Testing	1	- Develop and execute unit test cases. - Review and refine the test cases based on test results and address any identified issues or failures.	Test cases, test results report, and user acceptance test report.
	2	- Develop and execute integration test cases. - Review and refine the test cases based on test results and address any identified issues or failures.	
	3	- Conduct user acceptance and system testing.	

		- Address any identified issues or failures.	
Deployment	1	- Select the appropriate deployment platform, configure the software system, and deploy it. - Address issues and bugs during and after deployment. - Write installation guide and user manual.	Installation guide and user manual.

Table 3: *Timeline*

Week	Start Date	End Date	Tasks to Be Completed	Task Description
1	Aug 27	Sep 2	Project Signup (Completed Form)	Students are assigned team members and a project supervisor.
2	Sep 3	Sep 9	Work Plan and Budgeting	Include a detailed schedule, project implementation plan, assessment/testing plan, and budgeting.
3	Sep 10	Sep 16	Work Plan and Budgeting	Include a detailed schedule, project implementation plan, assessment/testing plan, and budgeting.
4	Sep 17	Sep 23	Requirements Collection and Analysis	Gather and analyze project requirements.
5	Sep 24	Sep 30	Requirements Collection and Analysis	Gather and analyze project requirements.
6	Oct 1	Oct 7	Design	Develop High-Level and Low-Level Design documents.
7	Oct 8	Oct 14	Design	Develop High-Level and Low-Level Design documents.
8	Oct 15	Oct 21	Midterm Report Submission	Submit a report with details from Project Signup, Work Plan, Requirements Analysis, and Design.

Figure 1: *Timeline*

Team Member Roles:

Table 4: *Team Member Roles*

Name	Role
Rem Hejazi	System designer, back-end developer (tracker server-AI)
Nour Al Akhras	System designer, back-end developer (web server)
Maya Emar	Database designer, back-end developer (web server)
Sarah Alshagawi	Business analyst, back-end developer (tracker server)
Renad Almahmoud	Front-end developer, UI/UX designer
Raneem Balharith	Front-end developer, UI/UX designer
Joody Koujan	Front-end developer, UI/UX designer
Ward Najjar	Front-end developer, UI/UX designer

Risk Assessment and Mitigation Plan:

Table 5: *Risk Likelihood and Impact*

Likelihood/ Probability	**Impact**				
	Very Low	**Low**	**Medium**	**High**	**Very High**
Very High	Low	Medium	High	Very High	Very High
High	Low	Medium	Medium	High	Very High
Medium	Low	Low	Medium	Medium	High
Low	Very Low	Low	Low	Medium	Medium
Very Low	Very Low	Very Low	Low	Low	Low

Table 6: *Risk Assessment*

Issue	Risk	Likelihood	Impact	Risk Level	Mitigation
Data Privacy and Security	Unauthorized access to sensitive user data, data breaches, or cyber-attacks.	Medium	High	High	- Employ secure authentication mechanisms, such as token-based authentication. - Authorization (admin and user). - Hashing the users' passwords
Data Accuracy	Inaccurate energy usage measurements due to sensor malfunctioning, or environmental factors.	Medium	High	High	- Error-handling mechanism to identify and manage errors during data collection, processing, and storage.
Power Outages or Network Connectivity Issues	Power or network disruptions affecting data collection and system functionality.	High	High	Very High	- Using caching mechanisms in the mobile app for temporary offline usage. - Notifying the user when a device is disconnected from electricity or the Internet.
System Scalability	Inability of the system to handle increasing user demand or growing data volumes.	Medium	High	High	- Design the system to be scalable by employing cloud-based infrastructure.
User Data Loss	Loss of user data due to system failures, user error, or accidental deletion.	Low	High	Medium	- Warning the users regarding data management and deletion. (when deleting account, room, or device)

Software Process:

We will utilize the Iterative Waterfall model, involving multiple iterations in each phase, for flexible and continuous improvement. In requirements Specification, we iteratively gather, analyze, and enhance system requirements. In design, we iteratively create architectural and detailed designs based on software requirements, design standards, and best practices. In Implementation, we adopt iterative cycles to construct the system. In testing, we iteratively assess and rectify failed tests, including acceptance testing. In software deployment, we iteratively deploy the system, addressing bugs through testing and quality assurance.

REQUIREMENTS SPECIFICATION

Functional and non-functional requirements of the system with the requirements iterations

Functional Requirements:

User Authentication

Registration:

FR-1: When a user attempts to register, the PowerEye system shall prompt the user to enter the following information:

- Valid Merros credentials (email and password).
- A separate password for the PowerEye system.

FR-2: The PowerEye system shall validate the provided PowerEye password to ensure it meets the following complexity criteria:

- The password is at least eight characters long.
- The password includes at least one uppercase letter.
- The password includes at least one lowercase letter.
- The password includes at least one numeric digit.
- The password includes at least one special character (e.g., !, @, #, $).

FR-3: If the provided password does not meet the complexity criteria, the PowerEye system shall display an error message, explaining the password requirements.

FR-4: The PowerEyen system shall validate the provided Meross credentials (email and password) to ensure they are associated with a valid Meross account.

FR-5: If the provided Meross credentials are not valid, the PowerEye system shall display an error message indicating that the Meross credentials are invalid.

Login:

FR-6: When a user attempts to log in, the PowerEye system shall prompt the user to enter their registered Merros email and PowerEye password.

FR-7: The PowerEye system shall validate the provided email and password against the database records.

FR-7.1: If the email is empty or doesn't match any registered user, the PowerEye system shall display an error message indicating incorrect login credentials.

FR-7.2: If the provided password does not match the stored hash, the PowerEye system shall display an error message indicating incorrect login credentials.

FR-8: If the provided email and password both pass validation, the PowerEye system shall grant access to the user's account.

FR-9: The system shall maintain a user's logged-in status for a period of one month without necessitating reauthentication.

Logout:

FR-10: When a logged-in user attempts to Logout, the PowerEye system shall terminate the user's current session.

User Account Management

Deleting user account:

FR-11: When a logged-in user attempts to delete their account, the PowerEye system shall display a confirmation dialog box to ensure user intent.

FR-12: If the user confirms their intent to delete the account, the PowerEye system shall release the associated email address and make it available for use in future registrations.

FR-13: Upon successful deletion of the account, the PowerEye system shall display a confirmation message and terminate the user's current session.

Viewing and Editing user profile:

FR-14: When a logged-in user attempts to view their account information, The PowerEye system shall provide access to the user to view their information.

FR-15: User information shall include the following fields:

- Merros email.
- Merros password.
- PowerEye password.
- Username.
- Profile picture.

FR-16: The user shall be able to edit the following fields of their user information:

- Merros password.
- PowerEye password.
- Username.
- Profile picture.

FR-17: The user shall have the ability to save the edited information, and the system shall update the corresponding fields in the database.

FR-18: The user shall have the option to cancel or discard any unsaved changes.

FR-19: If the user chooses not to edit a specific field, the current information shall be retained.

FR-20: The system shall validate any changes made to the PowerEye password to ensure they meet the password complexity requirements.

FR-21: The system shall validate any changes made to the Meross password to ensure they are associated with a valid Meross account.

Cost Goal Management

Setting Cost Goal:

FR-22: The PowerEye system shall enable logged-in users to establish a monthly energy consumption cost goal (upper limit) by specifying their desired cost.

FR-23: The system shall validate the user's input for the monthly energy consumption cost goal against the following criteria:

- The input must be a numeric value.
- The input must be a positive value.
- The input must be greater than or equal to the total energy cost incurred by the user since the beginning of the current calendar month.

FR-24: After the user successfully sets a new goal, the system shall display a percentage-based comparison between the newly set goal and the user's total energy cost for the previous calendar month.

FR-25: The system shall store the cost goal in the database and associate it with the user's account.

Modifying Goals:

FR-26: The PowerEye system shall allow users to modify their energy consumption goal after it has been initially set.

FR-27: When modifying the energy consumption goal, the system shall validate the new input against the validation criteria specified (see Setting Cost Goal).

FR-28: After the user successfully sets a new goal, the system shall display a percentage-based comparison between the newly set goal and the user's total energy cost for the previous calendar month.

Deleting Goals:

FR-29: The PowerEye system shall allow users to delete their energy consumption goal after it has been initially set.

FR-30: The system shall prompt the user to confirm the deletion of their energy consumption goal.

FR-31: Upon user confirmation, the PowerEye system shall delete the energy consumption goal.

FR-32: The system shall update the tracking and comparison processes to reflect the absence of a specified goal.

FR-33: In case the user chooses not to confirm the deletion, the PowerEye system shall cancel the deletion process and retain the existing energy consumption goal.

Viewing Historical Progress - Energy Goal Tracker:

FR-34: The PowerEye system shall present real-time tracking of the percentage of monthly energy consumption costs compared to the user's specified goal for the current calendar month.

FR-35: At the start of each new month the progress percentage shall be reset to zero.

Appliance Management

Adding a New Appliance:

FR-36: Upon receiving a user request to add a new appliance, the PowerEye system shall retrieve a list of all smart plugs linked to the user's Meross account.

FR-37: If there's an error in retrieving the list of smart plugs, the system shall display an error message indicating that there was an issue in retrieving the list of smart plugs.

FR-38: If the user's Meross account is not linked with any smart plugs, the PowerEye system shall display a message indicating that the user's Meross account does not have any linked smart plugs.

FR-39: The PowerEye system shall present the user with a list of smart plug names from their Meross account, indicating only the available ones (i.e., not already linked with any appliance) as selectable options.

FR-40: The user shall be able to choose one available smart plug from the list at a time.

FR-41: For each selected smart plug, the PowerEye system shall prompt the user to specify the appliance name and type. The user can choose from a predefined list of type options (e.g., "Lamp," "Fan," "Coffee Maker," etc.).

FR-42: The PowerEye system shall validate the following:

- An appliance name is provided for each smart plug.
- The appliance name is unique among all appliances in the user's account.
- The appliance name consists of 2 or more characters.

FR-43: If the name is invalid, the PowerEye system shall display an error message indicating the issue.

FR-44: After successful configuration, the PowerEye system shall update the list of appliances, displaying the selected names and types.

FR-45: The PowerEye system shall provide the user with the option to cancel the addition process at any stage.

FR-46: If the user chooses to cancel the addition, the PowerEye system shall abort the process, ensuring that no changes are saved.

Deleting an appliance:

FR-47: When a user attempts to delete an appliance from the PowerEye system, the system shall prompt the user for confirmation before proceeding with the deletion.

FR-48: If confirmed, the PowerEye system shall remove the appliance, disassociate it from any rooms, and release its connected smart plug.

FR-49: The user interface shall update to reflect the removal of the appliance from the list of available devices.

FR-50: If the user chooses to cancel the deletion, the PowerEye system shall abort the process, ensuring that no changes are saved.

Updating an appliance (changing name):

FR-51: When the user attempts to change the name of an appliance, the PowerEye system shall allow the user to modify the existing name by entering a new name.

FR-52: The PowerEye system shall validate the new name to ensure it meets the naming criteria specified (See Adding New Appliance).

FR-53: If the user attempts to enter an invalid name, the system shall display an error message indicating the issue and prevent the update until a valid name is provided.

FR-54: Once the user enters a valid new name and confirms the update, the system shall save the changes and update the appliance name in the database.

FR-55: The PowerEye system shall provide the user with the option to cancel the name change process at any stage.

FR-56: If the user chooses to cancel the name change, the PowerEye system shall abort the process, ensuring that no changes are saved.

Controlling an appliance (On/Off):

FR-57: When a user chooses to turn on/off an appliance, the PowerEye system shall apply the selected on/off action to the chosen appliance.

FR-58: The PowerEye system shall update the status (on/off) of the selected appliance.

Room Management

Creating a New Room:

FR-59: When a user initiates the creation of a new room, the PowerEye system shall prompt the user to enter a room name and select one or more appliances from a list of the existing devices (i.e., those already added to the PowerEye system).

FR-60: The PowerEye system shall validate the following:

- A room name is provided for each smart plug.
- The room name is unique among all rooms in the user's account.
- The room name consists of 2 or more characters.

FR-61: The PowerEye system shall validate that the user has selected one or more appliances from the list.

FR-62: After successful validation, the PowerEye system shall update the list of available rooms by adding a new room instance with the provided name and associate the selected appliances with it.

Deleting a Room:

FR-63: If a user selects the delete option for a room, the PowerEye system shall prompt the user for confirmation before proceeding with the deletion.

FR-64: If confirmed, the PowerEye system shall remove the room name from the list of available rooms while maintaining its associated appliances.

FR-65: The list of rooms shall promptly be updated to reflect the removal of the room.

FR-66: If the user chooses to cancel the deletion, the PowerEye system shall abort the process, ensuring that no changes are saved.

Updating a Room:

FR-67: When the user attempts to change the name of an existing room, the PowerEye system shall allow the user to modify the existing name by entering a new name.

FR-68: The PowerEye system shall validate the new name to ensure it meets the naming criteria specified (See Creating a new room).

FR-69: If the user attempts to enter an invalid name, the system shall display an error message indicating the issue and prevent the update until a valid name is provided.

FR-70: Once the user enters a valid new name and confirms the update, the system shall save the changes and update the room name in the database.

FR-71: The user interface shall update to reflect the new name of the room.

FR-72: The PowerEye system shall provide the user with the option to cancel the name change process at any stage.

FR-73: If the user chooses to cancel the name change, the PowerEye system shall abort the process, ensuring that no changes are saved.

Updating Room's Appliances:

FR-74: The PowerEye system shall allow the user to add one or more appliances from a list of the existing devices (i.e., those already added to the PowerEye system) to an existing room.

FR-75: The PowerEye system shall allow the user to remove appliances from an existing room.

Controlling a Room (On/Off):

FR-76: When a user chooses to turn on/off an entire room, the PowerEye system shall apply the selected on/off action to all appliances associated with that room.

FR-77: The PowerEye system shall update the status (on/off) of the selected appliances.

Energy Consumption Analysis:

Display Real-Time Power Usage per Appliance:

FR-78: The PowerEye system shall provide a real-time display of power usage for each individual appliance.

Display aggregated data per appliance:

Energy Consumption (kWh):

FR-79: The PowerEye system shall provide the ability to view daily energy consumption data for each individual appliance.

FR-80: The PowerEye system shall provide the ability to view weekly energy consumption data for each individual appliance.

FR-81: The PowerEye system shall provide the ability to view monthly energy consumption data for each individual appliance.

FR-82: The PowerEye system shall provide the ability to view yearly energy consumption data for each individual appliance.

Cost (SAR):

FR-83: The PowerEye system shall provide the ability to view daily cost data for each individual appliance.

FR-84: The PowerEye system shall provide the ability to view weekly cost data for each individual appliance.

FR-85: The PowerEye system shall provide the ability to view monthly cost data for each individual appliance.

FR-86: The PowerEye system shall provide the ability to view yearly cost data for each individual appliance.

CO2 Emissions (kg):

FR-87: The PowerEye system shall provide the ability to view daily CO2 emissions data for each individual appliance.

FR-88: The PowerEye system shall provide the ability to view weekly CO2 emissions data for each individual appliance.

FR-89: The PowerEye system shall provide the ability to view monthly CO2 emissions data for each individual appliance.

FR-90: The PowerEye system shall provide the ability to view yearly CO2 emissions data for each individual appliance.

Display aggregated data per room:

Energy Consumption (kWh):

FR-91: The PowerEye system shall provide the ability to view daily energy consumption data for each room, aggregated by the appliances within that room.

FR-92: The PowerEye system shall provide the ability to view weekly energy consumption data for each room, aggregated by the appliances within that room.

FR-93: The PowerEye system shall provide the ability to view monthly energy consumption data for each room, aggregated by the appliances within that room.

FR-94: The PowerEye system shall provide the ability to view yearly energy consumption data for each room, aggregated by the appliances within that room.

Cost (SAR):

FR-95: The PowerEye system shall provide the ability to view daily cost data for each room, aggregated by the appliances within that room.

FR-96: The PowerEye system shall provide the ability to view weekly cost data for each room, aggregated by the appliances within that room.

FR-97: The PowerEye system shall provide the ability to view monthly cost data for each room, aggregated by the appliances within that room.

FR-98: The PowerEye system shall provide the ability to view yearly cost data for each room, aggregated by the appliances within that room.

CO2 Emissions (kg):

FR-99: The PowerEye system shall provide the ability to view daily CO2 emissions data for each room, aggregated by the appliances within that room.

FR-100: The PowerEye system shall provide the ability to view weekly CO2 emissions data for each room, aggregated by the appliances within that room.

FR-101: The PowerEye system shall provide the ability to view monthly CO2 emissions data for each room, aggregated by the appliances within that room.

FR-102: The PowerEye system shall provide the ability to view yearly CO2 emissions data for each room, aggregated by the appliances within that room.

Display aggregated data for total (all appliances):

Energy Consumption (kWh):

FR-103: The PowerEye system shall provide the ability to view daily total energy consumption data for all appliances combined.

FR-104: The PowerEye system shall provide the ability to view weekly total energy consumption data for all appliances combined.

FR-105: The PowerEye system shall provide the ability to view monthly total energy consumption data for all appliances combined.

FR-106: The PowerEye system shall provide the ability to view yearly total energy consumption data for all appliances combined.

Cost (SAR):

FR-107: The PowerEye system shall provide the ability to view daily total cost data for all appliances combined.

FR-108: The PowerEye system shall provide the ability to view weekly total cost data for all appliances combined.

FR-109: The PowerEye system shall provide the ability to view monthly total cost data for all appliances combined.

FR-110: The PowerEye system shall provide the ability to view yearly total cost data for all appliances combined.

CO2 Emissions (kg):

FR-111: The PowerEye system shall provide the ability to view daily total CO2 emissions data for all appliances combined.

FR-112: The PowerEye system shall provide the ability to view weekly total CO2 emissions data for all appliances combined.

FR-113: The PowerEye system shall provide the ability to view monthly total CO2 emissions data for all appliances combined.

FR-114: The PowerEye system shall provide the ability to view yearly total CO2 emissions data for all appliances combined.

Notifications and Alerts:

Meross Credential Invalidation Notification:

FR-115: When a change or invalidation is detected in the user's Meross password, the PowerEye system shall notify the user to update their Meross password.

FR-116: The PowerEye system shall prompt the user to input their updated Meross password within the PowerEye app.

FR-117: The PowerEye system shall validate the updated Meross password to ensure it is associated with a valid Meross account.

Device Disconnection Notification:

FR-118: In the event of a device disconnection (i.e., being unplugged from electricity, or losing internet connectivity with the smart plug), the PowerEye system shall send a notification to the user, providing information about the disconnected device.

Near Goal Proximity Notification:

FR-119: When the total monthly energy consumption reaches 25%, 50%, or 75% of the user-defined monthly goal, the PowerEye system shall send a notification to the user. This notification will inform the user of their proximity to reaching the consumption goal.

Goal Reaching Notification:

FR-120: When the total monthly energy consumption reaches 100% of the user-defined monthly goal, the PowerEye system shall send a notification alert to the user, indicating that the consumption goal has been reached.

Goal Exceedance Notification:

FR-121: When the total monthly energy consumption reaches 125%, 150%, 175%, and 200% of the user-defined monthly goal, the PowerEye system shall send a real-time notification alert to the user, informing them of the exceeded consumption and providing details about the excess.

Recommendations for Optimal Energy Usage:

Recommendation Based on Peak Usage Times:

FR-122: The PowerEye system shall continuously monitor shiftable appliances' status to detect usage during KSA's peak energy demand period, which occurs between 13:00 and 17:00.

FR-123: If the PowerEye system identifies a shiftable appliance in use during KSA's peak energy demand period, the PowerEye system shall send a notification specifying the appliance in use and recommending deferring its usage until after the peak hours.

Recommendation Based on Phantom Mode:

FR-124: The PowerEye system shall detect appliances that are consuming electricity in phantom mode (standby mode).

FR-125: If the PowerEye system detects appliances in phantom mode, the PowerEye system shall send a notification specifying the appliances in phantom mode and recommending turning them off.

Recommendation Based on Energy Baseline:

FR-126: The PowerEye system shall compute an energy consumption baseline (average) for each appliance, establishing a reference point for typical energy usage for that appliance.

FR-127: If the energy consumption of an appliance exceeds its baseline, the PowerEye system shall send a notification specifying which appliances have exceeded their established energy baseline and recommending to reduce their usage.

User Guide and Support

FR-128: The system shall provide a user guide categorized into different topics, including:

- App navigation and menu options.
- Setting up user profiles.
- Device management.
- Reading energy consumption analysis.

- Energy optimization.
- Frequently asked questions (FAQs).
- Contact information

Non-Functional Requirements:

Usability Requirements:

NFR-1: The PowerEye system shall provide a user manual to facilitate the use of the system.

NFR-2: The PowerEye system shall prompt users to provide all needed information to be filled in, and the application shall display clear error messages specifying any missing fields.

Performance Requirements:

NFR-3: The PowerEye system shall be responsive and provide real-time energy consumption data for both iOS and Android platforms.

Reliability Requirements:

NFR-4: The PowerEye system shall have built-in error-handling mechanisms to handle unexpected situations gracefully and recover from failures quickly.

Integrity Requirements:

NFR-5: The PowerEye system shall support integration with the Meross cloud.

NFR-6: The PowerEye system shall protect against the unauthorized addition, deletion, or modification of the information of the users.

Security Requirements:

NFR-7: The PowerEye system shall require users to provide valid credentials (email and password) for authentication before gaining access to the app.

NFR-8: The PowerEye system shall employ token-based authentication protocols to ensure secure authentication for both users and devices.

NFR-9: The PowerEye system shall employ a data hashing technique to protect users' passwords.

NFR-10: The PowerEye system shall accept the password only according to the complexity criteria mentioned.

Compatibility Requirements:

NFR-11: The PowerEye system shall be compatible with both Android and iOS, ensuring that users can access and use the app seamlessly regardless of their device preferences.

Testability Requirements:

NFR-12: The PowerEye system shall pass a comprehensive set of test scenarios, with each requirement being associated with at least one test scenario, to ensure complete testing coverage.

Requirement Engineering (RE) Process:

First iteration:

Elicitation:

We initiated the process by conducting a thorough analysis of existing products in the market, gaining a comprehensive understanding of features, functionalities, and user experiences offered by similar Home Energy Monitoring Systems (HEMS). We followed this with in-depth interviews with 9 potential users, representing diverse technical backgrounds and experiences related to our system. A structured interview approach was developed, with prepared questions and follow-up questions to go deeper into specific areas. This technique allowed us to gather detailed and unbiased feedback on energy consumption tracking. The information gathered from these interviews provided input for defining the initial set of requirements.

Analysis and Specification:

After compiling an extensive list of features suggested by our 9 interviewees, we carefully evaluated each one. We considered factors like technical feasibility, available resources, and our team's expertise. We recognized that a few features may require resources or expertise beyond our current scope. These were earmarked for future consideration. This approach ensured that the final list of features was both user-driven and realistically achievable within our project's constraints. We utilized Data Flow Diagrams (DFDs) to understand the flow of data within the system and identify dependencies. As well as a Features tree to understand the main categories of features the system should have.

Prioritization:

In order to focus on the most critical functionalities, we conducted a thorough tally of how frequently each feature was mentioned during our interviews. We used the three-level scale technique (high: mentioned 5 or above, medium: mentioned 3-4, and low: mentioned 1-2). Features that were mentioned five times or more were given precedence, ensuring that the final product would address the most prevalent user needs. This methodical approach allowed us to prioritize features based on their frequency of mention, ensuring that the resulting product would be tailored to meet the most common and crucial requirements identified by our user base.

Verification and Validation:

After writing the initial set of functional and non-functional requirements derived from the prioritized features list, we validated them by reviewing and inspecting whereas all the team members read them carefully and ensured that the requirements were SMART and free of any errors or inconsistencies in order to ensure delivering a high-quality product that satisfies the user's needs and expectations.

Second Iteration:

Elicitation:

We conducted a follow-up survey with the same audience of 9 participants who were interviewed in the first iteration. This survey aimed to delve deeper into the gathered features, seeking more specific details and preferences to ensure a comprehensive and user-centric approach in the development of our Home Energy Monitoring System (HEMS) app.

Analysis and Specification:

We analyzed the survey responses to pinpoint specific requirements and user preferences. We employed various modeling techniques to refine and specify our requirements. This included representing them as use case scenarios in tabular form for a detailed view of system functions. A Holistic Use Case Diagram provided an overarching perspective of system interactions. Additionally, for specific features, we used models like Sequence Diagrams and Activity Diagrams to gain deeper insights into their behavior and interactions.

Prioritization:

In order to prioritize the list of features, we conducted a survey with a sample of 68 participants. The survey utilized a three-level scale - 'Very Important', 'Important', and 'Not

Important' - to evaluate each requirement. This meticulous approach allowed us to thoroughly assess the importance of each requirement in accordance with user needs and expectations.

Very important: Important and urgent to implement this requirement (The application can't work without it).

Important: Important but not urgent to implement this requirement (The application can work without it).

Not important: Not important and not urgent to implement this requirement (The application can wait for a later release).

Verification and Validation:

During the validation process, we confirmed that PowerEye effectively met the user's actual needs and expectations. Verification, on the other hand, ensured that PowerEye adhered to the specified requirements. The analysis models presented to our targeted users were designed to illustrate the system's flow and components. Their evaluations and comments were closely observed and meticulously assessed in relation to their identified needs.

More information about the detailed requirements can be found in the original Software Requirements Specification (SRS) .

DESIGN

Five levels of architectural design: description of processes, components, subsystems systems

Software Architecture:

Architectural Style:

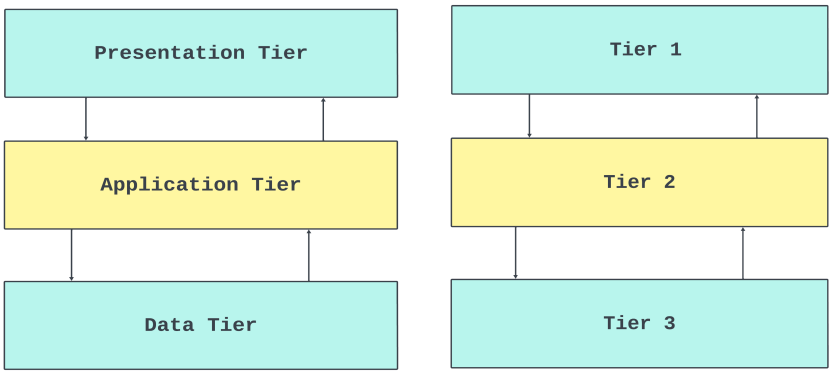
The PowerEye System will be constructed using the Three-tier client-server architecture, enhanced by incorporating the MVC pattern within the logic tier of this architectural framework.

Rationale/Justification:

The adoption of a Three-tier Architecture, complemented by the incorporation of the MVC pattern in the logic tier, aptly aligns with the specified Quality Attributes outlined in the project's requirements. This architectural choice provides a robust framework that addresses key aspects such as Security, Performance, Reliability, Compatibility, Testability, Usability, and Integrity. Each attribute is meticulously considered and integrated into the system, ensuring a well-rounded and effective solution.

A Simple Box and Line Diagram for 3-tier Architecture:

Figure 2: *3-tier Architecture*



Tiers of PowerEye Application:

- **Presentation Tier:** the top layer is responsible for the user interface and experience. It manages how users interact with the app, presenting information in a user-friendly manner and ensuring easy navigation.
- **Application Tier:** the middle layer that acts as the application's "brain". It handles core functionalities like user authentication, energy consumption analysis, and recommendations. Ensuring the app functions correctly and efficiently.

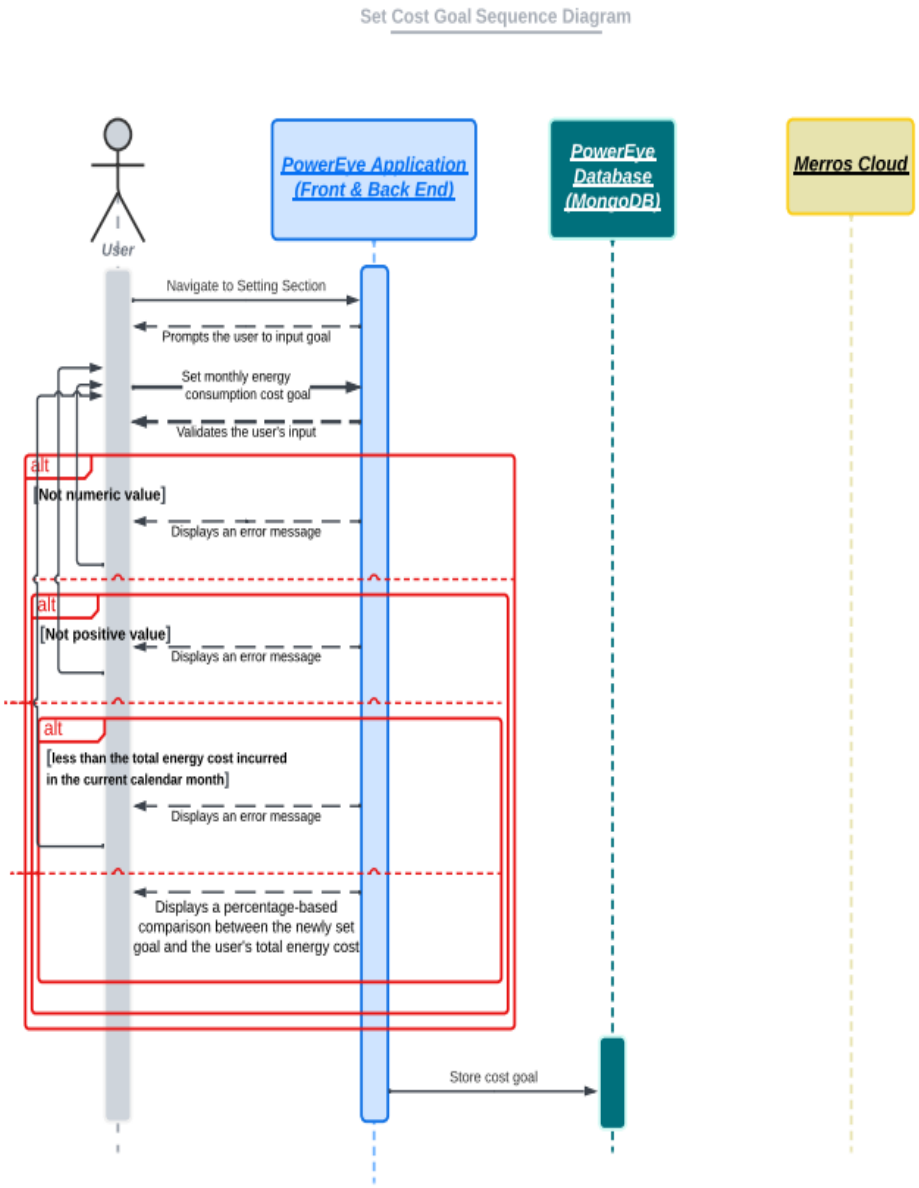
- **Data Tier:** the bottom layer is responsible for storing and managing the application's data. It maintains data integrity, security, and accessibility, allowing the app to retrieve and present data to users through the other tiers.

4+1 Views Model:

Logical View:

The logical view focuses on the functional and structural organization of the software components within the system. It describes the system from a software perspective, emphasizing the modular decomposition, relationships between components, and their interactions. The logical view provides a high-level understanding of the system's functionality and how it is structured by using static or dynamic diagrams. Since our application follows a classless design paradigm, which means we do not adhere to traditional object-oriented programming concepts we won't be able to draw the class diagram however we draw the schema diagram which will illustrate a collection structure that closely resembles the class diagram. Here's an example of a sequence diagram. For more diagrams, please refer to the original Software Design Document (SDD).

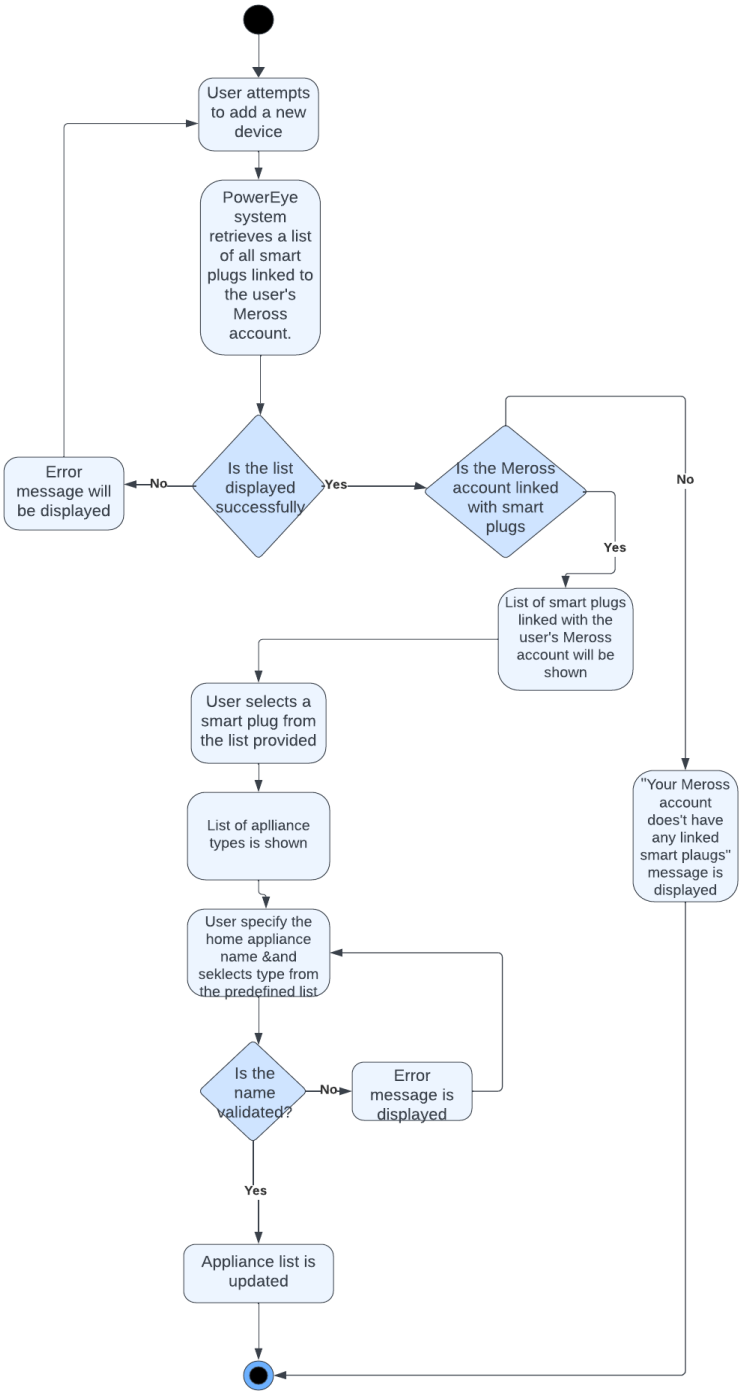
Figure 3: *Set Cost Goal Sequence Diagram*



Process View:

The process view focuses on the dynamic aspects of the system, emphasizing the runtime behavior, concurrency, and interactions between software components. It provides insights into how the system executes and handles processes, threads, and concurrent activities. Here's an example of an activity diagram. For more diagrams, please refer to the original Software Design Document (SDD).

Figure 4: *Adding New Appliance Activity Diagram*

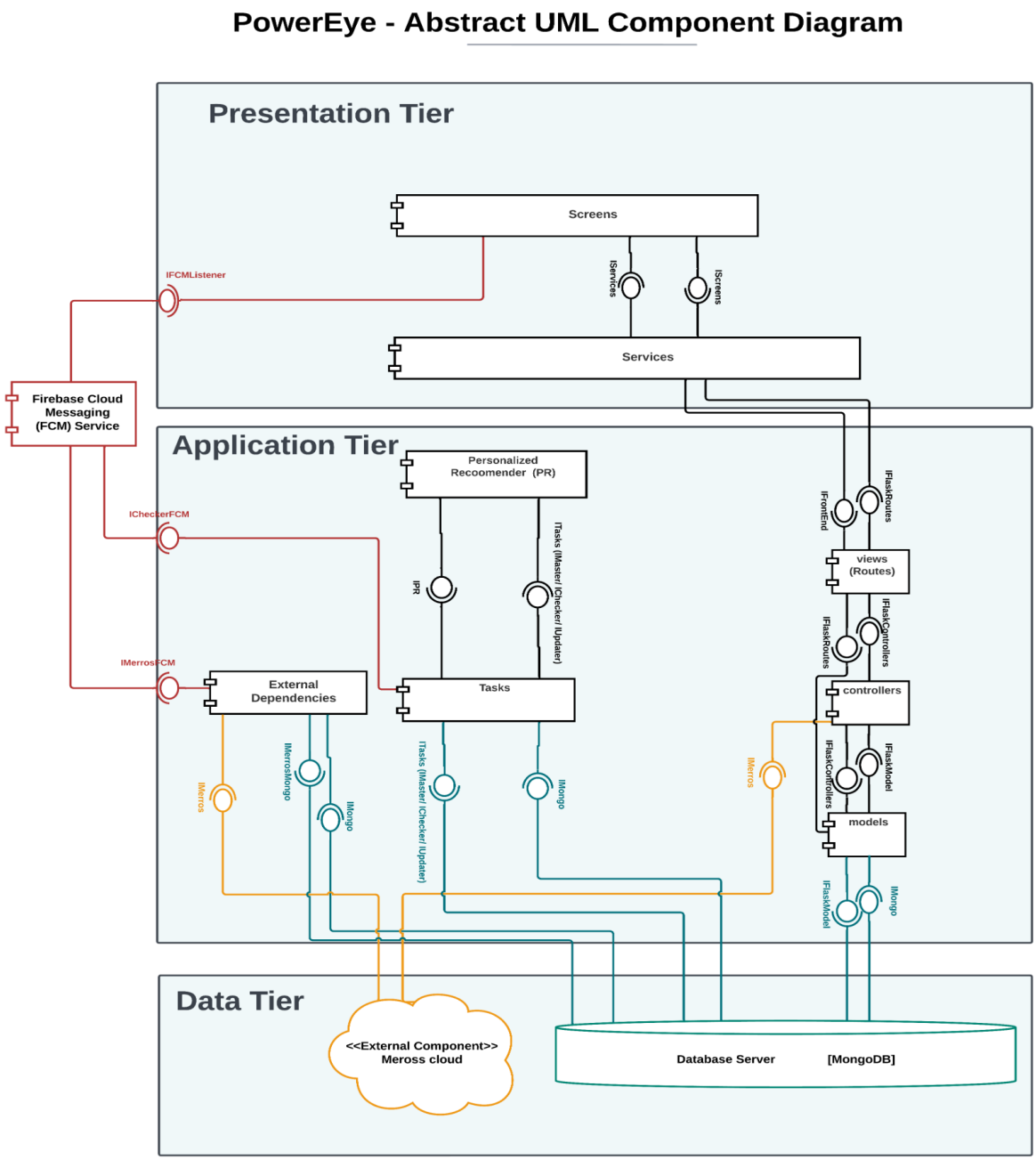


Development View:

The development/implementation view illustrates a system from a programmer's perspective. It shows how the software components are developed, built, and integrated to create

the overall system. The following is the Component Diagram based on the Three-tier architectural style of PowerEye:

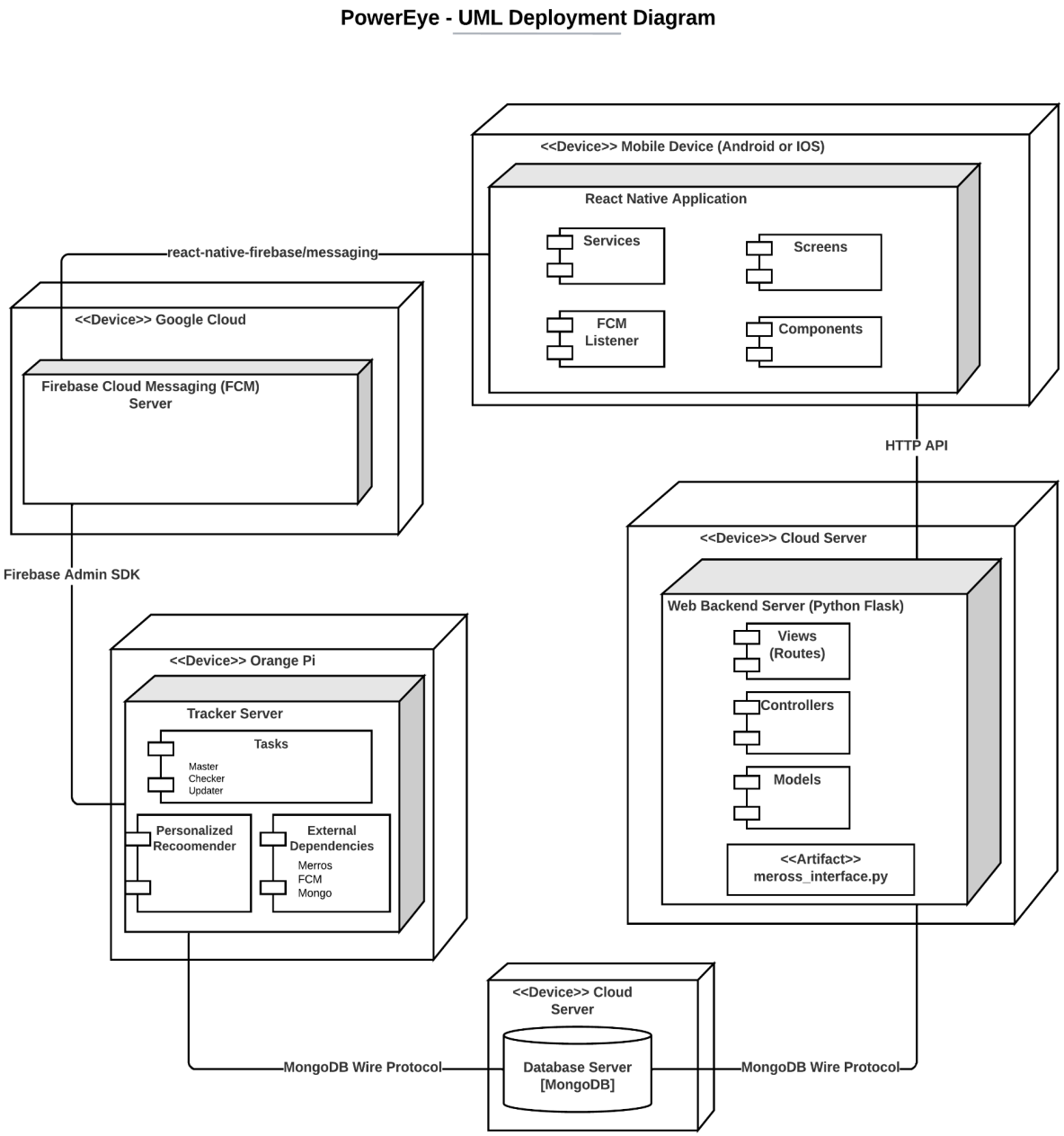
Figure 5: *Abstract Component Diagram for PowerEye*



Physical View:

The physical/deployment view shows the system from a system engineer's point of view. It describes the physical distribution of system components across different hardware, including servers, processors, network infrastructure, and other physical resources as well as the physical connections between these components. The following is the Deployment Diagram of PowerEye:

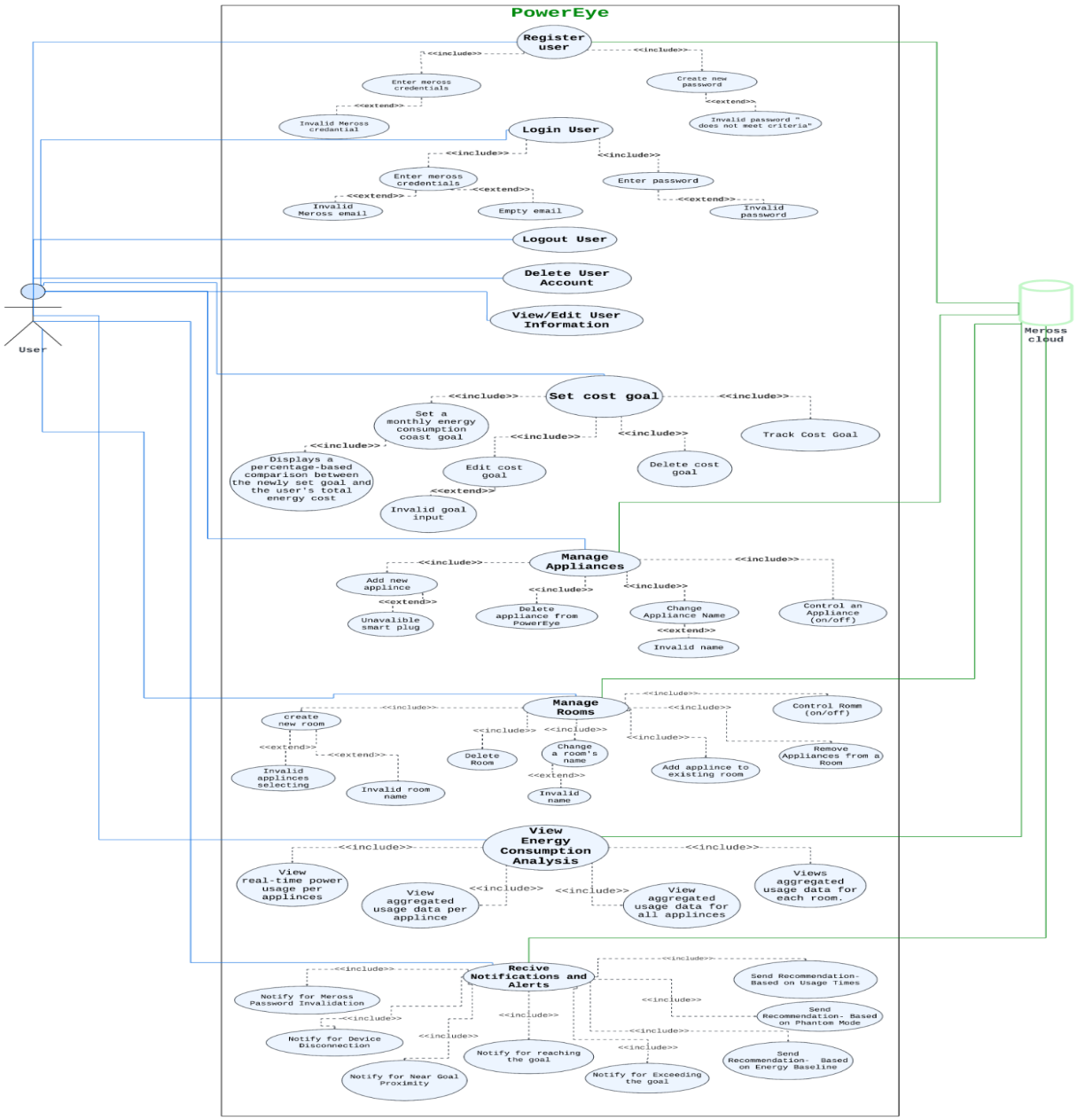
Figure 6: *Deployment Diagram for PowerEye*



User View:

The user view represents the system from the perspective of its users and focuses on the user interface and interaction with the system. It describes how users will interact with the system's features and functionalities to achieve their goals. The following is the Use Case diagram of PowerEye:

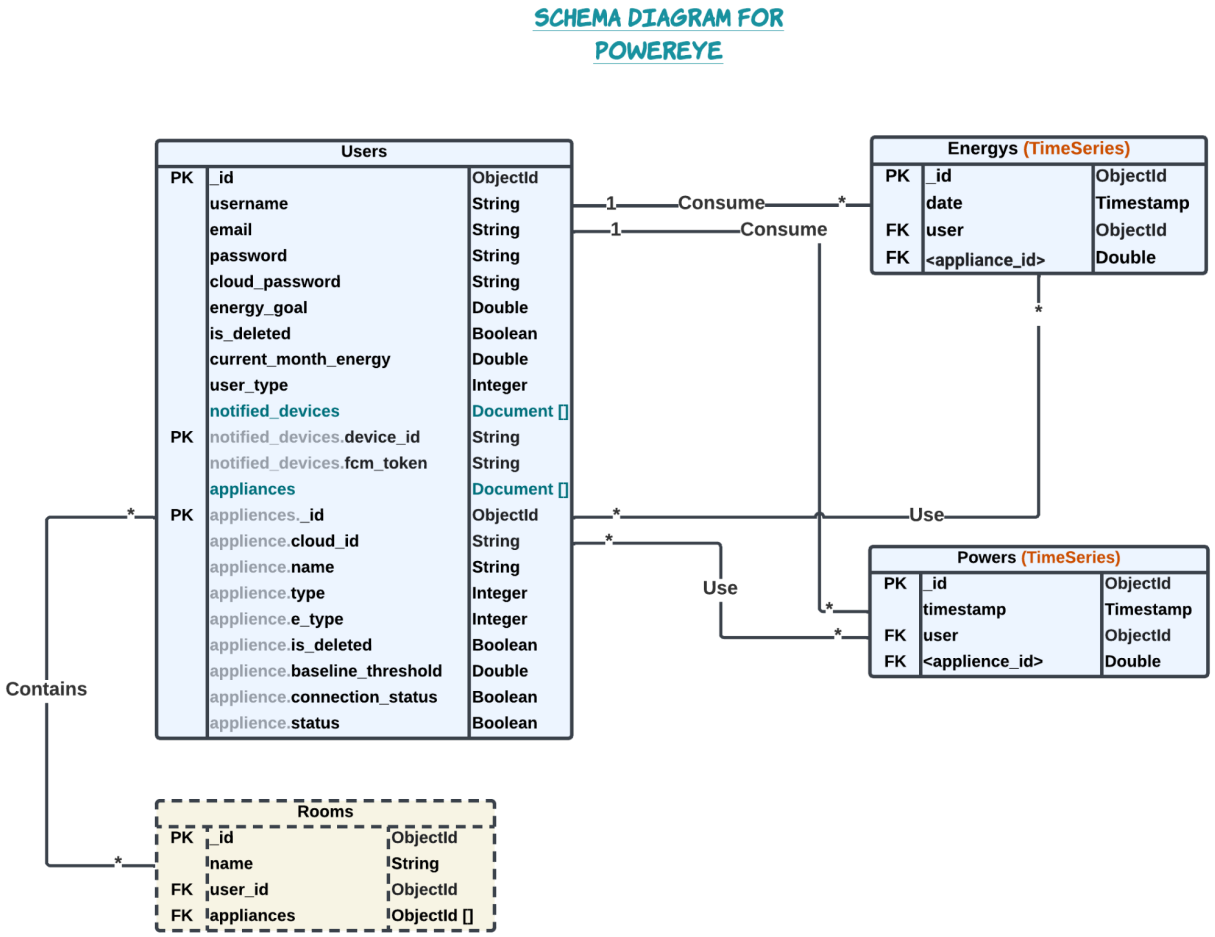
Figure 7: *Use Case Diagram for PowerEye*



Database:

Database Schema:

Figure 8: *Schema Diagram for PowerEye*



Database Schema Explanation:

The PowerEye system has four collections: "Users", "Energys", "Powers", and "Rooms". Energys and Powers are going to be a TimeSeries Collections which means efficiently storing sequences of measurements over a period of time. Each collection is represented as an entity in the Schema Diagram, and each entity has fields & keys.

Here's a breakdown of the fields within each collection:

Users Collection:

- **_id:** Unique identifier for each user.
- **username:** User's username.
- **email:** User's email address.
- **password:** Encrypted user's password.
- **cloud_password:** password for Plug Cloud (not encrypted; it is required for cloud APIs).
- **energy_goal:** User's monthly energy consumption goal.
- **is_deleted:** Flag indicating if the user is deleted or active.
- **user_type:** User type for Plug Cloud integration (1 = Meross, 2 = Tuya).
- **current_month_energy:** User's energy consumption for the current month. It is used for tracking progress towards **energy_goal**.

Notified_devices: that will be embedded with the Users:

- **device_id:** Unique identifier for each user device.
- **fcm_token:** token for user registration in the Firebase for notifications.

Appliances Collection: that will be embedded with the Users:

- **_id:** Unique identifier for each appliance.
- **cloud_id:** device ID for Plug Cloud integration.
- **name:** Name of the appliance.
- **type:** Type of the appliance. The type that is shown to the user.
- **e_type:** Energy type of the appliance (1= none, 2 = shiftable, 3 = shiftable and phantom).
- **is_deleted:** Flag indicating if the appliance is deleted or active. It is used because the appliance's power and energy data cannot be removed[2] even when the appliance is no longer in use.
- **baseline_threshold:** Threshold value for daily baseline energy consumption.
- **connection_status:** Status of the appliance's internet connection.
- **status:** On/Off status.
- **energy:** appliance energy consumption for the current day.

Energys (TimeSeries Collection) (Daily):

- **_id:** Unique identifier for each energy data entry.

[2] TimeSeries Collections in MongoDB don't allow delete operations.

- **<appliance_id>[3]:** Dynamic field where its key references the appliance id and its value contains the energy of the day specified in the date field.
- **date:** Timestamp of the energy measurement.
- **user:** Reference to the user who owns the appliance.

Powers (TimeSeries Collection) (1 min):

- **_id:** Unique identifier for each energy data entry.
- **<appliance_id>:** Dynamic field where its key references the appliance id and its value contains the energy of the minute specified in the timestamp field.
- **timestamp:** Timestamp of the power measurement.
- **user:** Reference to the user who owns the appliance.

Rooms Collection: that's going to be a weak entity depending on the existence of appliances

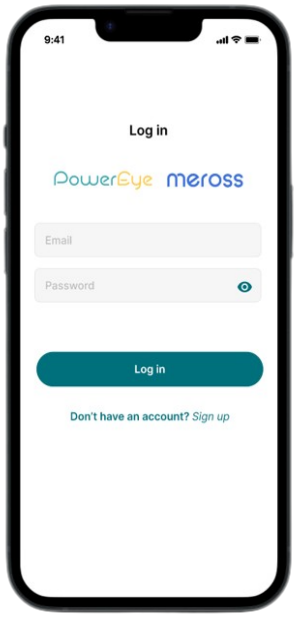
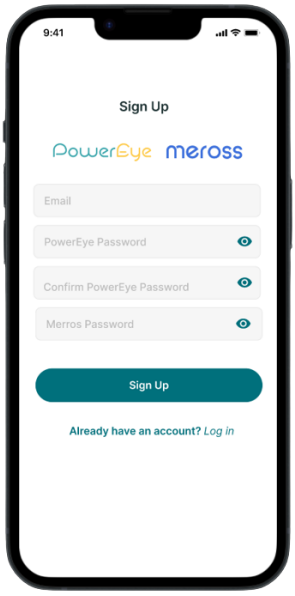
- **_id:** Unique identifier for each energy data entry.
- **name:** Name of the room.
- **user_id:** Reference to the user who owns the room.
- **appliances:** List/array of appliance IDs.

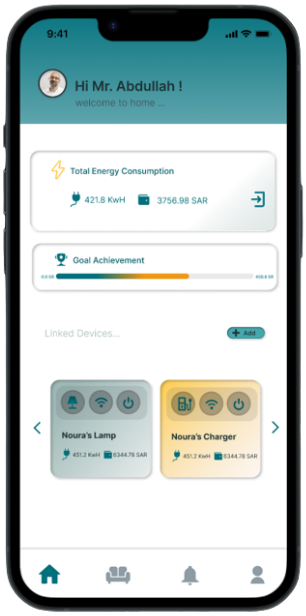
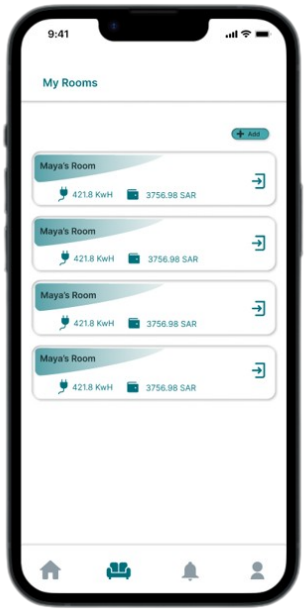
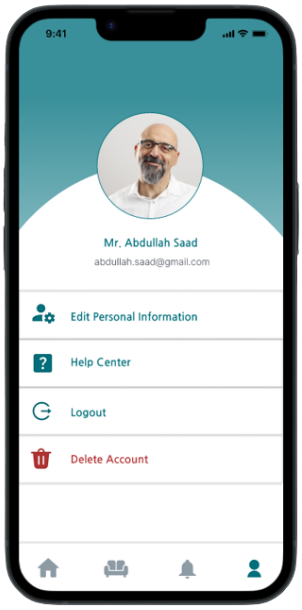
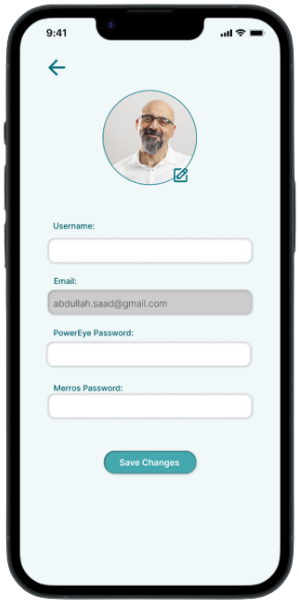
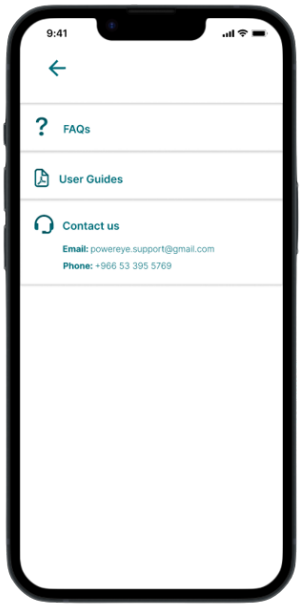
[3] This field is dynamic and depends on how many appliances the user has at this timestamp. This allows for flexible association of energy and power data with the corresponding appliance.

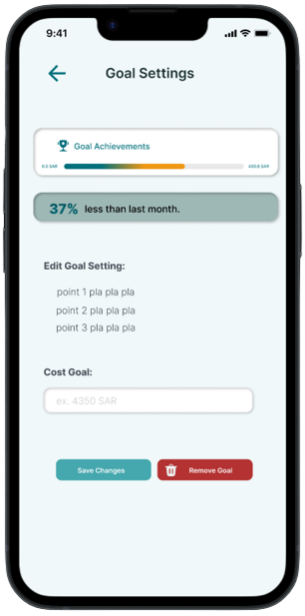
User Interface:

High-fidelity prototype.

Table 7: *Interfaces*

Starting Page	Login Page	Signup Page
		
Home Page	**My Rooms Page**	**Notification Page**

		
Profile Pge	**Edit Profile Info Page**	**Help Center Page**
		
Setting Goal Page	**Total Page**	**Appliance Page**



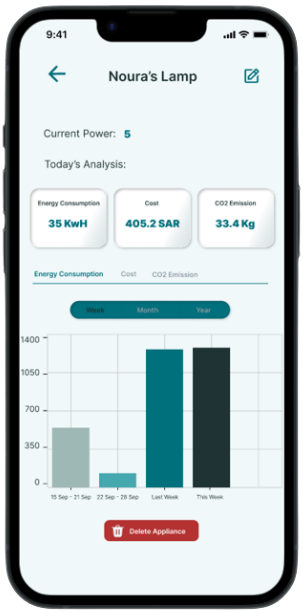
Are you sure you want to remove your Goal?

Once you remove this goal, it can't be restored. You will lose access to to past data and analysis.

Remove | Cancel

Add Appliance 1 Page

Add Appliance 2 Page

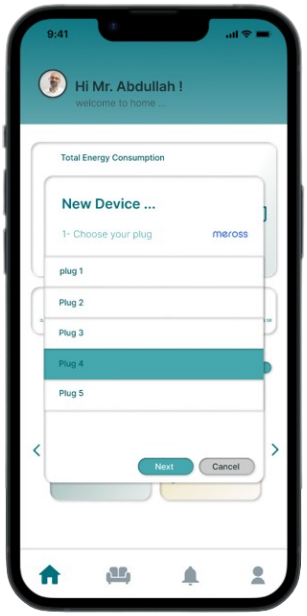


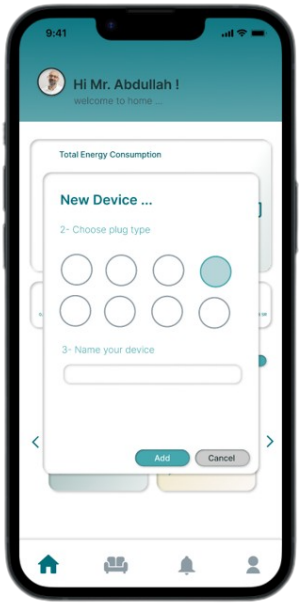
Are you sure you want to permanently delete [Device Name]?

Once you delete this device, it can't be restored. You will lose access to to past data and analysis.

Delete | Cancel

Add Appliance 3



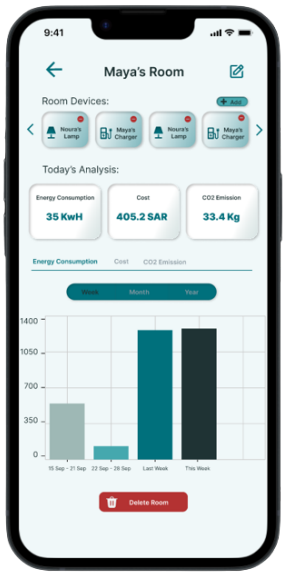
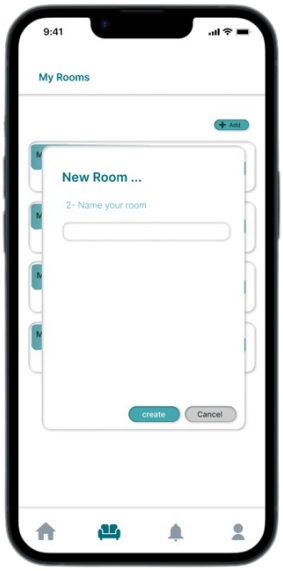
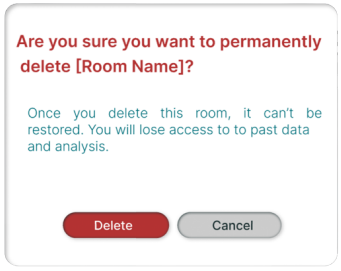
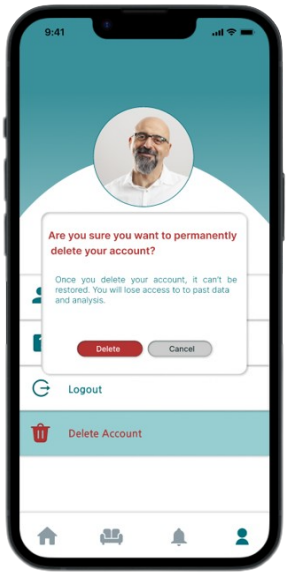
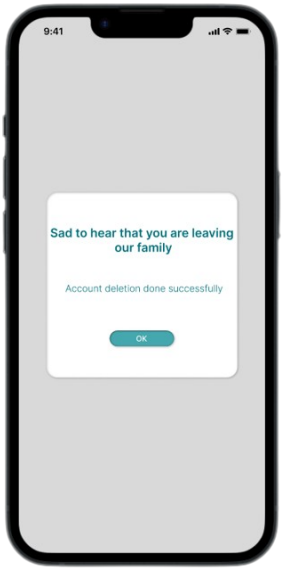


Unable to retrieve plugs

Retry | Cancel

No smart plug found !

Ok

Room Page	Add Room 1 Page	Add Room 2 Page
		
	**Delete Account Page**	**Account Deleted Page**
		

Software Design Process:

First Iteration:

The first iteration of the software design process involves the crucial step of identifying the architectural style. For our project, we chose a three-tier pattern. The three-tier pattern is a design approach that separates a software system into three distinct layers: the presentation layer, the application layer, and the data layer. Also, we utilized the 4+1 Views Model for architectural viewing. It provides multiple perspectives or "views" of the system, ensuring a holistic understanding of its structure and behavior. The four views in this model include the logical view we developed the Sequence Diagram_to illustrate the interaction and flow of messages between different components or objects. The process view utilized the Activity Diagram that was created during the requirement phase to understand the system's dynamic behavior and the sequence of activities. For the development view, we created a less detailed Component Diagram that showcased the individual components and their relationships, aiding in understanding the system's modular structure. The physical view was represented through the Deployment Diagram, which illustrated the physical deployment of components on hardware or computing resources. Lastly, the user view was captured through the Use Case Diagram, which identified the various use cases and the interactions between actors and the system

Second Iteration:

In the second iteration of the software design process, we initially planned to proceed with the Detailed Design phase. However, Our application follows a classless design paradigm. Therefore we embarked on a more detailed version of the Component Diagram. This detailed version of the component diagram allowed us to go deeper into the system's architecture and understand the interactions between various components. We identified the specific functionalities and responsibilities of each component, including their input and output interfaces.

More information about the detailed design can be found in the original Software Design Document (SDD).

IMPLEMENTATION

Implementation details over 2 iterations, software/tools and algorithms

Introduction:

PowerEye, a complex project, relies on a meticulously structured and modular implementation to facilitate effective control and monitoring of smart plugs and energy consumption. To simplify the understanding of this intricate system, we adopt an approach that illustrates how each component is implemented. In this document, we provide insights into the implementation process for each part of the system.

Initially, the system is segmented into modular components. To depict this, we create a straightforward component diagram, delineating the system components, their roles, and their intercommunication. Subsequently, each component is extensively expounded upon, accompanied by a more detailed component diagram that focuses on the specific component.

However, the Tracker Server operates on Object-Oriented Principles. For this server, we employ a class diagram to provide a comprehensive portrayal of its operations and the relationships between the classes.

For codebase illustration, we employ **tree-cli**, a robust tool adept at generating detailed tree-like project directory structures. We utilize its output, complementing it with comments that succinctly explain the role of each file and directory, providing a clear map of the codebase's layout and functionality.

To further elucidate the server's inner workings, flowcharts provide a valuable insight into the program flow and the core functionalities and logic of the Tracker Server. These flowcharts help explain how the system's tasks are executed and how the Energy Personalized Recommender (EPR) operates, ensuring transparency and emphasizing a user-centric approach to energy optimization.

Furthermore, the inclusion of algorithms, data structures, mathematical formulations, and pseudocodes offers a deeper insight into the core logic of the EPR module, demonstrating how the system deals with different events for the user; These elements are fundamental for implementing and maintaining the system's personalized energy optimization features and are vital for developers and stakeholders in comprehending the technical intricacies that make PowerEye a powerful and user-focused system.

Development Environment:

Developments Tools:

Table 8: *Used Tools*

Tool	Description	Used in
VS Code Editor	A highly customizable and popular source code editor developed by Microsoft. It supports multiple programming languages and offers features like syntax highlighting, debugging, extensions, and version control.	Web Server, Tracker Server, Mobile Application
VS Code Debugger	A powerful tool integrated directly into the VS Code editor. It helps developers find and fix issues in their code efficiently.	Web Server, Tracker Server
Git	A distributed version control system (DVCS) that enables tracking changes in source code. It supports collaboration and code history, enhancing software development.	Web Server, Tracker Server, Mobile Application
Pip Package Installer	A command-line tool used for installing and managing Python packages. It is used to install, manage, and upgrade Python packages and libraries from the Python Package Index (PyPI).	Web Server, Tracker Server
Node Package Manager (NPM)	A package manager for the JavaScript programming language and is widely used in the Node.js environment. NPM allows developers to easily manage and install third-party libraries (packages or modules) .	Mobile Application
Postman	A popular collaboration platform for API development. It provides tools for designing, testing and managing APIs. So, it provides a user-friendly interface for making HTTP requests.	Web Server
Jupyter Notebook	An open-source web application that allows you to create and share documents that contain live code, equations, visualizations, and narrative text. It's commonly used for data analysis, and interactive coding with languages like Python, R, and Julia.	Tracker Server
Spyder	An open-source IDE primarily designed for scientific computing and data analysis in Python. It provides features like an interactive console, variable explorer, and code editor.	Tracker Server

Firebase SDK	The Firebase SDK is a set of development tools and libraries from Google that enables easy integration of Firebase services into mobile, web, and game applications. It simplifies tasks like user authentication, real-time data, cloud storage, and analytics.	Mobile applications, Tracker Server
PyTest	Pytest is a widely-used, open-source testing framework for Python. It simplifies the process of writing and executing tests and offers a range of features and plugins for various testing needs. Pytest is known for its simplicity, scalability, and extensibility.	Web Server, Tracker Server (Unit, Integration, Functional, End-to-End Testing)
Expo Go	Expo Go is a free, open-source sandbox for learning and experimenting. It works as an emulator for React Native code for both IOS and Android devices.	Mobile Application

Programming Languages, & Software Frameworks:

Table 9: *Used Languages/Frameworks*

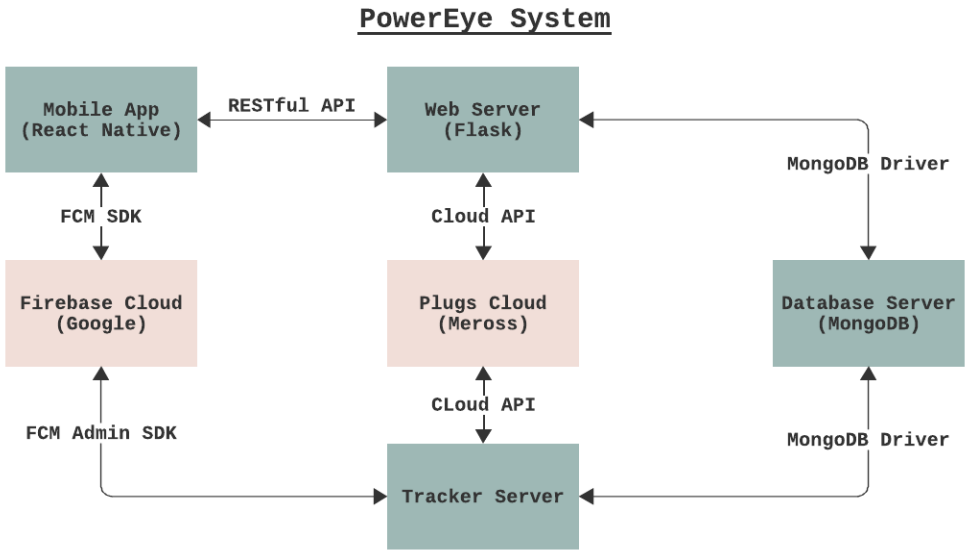
Language/Framework	Description	Used in
Python	A high-level, versatile, and widely-used programming language known for its readability, extensive libraries, and simplicity.	Web Server, Tracker Server
Flask	A micro web framework for Python. It is designed to be lightweight and minimalistic, providing the essentials for building web applications without imposing a lot of structure or components.	Web Server
Apscheduler[4]	A Python library used for task scheduling. It allows you to schedule functions or jobs to run at specific times or intervals. APScheduler is often used for automating recurring tasks and is a valuable tool for managing time-based operations in Python applications.	Tracker Server
React Native	An open-source framework of JavaScript and React framework used for building mobile applications. It is used for developing native mobile apps for both Android and IOS using a single codebase.	Mobile Application

[4] Though it's not a framework, it's worth mentioning here for its significant role in the Tracker Server.

System Architecture & Components:

System High-level Architecture:

Figure 9: *System Component Diagram*



The PowerEye system is broken down into the following components:

Database Server:

MongoDB serves as the central repository for storing and managing various types of data, including user profiles, appliance and room information, as well as power and energy consumption records. Both servers seamlessly interact with MongoDB through the MongoDB driver. MongoDB's flexibility and scalability are well-suited for handling unstructured and semi-structured data, making it an excellent choice for storing appliance-related information (status, power, etc). It enables efficient querying and retrieval of data for various application functionalities.

Plugs Cloud (Meross):

The Plugs Cloud component acts as a bridge between the application and Meross smart plugs. It enables communication through the API provided by the Meross cloud, allowing retrieving real-time information, and enabling remote control. Using the Meross Cloud ensures seamless integration with Meross smart plugs, allowing users to monitor and control their devices through our application “PowerEye”. This real-time interaction enhances user experience and functionality.

Web Server:

The web server component serves as the web server responsible for hosting the user interface, handling HTTP requests via RESTful APIs, user authentication, and serving as the primary point of interaction between users and the application using the Flask framework. Flask's lightweight and Pythonic nature make it a suitable choice for building web-based interfaces. Its simplicity and extensibility enable developers to create responsive user interfaces and manage user interactions effectively.

Tracker Server:

The Tracker Server is the backbone of the system, managing power data from Meross Cloud, storing device details in MongoDB, tracking events like energy goals, and delivering personalized recommendations and push notifications through the FCM Admin SDK. It centralizes backend operations, processes Meross device data, and offers valuable insights. Its event monitoring, personalized recommender, and push notification system boost user engagement and satisfaction.

Firebase Cloud:

Firebase Cloud Messaging (FCM) is responsible for sending push notifications to mobile devices. It acts as a communication bridge between the Tracker server and the mobile application. Notifications are delivered to mobile apps via the FCM SDK. FCM simplifies the process of sending real-time notifications to mobile users. It ensures that users are promptly informed of critical events, such as reaching energy goals or receiving recommendations. This enhances user engagement and keeps users informed about their smart plug usage.

Mobile Application:

The React Native-based mobile app offers a platform-independent interface for Meross smart plug control. Users can view device status, consumption data, recommendations, and control devices remotely. React Native streamlines cross-platform app development with a single codebase, saving time and ensuring a consistent experience on Android and iOS.

By breaking down the system into these components, a robust and modular architecture is created. Each component focuses on its specific responsibilities, promoting maintainability, scalability, and the ability to make updates or improvements without disrupting the entire system. This architecture ensures efficient communication, data storage, user interaction, and real-time notifications; enhancing the overall user experience.

Database:

The database consists of four collections:

Users: Stores user information, including an embedded **Appliances** document that contains details about the user's appliances and another embedded document, **Notified_devices**, for Firebase tokens.

Rooms: Stores a list of appliance IDs.

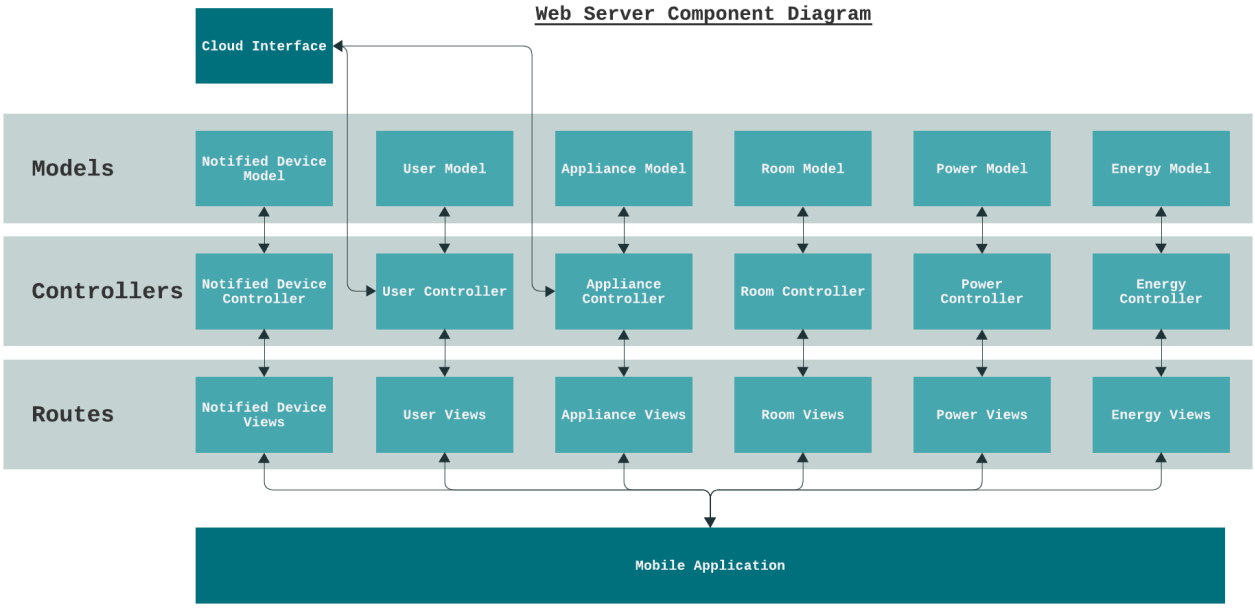
Powers: Time series collection that stores power readings with minutely intervals.

Energys: Time series collection that stores energy readings at daily intervals.

For more detailed information about the schema and these collections, please refer to the Database Design Section.

Web Server:

Figure 10: *Web Server Class Diagram*



The PowerEye Web Server adheres to Three-tier client-server architecture, enhanced by incorporating the Model-View-Controller (MVC) pattern within the logic tier, emphasizing RESTful APIs. In this design, views are relocated to the front-end (mobile application), establishing communication with the web server through RESTful APIs. This approach employs six primary components to oversee different facets of the system.

- **User Component:**

  - Model: Manages user-related data, such as profiles, preferences, and authentication.

  - Controller: Handles user-related logic, and interactions between the model and views.

- Views (Routes): Includes routes for user registration, login, profile settings, and other user-related activities.

- **Appliance Component:**
  - Model: Stores information about appliances, their configurations, and energy usage.
  - Controller: Manages appliance-related logic, configurations, and interactions with the model and views.
  - Views (Routes): Contains routes for adding, controlling, and monitoring appliances.

- **Notified Device Component:**
  - Model: Stores information about user devices and Firebase tokens.
  - Controller: Manages interactions with user device data, and coordinates with the model and views.
  - Views (Routes): Includes route for posting the FCM tokens along with the device data.

- **Room Component:**
  - Model: Represents rooms in a household, storing data about the room and connected appliances.
  - Controller: Handles room-related logic, configurations, and interactions with the model and views.
  - Views (Routes): Includes routes for creating, editing, and viewing room details.

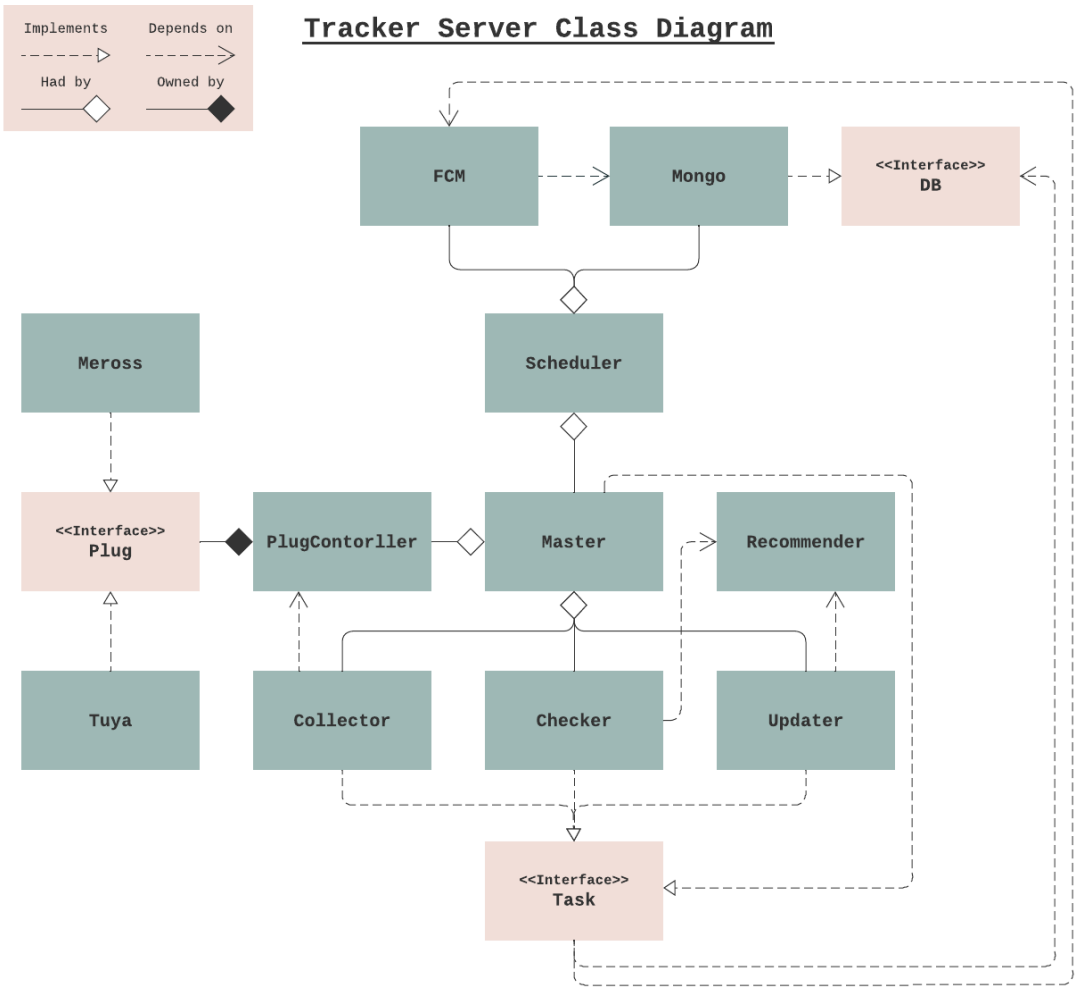
- **Energy Component:**
  - Model: Manages data related to energy consumption, and historical energy records.
  - Controller: Handles energy-related logic, and interactions with the model and views.
  - Views (Routes): Contains routes for viewing energy usage, and accessing historical data.

- **Power Component:**
  - Model: Represents real-time power data from household appliances.
  - Controller: Manages interactions with the real-time power data,, and coordinates with the model and views.
  - Views (Routes): Includes routes for real-time power monitoring and analysis.

Both the User and Appliance components interact with the cloud interface to retrieve data from and control smart plugs.

Tracker Server:

Figure 11: *Tracker Server Class Diagram*



The Tracker Server is composed of 3 interfaces and 11 classes, all of which are shown in Figure 11. The core of the server, where its execution begins, is the **Scheduler** class. Within the Scheduler, APScheduler is used to handle task scheduling by creating an instance of the BlockingScheduler. APScheduler manages multithreading, enabling each task to run on a separate thread. The thread pool is sized according to the number of CPU cores available on the hosting machine.

The **Master** task is instantiated within the Scheduler class and receives dependencies injected from the **FCM** (Firebase Cloud Messaging) and **Mongo** classes. These classes serve as interfaces to communicate with the Firebase Cloud and the MongoDB server, respectively. **FCM** focuses on pushing notifications to the Google Cloud Messaging Firebase, while the **Mongo** class implements the **DB** interface. This interface offers flexibility for potential future database substitutions. Within the **Mongo** class, the functions are designed to align with the specific

requirements of the Tracker Server. The **Master** task is also injected by the Scheduler and scheduled to run at one-minute intervals. Its primary purpose is to manage user additions and deletions, consequently launching new tasks for users and terminating existing tasks as needed.

In addition to the **Master** task, the server comprises three main tasks: **Collector**, **Checker**, and **Updater,** all of which implement the **Task** interface. The **Collector** task is responsible for gathering data from smart plug devices, including power consumption, on/off status, and connection status. Furthermore, it calculates daily energy consumption for each device and notifies users of any disconnections. This task operates on a one-minute schedule and relies on an additional dependency, the **PlugController**. The **PlugController** manages interactions with the smart plug cloud. It creates a plug instance based on the user's cloud_type and provides the necessary functionality for cloud interactions. Additionally, it accommodates both synchronous and asynchronous interactions.

Although PowerEye primarily is designed to work with **Meross** smart plugs, the system can easily expand to support other brands. To supplement our data collection within budget and time constraints, we've integrated the **Tuya** cloud as well. For more details about plug cloud integration, please refer to the external integration section. Both Tuya and Meross implement the **Plug** interface.

The **Checker** task is executed every minute. Its primary responsibility is to track events related to the **Energy Personalized Recommender (EPR)**, which includes monitoring energy goals, devices phantom mode and more. Further details on this E**PR** are available in the Personalized Recommender section.

The **Updater** task runs daily. It is responsible for updating current month energy consumption data, transferring daily energy consumption to the Energies collection for historical records, and managing the training of machine learning model clusters and forecasting models.

In brief, the Tracker Server is driven by the Scheduler class, effectively managing tasks. The Master task, with FCM and Mongo dependencies, handles user management. Three core tasks: Collector, Checker, and Updater, gather data, monitor events, and update energy consumption. The system is adaptable, supporting multiple plug brands and cloud platforms, providing versatility and reliability to users.

Firebase Cloud Messaging (FCM):

Definition:

Firebase Cloud Messaging (FCM) serves as a cornerstone of the PowerEye system, facilitating essential real-time, bidirectional communication between the PowerEye server and the PowerEye mobile application installed on users' smartphones. This dynamic component plays a

pivotal role in delivering instantaneous push notifications to mobile devices, actively involving users in the management of their smart plug usage. As a free mobile notification service provided by Google, FCM empowers app developers with the capability to dispatch notifications from Google Cloud Messaging (GCM) servers to their users, thus ensuring uninterrupted and seamless real-time communication between application servers and mobile devices. In this document, we delve into the workings and applications of push notifications within the PowerEye system, with a specific focus on how FCM enhances user engagement and interaction.

How Push Notifications Work and Their Types Done in the Project:

- **In-App Notifications:** Users receive notifications while actively using the PowerEye mobile application. These notifications keep users informed about real-time events and changes, enhancing their ability to manage smart plug usage.
- **Background Notifications:** Even when the PowerEye mobile app is running in the background, users receive notifications, which are displayed as system-level notifications on their devices. These background notifications ensure users stay updated, even if the app isn't in the foreground.
- **Push Notifications:** These notifications are sent to users' devices, even if the PowerEye app is not actively running. Firebase Cloud Messaging (FCM) ensures that push notifications are delivered promptly, helping users receive timely alerts and reminders.

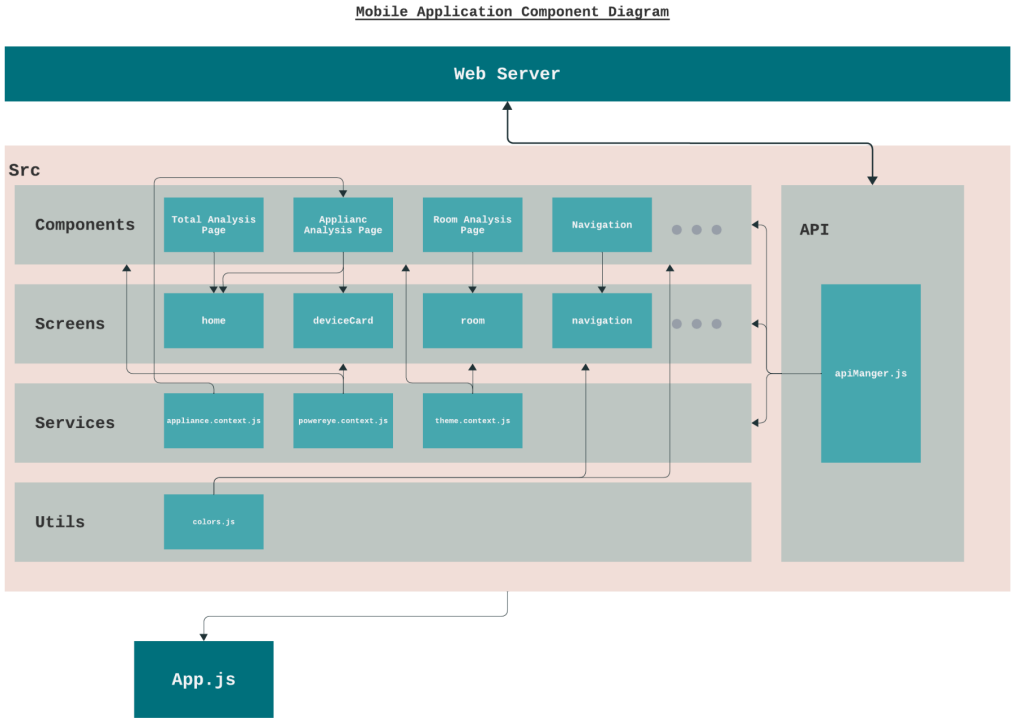
Application in Our Project:

- **Notification Trigger Events:** In the PowerEye system, various events trigger notifications, such as reaching energy consumption goals, peak time reminders, phantom mode alerts, or exceeding energy baselines. The Tracker Server identifies these events based on user settings and appliance data, and further discussion on this topic will follow.
- **Event Detection:** The Tracker Server continuously monitors and collects data from the smart plugs and tracks user-defined events. For example, if a user has set a specific energy consumption goal, the system monitors their energy usage and triggers an event when the goal is achieved or exceeded.
- **Notification Content:** When an event is detected, the system prepares the content of the notification. This content typically includes information about the event that enables or suggests future action.
- **FCM Integration:** Once the notification content is prepared, the PowerEye system uses Firebase Cloud Messaging (FCM) as the communication bridge to send the notification to the user's mobile device. FCM simplifies the process of sending real-time notifications, ensuring that they are delivered promptly.

- **Mobile App Reception:** On the user's mobile device (IOS or Android), the PowerEye mobile application is integrated with FCM. When FCM sends a notification, the app receives and displays it as a push notification on the user's device.
- **User Interaction:** Users can interact with the push notifications to take recommended action. This interaction allows users to stay informed and make decisions based on the notifications.

Mobile Application:

Figure 12: *Mobile Application Component Diagram*



The mobile application basically is one of the important parts that should be done in order to have a connection between the user and the server. It works as the bridge between the user and the logic information saved at the back server. To make the application easy to navigate and implement we structured the folders as:

Src folder:

- **Components:** a folder that contains all the reusable UI components that could be used across multiple screens. Each reusable component will have its own style and asset

connecting to it. Also, each component could be in a folder that consists of other components for more organization if needed.

- **Screens:** is a folder that holds all the pages of our application. Each screen will have a separate folder that contains the javaScript file and any associated style or assets. The folder could have other subfolders for more organizations if needed.
- **API:** This folder contains all the API Calls and data fetching from the server. It includes all fetched data related to authentication, creating,retrieving, updating, and deleting requests.
- **Services:** here where you can find all the files related to our application logic and functionalities. All the data manipulation related to the data that has been fetched is done on these files.
- **Utils:** This folder contains the utility helpers such as the color theme used, typography, …etc.

App.js:

- This file is considered as the central/entrypoint of all the configuration needed to develop the application. Also, here where you can find all the firebase cloud messaging controllers needed for establishing the notifications.

Codebase Structure:

In this section, we will outline the folder structure and the roles of the files.

Back-End:

PowerEye-backend

```
├── .gitignore                ignore files for Git
├── codebase.tree             directory structure diagram
├── README.md                 project instruction
├── tracker_server/
└── web_server/
```

Web Server:

web_server

```
├── app                       main application logic
│   ├── config.py             configuration settings
│   ├── config_testing.py     testing configuration settings
│   ├── controllers           all logics
```

```
│   │   ├── appliance_controller.py    logics related to appliance
│   │   ├── energy_controller.py       logics related to energy
│   │   ├── power_controller.py        logics related to power
│   │   ├── room_controller.py         logics related to room
│   │   └── user_controller.py         logics related to user
│   ├──  extensions.py                 3rd party libraries; add extra functionality to Flask
│   ├── models                         Data models (DB)
│   │   ├── appliance_model.py         appliance model
│   │   ├── energy_model.py            energy model
│   │   ├── notified_device_model.py   notified device model
│   │   ├── power_model.py             power model
│   │   ├── room_model.py              room model
│   │   ├── user_model.py              user model
│   │   └── __init__.py                indicate a Python package
│   ├── uploads                        file system for profile pictures
│   │   └── images                     uploaded profile picture directory
│   ├── utils                          store modules that are used across system
│   │   ├── cloud_interface.py         methods related to Tuya & Meross
│   │   ├── enums.py                   define fixed set of options for a variable
│   │   ├── image_sys.py               methods related to profile picture  system
│   │   └── __init__.py                indicate a Python package
│   ├── views                          all routes (POST, GET, DELETE, PUT)
│   │   ├── appliance_views.py         routes related to appliance
│   │   ├── energy_views.py            routes related to energy
│   │   ├── power_views.py             routes related to power
│   │   ├── room_views.py              routes related to room
│   │   ├── user_views.py              routes related to user
│   │   └── __init__.py                indicate a Python package
│   └── __init__.py                    main class to run server
├── .env                               environment variables file
├── .flaskenv                          Flask environment variables file
├── README.md                          instruction on how to run the server
└── requirements.txt                   python package dependencies list
```

Tracker Server:

tracker_server

├── .secrets	secret configuration files
│ ├── .env	environment variables file
│ └── firebase-secret-key.json	firebase service account credentials
├── app	main application logic
│ ├── external_dependencies	integration with external systems
│ │ ├── fcm.py	integration with Firebase Cloud
│ │ ├── meross_module.py	integration with Meross Cloud
│ │ ├── mongo.py	integration built on PyMongo package
│ │ └── tuya_module.py	integration with Tuya Cloud
│ ├── interfaces	abstract base classes
│ │ ├── db.py	interface for database queries
│ │ ├── plug.py	interface for smart plugs cloud interactions
│ │ └── task.py	interface for tasks
│ ├── tasks	all the tasks
│ │ ├── checker.py	checks EPR events
│ │ ├── collector.py	collects data from Plug Cloud
│ │ ├── master.py	manage the creation and deletion of tasks
│ │ └── updater.py	updates energy data and ML models
│ ├── plug_controller.py	creates and control the Plug interactions
│ ├── recommender.py	energy personalized recommender
│ └── types_classes.py	enum classes for plug, energy, & notification types
│ └── __init__.py	indicate a Python package
├── logs/	storage for log files
├── models_filesystem	ML models for appliances
│ ├── cluster_models/	cluster models
│ └── forecast_models/	forecast models
│ └── test_models/	test models
│ ├── results	experimental results
│ │ ├── data/	data for results codes
│ │ ├── energy_comparison.py	energy comparison results
│ │ ├── execution_time.py	time execution for Checker.run()
│ │ └── ml_models.py	ML models scores
├── tests	Pytest tests
│ ├── integration/	integration tests
│ └──unit/	unit tests
│ └── __init__.py	indicate a Python package

├── requirements.txt	python package dependencies list
├── README.md	instruction on how to run the server
└── scheduler.py	main class to run the server

Frontend:

PowerEye-frontend

├── App.js	index file for connecting all screens
├── app.json	Configuration file
├── assets/	contains media used in project
│ ├── FAQ.pdf	pdf document
│ ├── images/	images used in the project
│ └── UserGuide.pdf	pdf document
├── node_modules/	all project dependencies
├── .expo/	React native cli for app configuration
├── babel.config.js	Configuration file for Babel, a JavaScript compiler
├── eas.json	Configuration file for the Expo Application Services
├── google-services.json	Google services for Android
├── GoogleService-Info.plist	Google services for iOS
├── package-lock.json	project dependencies and metadata
├── package.json	project dependencies and metadata
├── README.md	project instructions
├── src	contains all the code files needed
│ ├── api	all js file related to API calls
│ │ └──apiManager.js	API requests
│ ├── components	reusable codes shared among multiple screens
│ │ ├── addApplianceButton.js	reusable add device component
│ │ ├── AddNewRoom.js	reusable add room component
│ │ ├── appliancePageAnalysis	shared codes for appliance analysis
│ │ │ ├── Barchart.js	contains barchart for data analysis
│ │ │ ├── energyConsumption	first tap view navigation in appliance page
│ │ │ │ ├── month	second tap view (month)in appliance page
│ │ │ │ │ └── thirdTabView_ApplianceMonthEC.js	contains duration taps (JAN,FEB,..)
│ │ │ │ ├── secondTabView_ApplianceEC.js	contains taps(week, month,year)
│ │ │ │ ├── week	second tap view (week)in appliance page
│ │ │ │ │ └── thirdTabView_ApplianceWeekEC.js	contains duration taps (this,last,..)

```
│ │ │ │ └── year                    second tap view (year)in appliance page
│ │ │ │     └── thirdTabView_ApplianceYearEC.js   contains duration taps (this,last,..)
│ │ │ └── firstTabViewAppliance.js contains taps(EC, Cost,CO2)
│ │ ├── Card.js                     reusable  card design
│ │ ├── deleteButton.js             reusable delete button
│ │ ├── devicesIcons.js             reusable icons
│ │ ├── goalAchievement.js          reusable goal progress bar
│ │ ├── GradientProgressBar.js      colored progress bar line
│ │ ├── MyRoomComponent.js          reusable room component
│ │ ├── navigation                  navigation stack codes
│ │ ├── removeGoal.js               removing goal button
│ │ ├── roomPageAnalysis            shared codes for room analysis
│ │ ├── saveGoalChanges.js          saving goal button
│ │ ├── scrollableCard.js           scrollable cards structure
│ │ └── totalPageAnalysis           total analysis using  PieChart
│ ├── screens                       all pages of the application
│ │ ├── deviceCard
│ │ │ └── deviceCard.screen.js      deviceCard implementation
│ │ ├── goalAchievement
│ │ │ └── goalAchievement.screen.js goal page implementation
│ │ ├── home
│ │ │ ├── adddevice.js              add device implementation
│ │ │ ├── addDeviceModal.js         modal appear when adding device
│ │ │ ├── deviceCard.js             device cards implementation
│ │ │ ├── deviceCardsSwiper.js      card swiper implementation
│ │ │ ├── devicesIcons.js           device icons  implementation
│ │ │ ├── goalAchievement.js        gaol bar  implementation
│ │ │ ├── GradientProgressBar.js    colored progress  implementation
│ │ │ ├── header.js                 header  implementation
│ │ │ ├── home.screen.js            all home implementations
│ │ │ ├── noPlugsFound.js           modal appear when no plug found
│ │ │ ├── totalEnergyConsumption.js total consumption card code
│ │ │ └── unableToRetrievePlugs.js  modal unable to retrieve plug
│ │ ├── loading
│ │ │ └── loading.screen.js         implementing the starting page
│ │ ├── login
```

```
│   │   │   └── login.screen.js              implementing login page
│   │   ├── notification
│   │   │   └── notification.screen.js       implementing notification page
│   │   ├── profile
│   │   │   ├── deleteAccount.js             implementing modal of delete account
│   │   │   ├── deletedAccountMessage.jsimplementing the leaving msg
│   │   │   ├── EditPersonalInfo.screen.js implementing edit info page
│   │   │   ├── faq.screen.js                attaching FAQ Pdf
│   │   │   ├── helpCenter.screen.js         help center
│   │   │   ├── logoutModal.js               logout modal msg
│   │   │   ├── profile.screen.js            profile page
│   │   │   └── userGuide.screen.js          attaching user guidePdf
│   │   ├── room
│   │   │   ├── deviceCard.js                device cards implementation
│   │   │   ├── deviceCardsSwiper.js         room card swiper
│   │   │   ├── devicesIcons.js              device icons
│   │   │   ├── room.component.js            implementing models
│   │   │   └── room.screen.js               all room components
│   │   ├── signup
│   │   │   └── signup.screen.js             signup  page
│   │   └── totalEnergyConsumption
│   │       └── totalEnergyConsumption.screen.j all total components
│   ├── services
│   │   ├── appliance.context.js all room functionalities
│   │   ├── calculationFunctions.js          calculations detail
│   │   ├── powerEye.context.jsall shared variables across the app
│   │   ├── theme
│   │   │   ├── colors.js                    all colors used with theme service
│   │   │   ├── fonts.js                     all font used with theme service
│   │   │   ├── index.js                     all theme used
│   │   │   ├── sizes.js                     all sizes used with theme service
│   │   │   └── spacing.js                   spacing used with theme service
│   │   └── theme.context.js                 theme to be parent for its files
│   └── utils                                shared utilities as font, and color
│   └── colors.js                            our color pallet defined
└── .gitignore                               ignore files for Git
```

Integration with External Systems:

The PowerEye system is a Home Energy Monitoring System (HEMS) designed for detailed appliance-level energy monitoring, and for this purpose, it leverages the use of smart plugs. Since developing our own smart plugs was not feasible, we chose to utilize commercial smart plugs. These off-the-shelf smart plugs cannot be directly coded or controlled, leading to the need of interacting with their respective cloud platforms. Our choice for smart plugs led us to Meross, primarily due to their affordability and the availability of a user-friendly API for cloud interaction. In contrast, other brands often came with higher price tags, lacked public APIs, or offered paid APIs. We also acquired another brand, Tuya, known for its cost-effectiveness, although the automation of user registration was not feasible for our system. As a result, we have only two Tuya users, 'Ward' and 'Qatar Alnada', who serve the sole purpose of providing additional data for testing our AI algorithms.

While our system is currently primarily configured for Meross, its design is flexible, allowing for easy extension to work with any other smart plug cloud. Our system communicates with smart plugs' clouds using three main functions: **login()** to gain user access, **get_devices()** to retrieve the list of available devices in the cloud, and **get_info()** to gather specific device information such as power usage, on/off status, and connection status. For more details about these functions, please refer to the PowerEye-backend repository.

Algorithms:

In this section, we will emphasize the Tracker server, which serves as the system's core. We'll present a high-level overview of the program's flow through flowcharts. Furthermore, we'll focus on the algorithms employed in the Energy Personalized Recommender (EPR), while Machine Learning (ML) algorithms will be addressed in a Jupyter Notebook.

Data Structures:

Our server primarily leveraged standard Python data structures like lists, tuples, and dictionaries, which proved adequate for our needs. For machine learning algorithms, we also utilized data structures provided by the Pandas library. Pandas, a versatile library for data science and machine learning uses, introduced us to its powerful data structures, namely series and dataframes.

Mathematical formulation:

The only directly recorded quantity in our system is Power. All other quantities are derived.

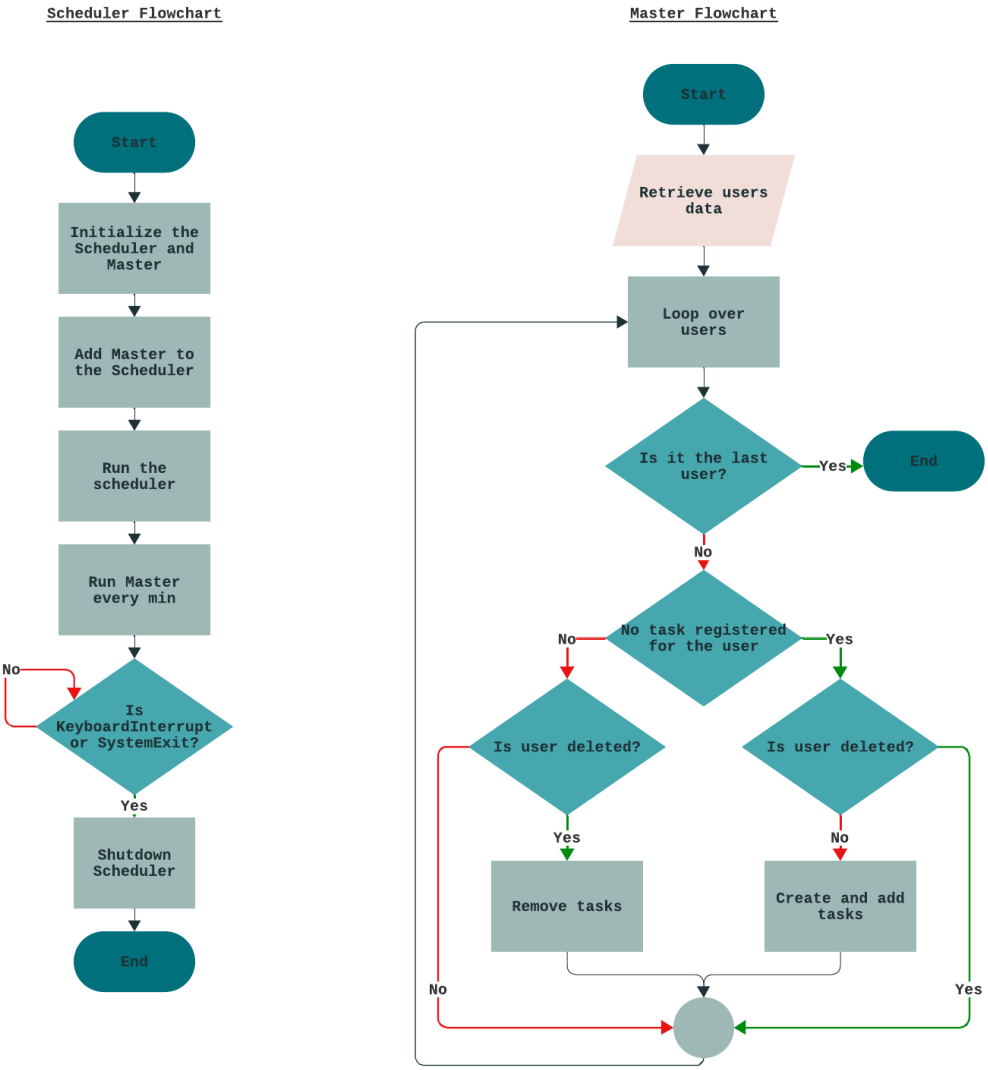
Table 10: *PowerEye Mathematical Quantities*

Quantity	Formula	Unit	Description
Power(***P***)	–	***W***	Power quantity is received from the smart plugs through their respective clouds at minute intervals, and it is subsequently stored in the 'Powers' collection
Energy(***E***)	$\sum_{i=1}^{n} P_i \cdot \Delta t = \sum_{i=1}^{24*60} \frac{P_i}{1000} \cdot \frac{1}{60}$	***kWh***	Energy quantity is calculated by converting power from watts to kilowatts, then multiplying it by the time interval, which is set at one minute (1/60 hour). The daily energy consumption values are aggregated and stored in the 'Energys' collection.
Energy Cost(***Cost***)	$E \cdot RCT = E \cdot \frac{0.8}{100}$	***SAR***	Energy cost is calculated by multiplying the energy consumption by the **Residential Consumption Tariff**, which is obtained from the official Saudi Electricity Company website. This calculation is performed directly on the frontend side, eliminating the need to store this value.
CO2 Equivalent (***CO2e***)	$E \cdot ECF = E \cdot 0.569$	***kg***	Carbon Dioxide Equivalent (CO2e) is a unit that quantifies the impact of greenhouse gasses. It's determined by multiplying energy consumption by the **Electricity Conversion Factor**, which can vary depending on the energy source and regional factors. In Saudi Arabia, ECF is set at 0.569 (Abdulkareem & Ellaboudy, 2023). This calculation is done directly on the frontend, eliminating the need to store the value.

Tracker Server Program Flow:

The server begins with the Scheduler, which in turn adds the Master task. The Master task runs every minute . It creates the following tasks: Collector, Checker, and Updater for each user.

Figure 13: *Scheduler and Master Flowcharts*



The Collector task runs every minute, utilizing the PlugController to communicate with the Plug Cloud. Likewise, the Checker task also runs every minute and is responsible for validating all EPR events and resetting their flags at the appropriate intervals. The Updater task operates daily, primarily focusing on energy calculation, transfer to its designated collection, and ML model training. Additional details can be found in the Energy Personalized Recommender section.

Figure 14: *Collector Flowchart*

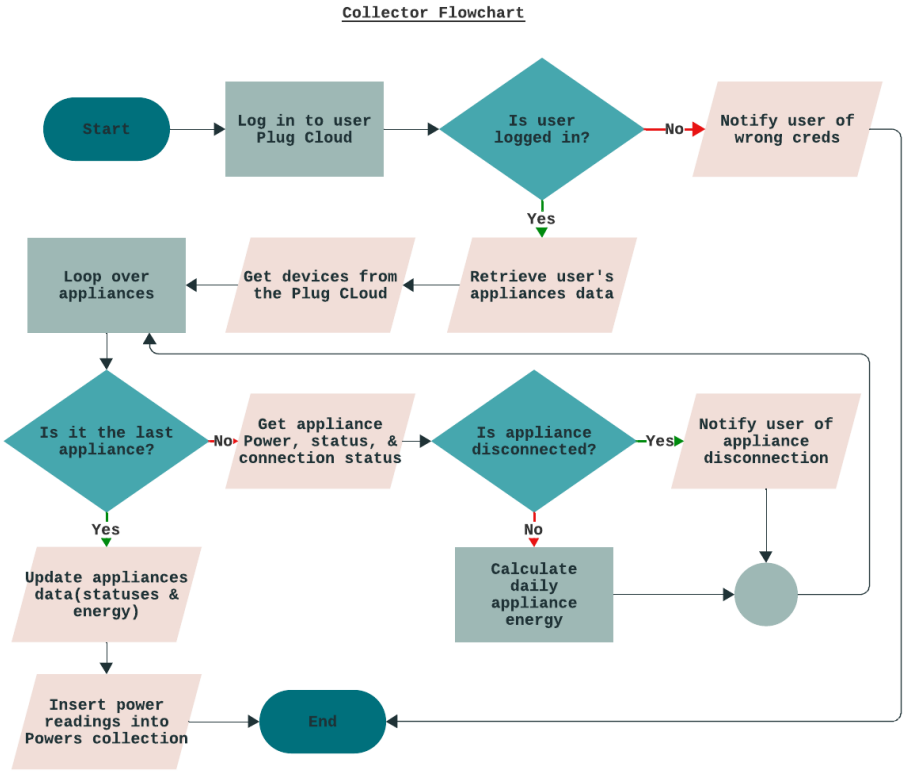


Figure 15: *Checker Flowchart*

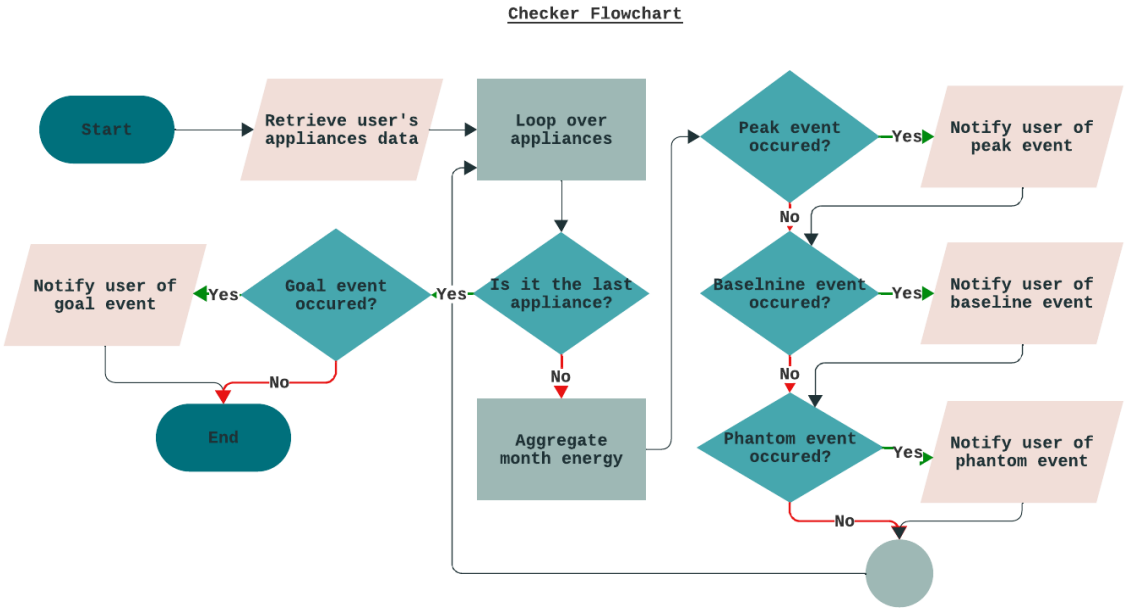
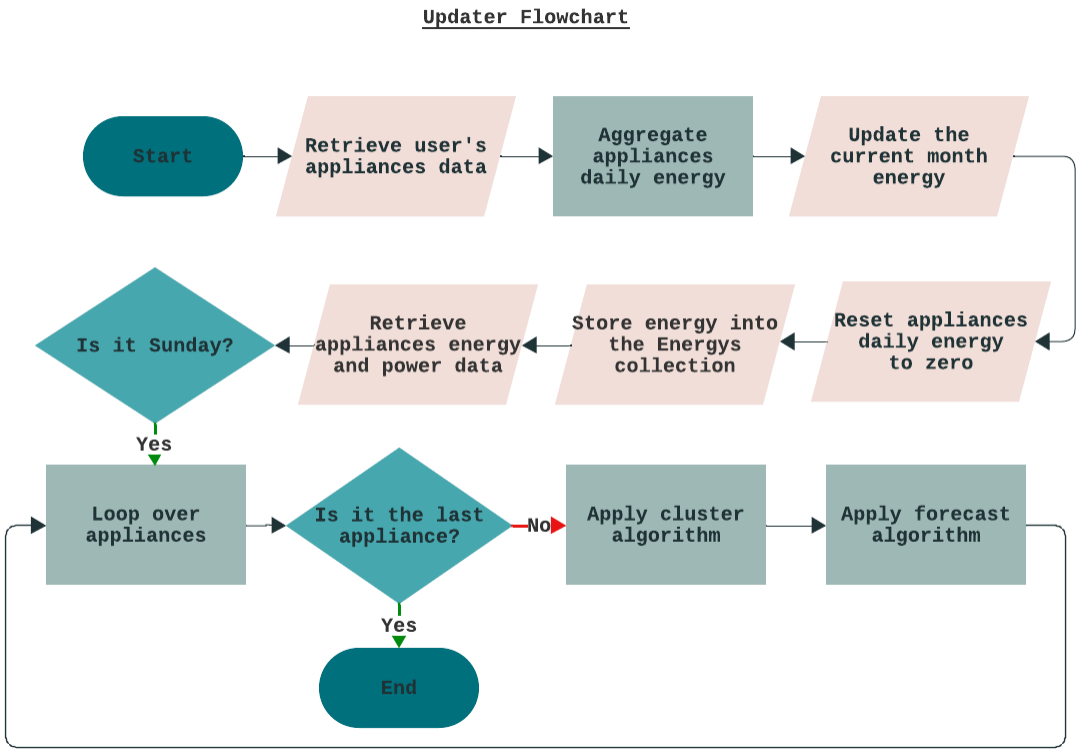


Figure 16: *Updater Flowchart*



Energy Personalized Recommender (EPR):

The primary aim of home energy monitoring systems (HEMS) is to enable users to monitor and enhance their energy consumption. This can be achieved through personalized recommendations for reducing energy use or direct automation. While automation is efficient, recommendations offer flexibility and personalization. Automation follows fixed rules, while recommendations adapt to individual preferences.

Various models are employed for energy optimization in recommender systems, including collaborative filtering, content-based, and hybrid (Techlabs, 2021). Collaborative filtering suggests recommendations based on similar users' behaviors, content-based uses item features, and hybrid combines both. Our EPR adopts a content-based approach, aligning with our user-centric design and appliance data richness. While we begin with this approach, we retain the flexibility to incorporate collaborative filtering or other methods in the future for increased adaptability. Check the appendix for more details.

In our proposed recommender system, we focus on four aspects of energy optimization: energy goal, peak time, phantom mode, and energy baseline.

- **Energy goal:** Users can customize their monthly energy consumption goals, specifying them in terms of cost. This added flexibility allows users to align their goals with their

preferences. Our system calculates the energy corresponding to the cost goal. Users are then notified when their energy consumption hits specific percentage thresholds, ranging from 25% to 200%.

- **Peak time:** avoiding appliance usage during peak times benefits the environment by reducing emissions and conserving resources. It also helps governments save on infrastructure costs, improve grid reliability, and promote renewable energy use. For that reason we will inform the user to postpone using appliances during peak times. In Saudi Arabia, peak time is recognized by the government as 1-5 pm (Saudi Electricity Company). Certain appliances are categorized as shiftable appliances[5] based on their energy patterns.
- **Phantom mode:** in appliances refers to their ability to consume electricity even when they're turned off but still plugged in. This happens because they stay in standby mode to respond quickly when needed. In this scenario we will inform the user to turn off appliances in their phantom mode. Certain appliances are categorized as phantom appliances based on their energy patterns. Since different appliances have different phantom power, k-means algorithm is used here to cluster the data into two clusters on_cluster and phantom_cluster.
- **Energy baseline:** Our system provides users with insights into their energy consumption patterns by sending alerts when they exceed their daily baseline. Each appliance in the system is equipped with its own model, allowing it to forecast energy consumption for the upcoming day. This personalized forecasting enhances user awareness and helps manage energy usage more effectively.

Pseudocode:

In this section, our focus shifts to the core logic behind the Energy Personalized Recommender (EPR) and its encapsulation within the Checker class. EPR functions provide contextless event logic, while the Checker class takes on the vital task of invoking these functions at the right moments. It handles flag setting and resetting, and user notifications, all of which are essential responsibilities of the Checker class, as illustrated below.

Energy Goal:

The **check_goal()** function, provided by the **EPR** module, calculates the percentage of the user's energy goal. This percentage is determined by dividing the current month's energy consumption by the user's goal, rounding it down to the nearest quarter, and ignoring any quarter greater than 2. The result is then converted into a percentage out of 100.

[5] Check the appendix for more information

Within the **Checker** task, this logic is encapsulated in the **notify_goal()** function. It first checks whether the user has set an energy goal. If so, it calls **check_goal()**. If it returns a percentage, and the corresponding flag is true, the **FCM** module sends a notification. This notification includes the user ID, notification type, and percentage. Subsequently, the flag for that percentage is set to false to prevent additional notifications for the same percentage in the current month.

Pseudocode 1: *Energy Goal*

```
INPUT: month_enegry(float), goal(float)
OUTPUT: (int)
ALGORITHM check_goal(month_enegry, goal):
    percentage <- month_enegry / goal
    rounded <- Lpercentage * 4⌋ / 4
    IF rounded <= 2:
        RETURN INT(rounded * 100)
    ELSE:
        RETURN 0

INPUT: month_enegry(float), goal(float)
OUTPUT: _
ALGORITHM notify_goal(month_enegry, goal):
    IF goal > 0:
        percentage <- PR.check_goal(month_enegry, goal)
        IF percentage AND goal_flags[percentage]:
            fcm.notify(user_id, NotifType.GOAL, percentage)
            goal_flags[percentage] <- FALSE
```

Peak Time:

The **check_peak()** function, provided by the EPR module, offers a simple logic. It checks whether the appliance is turned on during peak times and checks its energy type, if it's categorized as shiftable.

Within the **Checker** task, this logic is encapsulated in the **notify_peak()** function. First, it verifies the flag for the specific appliance.If it is, the **check_peak()** function is invoked. If it returns true, the **FCM** module sends a notification, including the user ID, notification type, and appliance name. Following this, the flag for that appliance is set to false to prevent additional notifications for the same appliance during the current day.

Pseudocode 2: *Peak Time*

```
INPUT: cur_hour(int), status(boolean), e_type(EType), types(list)
OUTPUT: (boolean)
ALGORITHM check_peak(cur_hour, status, e_type, types):
    is_peak <- PEAK_START <= cur_hour < PEAK_END
    RETURN is_peak AND status AND e_type IN types

INPUT: id(str), status(boolean), e_type(EType), name(str)
OUTPUT: _
ALGORITHM notify_peak(id, status, e_type, name):
    IF peak_flags[id]:
        IF PR.check_peak(cur_hour, status, e_type, shiftable):
            fcm.notify(user_id, NotifType.PEAK, name)
            peak_flags[id] <- FALSE
```

Phantom Mode:

The **check_phantom()** function, provided by the EPR module, performs a series of checks to determine if an appliance is in phantom mode. First, it checks if the appliance already has a cluster model or not. Then, it checks if the appliance is turned on. If the appliance is on, it uses the model to predict the cluster of the power value. Additionally, the cluster algorithm predicts the cluster of power with the value 0 to compare that cluster with the predicted one. If the predicted cluster matches the cluster of 0 reading, it indicates that the appliance is in phantom mode, and the function returns True.

Within the **Checker** task, this logic is encapsulated in the **notify_phantom()** function. The first step is to verify the flag for the specific appliance. If it's set, the function loads the model from the models' file system using the internal **get_model()** function. After that, it invokes the **check_phantom()** function. If this function returns True, the **FCM** module sends a notification that includes the user ID, notification type, and appliance name. Following this, the flag for that appliance is set to False, and the timestamp is reset to the current time, preventing additional notifications for the same appliance in the following hour.

Pseudocode 3: *Phantom Mode*

```
INPUT: model(clusterModel), power(float), status(boolean)
OUTPUT: (boolean)
ALGORITHM check_phantom(model, power, status):
    IF model AND status:
```

```
            predicted_labels <- model.predict([0, power])
            RETURN  predicted_labels[0] == predicted_labels[1]
        RETURN FALSE

INPUT: id(str),  power(float),  status(boolean), name(str)
OUTPUT: _
ALGORITHM notify_phantom(id,  power,  status,  name):
    IF phantom_flags[id][0]:
        model <-  get_model(id)
        IF PR.check_phantom(model,  power,  status):
            fcm.notify(user_id, NotifType.PHANTOM, name)
            phantom_flags[id][0] <- FALSE
            phantom_flags[id][1] <- Current timestamp
```

Energy Baseline:

The **check_baseline()** function, provided by the EPR module is intricate, and its details are excluded here for brevity. However, you can find an in-depth discussion in the accompanying Jupyter notebook.

Within the **Checker** task, this logic is encapsulated in the **notify_baseline()** function. Initially, it verifies the flag for the specific appliance and checks whether the appliance has an associated baseline. It's important to note that not all appliances are eligible to have a forecast model. If the conditions are met, the **check_baseline()** function is invoked. If it returns true, the **FCM** module sends a notification, including the user ID, notification type, and appliance name. Subsequently, the flag for that appliance is set to false to prevent additional notifications for the same appliance during the current day.

Pseudocode 4: *Energy Baseline*

```
INPUT: id(str),  baseline(float),  name(str),  powers(int)
OUTPUT: _
ALGORITHM notify_baseline(id,  energy,  baseline,  name):
    IF baseline > 0 AND baseline_flags[id]:
        model <-  get_model(id)
        IF PR.check_baseline(baseline, powers, model):
            fcm.notify(user_id, NotifType.BASELINE, name)
            baseline_flags[id] <- FALSE
```

All flags are set to TRUE at the beginning, this table shows when and where they are reset:

Table 11: *Flags Description*

Name	Data structure	When
goal_flags	{<percentage>: boolean}	At the beginning of each month (day=1, hour=1, min=1)
peak_flags	{<appliance_id>: boolean}	At the beginning of each day (hour=1, min=1), seconds are ignored
phantom_flags	{<appliance_id>: [boolean, <timestamp>]}	After one hour from the last notification for the same appliance, a timestamp is stored within the flag. This timestamp helps track the period for each appliance
baseline_flags	{<appliance_id>: boolean}	At the beginning of each day (hour=1, min=1), seconds are ignored

The remaining EPR functionalities are elaborated upon in the Jupyter Notebook under 'reme2718' Kaggle account.

Version Control and Collaboration:

In our PowerEye project, we've implemented an efficient system for managing our code and collaborating effectively. We leverage the power of Git, which seamlessly integrates with Visual Studio Code, and we use GitHub as our central platform for storing and sharing our project. Our collaboration approach is centered around a branch-based workflow.

- **Individual Workspaces:** Each team member has their own dedicated branch. This branch is separate from the main project.
- **Parallel Development:** Having separate branches means that team members can work on different parts of the project at the same time without interfering with one another.
- **Committing Progress:** As team members work on their tasks, they make incremental changes and save them within their own branches. It helps keep the main project stable and prevents errors from slipping in.
- **Code Reviews:** Once a team member completes their work in their branch, they request a review from other team members. This is like having a group of editors who double-check your work for any mistakes or improvements before it's officially included in the project.
- **Merging into the Master:** When the team leader agrees that the work in a branch is ready, the changes are "merged" into the Master branch.

This approach not only keeps our project organized and prevents conflicts but also ensures that only well-tested and approved code becomes a part of the main project. You can find the

PowerEye-backend repository under the 'rem2718' account here and the **PowerEye-frontend** repository under the 'RaneemBalharith' account here.

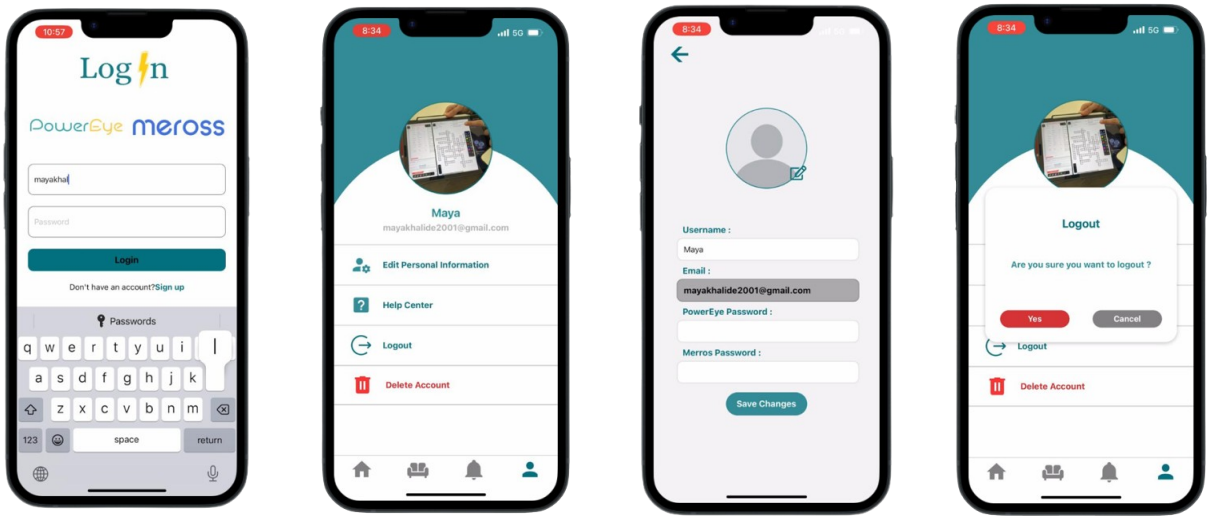
User Interface (UI) Design:

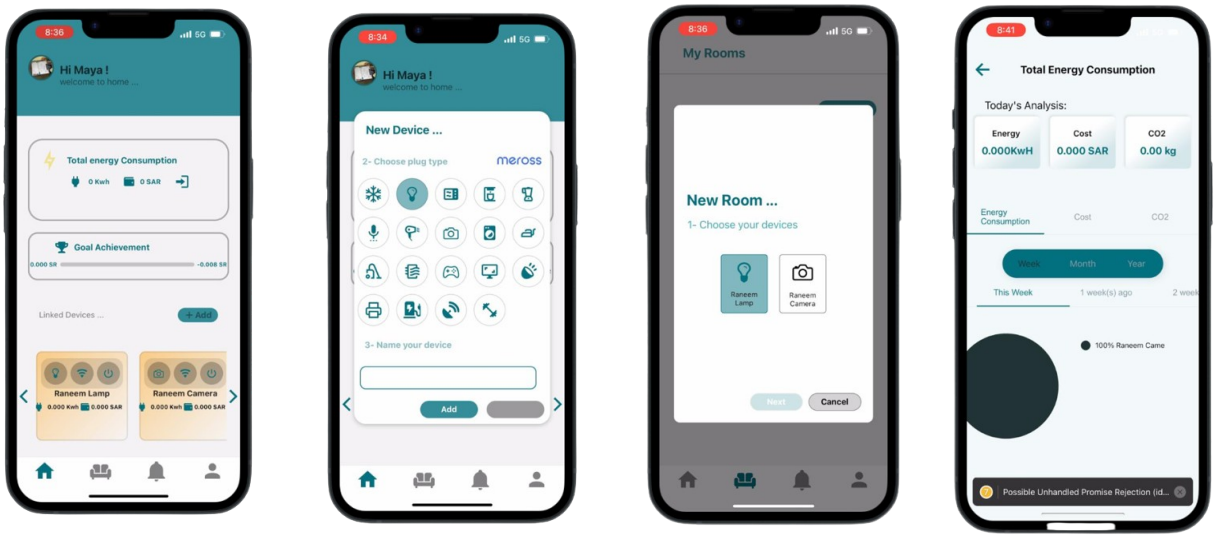
User Interface Design Process:

User Interface (UI) design is the systematic process of designing an effective and easy-to-use interface. There are several stages that go into a successful UI design. The process starts by defining the scope of the project, as well as understanding user needs and target audience. Various methods are used to get the user's decision, such as interviews, questionnaires, etc. Their suggestions and preferences have been taken into account while designing the UI. Sketches or wireframes are used to visualize the layout and structure of the interface. Then, the decision on visual elements like typography and color is made. A low-fidelity prototype is created to gather feedback on the design and if necessary, refine it further. Iteration and refinement is a continuous process based on feedback and usability aspects. All team members work together to ensure a smooth implementation of the project. Continuous evaluation and improvement are done to make sure the product is functioning properly. By following this process we have created interfaces that satisfy user needs and provide an outstanding user experience.

Screenshots or mockups of the UI:

Figure 17: *Mobile Application mockups - Final UI*





User Experience:

While developing the PowerEye app, the user's experience has been taken into account. Users' journey will be seamless and intuitive while interacting with the system. From the moment users engage with PowerEye, they are greeted with a visually appealing and user-friendly interface that guides them through the various features and functionalities. Users can use the bottom bar to navigate through different screens smoothly. Inside each screen, users can easily understand the functionalities provided as visual elements have been added to make the page content much clearer for them. Feedback mechanisms are in place to provide users with a sense of control and assurance, allowing them to track progress and make informed decisions. Overall, the user experience is designed to foster a sense of delight and satisfaction, leaving users with a positive impression and a desire to continue engaging with the product or service.

Visualization:

In order to make the analysis for the energy consumption, cost, and CO2 much clearer to PowerEye users, charts were used instead of written words. To implement these charts, **react-native-chart-kit, HorizontalBarGraph** libraries components were used. Through this library different types of charts can be imported with predefined attributes and properties that make the implementation easier. Once a chart is imported, you can easily add the labels, data, styles, legend, .etc. This library makes the analysis more appealing for users, and it makes the understanding of the analysis much clearer.

Implementation Process:

Strategy:

A depth-first strategy focuses on developing a single component or feature fully before moving on to the next, diving deep into one aspect at a time. In contrast, a breadth-first strategy aims to build and integrate multiple components or features simultaneously, covering a wider scope across the system's breadth. A depth-first strategy is better suited for our system implementation based on the priority of features.

In our implementation, we follow a depth-first strategy by initially focusing on high-priority features. We dedicate our resources and effort to the core functionalities, ensuring their reliability and efficiency. Once these critical components are successfully implemented, we gradually expand the system's capabilities by incorporating additional features. This approach allows us to maintain a strong foundation while systematically building upon it, ensuring that our system remains stable and reliable throughout its development.

Iterations Details:

Table 12: *Iterations Description*

Iteration	Tracker Server	Mobile Application
1st iteration	- Track Cost Goal - Check for New User/Appliance - Collect and store Power, Energy & appliances data - Update appliances data - Notify for Meross Password Invalidation - Notify for Near Goal Proximity - Notify for Exceeding the goal	- User account registration - User login page - User ability to logout - View user Information - Setting/tracking cost goal - Adding new appliance - Controlling the switch on appliances - Display historical data per appliance - Display historical data for total - Goal Comparison with past month goal - Changing appliance/room name - Display power current usage
2nd iteration	- Notify for Device Disconnection - Notify for reaching the goal - Send Recommendation based on peak times - Send Recommendation based on Phantom Mode - Send Recommendation based on Energy Baseline	- Deleting account - User edit personal information - User edit/delete the cost goal - Delete appliance - Create/delete room - Add/remove appliance to room - Controlling room switch - Display historical data per room - Access FAQ and User guides - API calls request & data fetching - Authentication, session, and caching

Performance Optimization:

All components within our system adhere to standardized frameworks, optimizing their performance through efficient algorithms. However, the Tracker Server stands apart. It serves as the core scheduler for vital system tasks. Although frameworks like Celery exist for such purposes and cater to distributed systems and scalability, their complexity poses challenges. Hence, we opted to develop this server from scratch, leveraging beneficial libraries, primarily APScheduler.

The Tracker Server employs multithreading. Each user's Collector, Checker, and Updater tasks operate on separate threads. APScheduler manages thread creation and termination (Master Task), and determines when to awaken and sleep each thread (Scheduler).

This approach empowers the Tracker Server to manage task loads and allows scalability by adding hardware resources. However, a bottleneck arises if a user possesses numerous appliances. Our database embedded the appliances' documents within the user document, which Mongodb poses limitations on document size. Additionally, having one thread per user checking all their appliances might strain the server as the number of appliances increases. As a solution, we assume a reasonable number of appliances for each user.

Conclusion:

In conclusion, The PowerEye project is meticulously structured to control, interpret, optimize, and monitor smart plugs using the data to build a connection with users. The system is thoughtfully constructed and divided into modular components, with a tree-cli codebase, a comprehensive flowchart. This user-centric approach extends to the selection of tools and languages, with each chosen to align perfectly with the specific environmental requirements, ensuring the utilization of the most effective and efficient resources.

The system architecture employs a systematic component breakdown, connecting each component to its corresponding elements to facilitate data transfer and connection. Emphasizing a seamless UI experience with MongoDB, Meross cloud, The web server. Also a powerful data control and management with the tracker server. The project's structure is designed to create a seamless connection between each component, emphasizing efficiency and collaboration. The project's implementation integrates seamlessly with external systems, particularly through the utilization of Meross capabilities. The project's meticulously designed algorithms emphasize ERP and draw insights from continuous research, leveraging versatile ML libraries like Pandas for data analysis and structures.

TESTING

Test case specification, testing and quality assurance documentation, and test results

Test Planning:

Test Objectives:

Software testing serves several purposes during an application's development lifecycle, contributing significantly to the overall quality and dependability of the product. Our primary goal is to test the functionality of key features. Our key goals in testing are to find and resolve bugs, validate that PowerEye satisfies defined criteria, ensure the application performs as intended across various settings, and improve the overall user experience. Additionally, our testing efforts contribute to evaluating the validity of all functions and component integrations Overall, our application testing goals include producing a good-quality usable product.

Test Levels:

Testing occurs at several stages in software development to ensure that our application is reliable and operates as planned. The first step is unit testing, where we isolate individual components or modules, promoting code quality by detecting and fixing bugs early. Following that, we move to integration testing, analyzing how distinct units interact to ensure smooth communication between components and detect any collaboration-related errors. Furthermore, there is system testing, examining the entire application to determine if PowerEye meets the criteria and functions properly in a real-world scenario. Finally, we ensure the quality of the application through acceptance testing, which confirms that the application satisfies requirements and functions as expected in its intended environment. This phase is critical for developing confidence in the application's capacity to meet user demands and expectations.

While acknowledging multiple test levels in software development, this document emphasizes the CS team's specific focus on unit and integration for the tracker server. For more details about unit and integration tests for the web server and mobile application, please refer to the Software Testing Document. This targeted focus underscores the team's dedication to a meticulous and effective testing strategy aligning with the PowerEye project's goals and objectives.

Test Types:

Focusing on unit, integration, and usability testing is essential for multiple reasons to ensure the robustness and reliability of our software. Unit testing allows us to isolate and examine specific components or modules, promoting code quality by detecting and correcting bugs at an early stage. This meticulous examination of isolated units ensures that each component performs its designated functions correctly, contributing to a solid foundation for the entire system. Our focus on unit testing includes **Component Testing**, which aims to verify the functionality of individual components in isolation. The purpose of Component Testing is to ensure that each isolated unit operates as intended, laying the groundwork for a dependable software system.

Moving beyond unit testing, integration testing becomes crucial in ensuring seamless communication between distinct units and detecting any collaboration-related errors. By emphasizing integration testing, we address the complexities of how different modules interact, providing insights into the overall system's functionality. Our focus on integration testing includes **Top-Down Integration Testing**, which starts from the top-level modules and progresses downward, simulating interactions between higher and lower-level components. The purpose of Top-Down Integration Testing is to identify and address issues in the integration of modules early in the testing process.

Finalizing with usability testing for effectiveness and user-friendliness of the software. Usability testing focuses on assessing how well the software meets the needs of its end-users, identifying areas for improvement in terms of user interface, navigation, and overall user experience. By conducting usability testing, we can address potential usability issues before the software is released, enhancing user satisfaction and reducing the likelihood of post-launch adjustments.

Test Technique:

We highlight the significance of functional and usability testing in our application testing strategy to provide a strong and user-friendly solution. Functional testing is essential for analyzing the application's main features and verifying that each feature works as intended and satisfies the requirements. This thorough assessment aids in the identification of any differences between predicted and actual results, assuring the application's dependability and correctness. Concurrently, usability testing is critical in reviewing the user experience, concentrating on how intuitive and accessible the program is for its target audience. We hope to detect and correct any usability flaws, increase user happiness, and improve overall usability by engaging actual users.

Testing Environment and Tools:

In order to successfully navigate through all testing levels, it is imperative to ensure the testing environment is well-prepared and meets the necessary criteria.

Software:

Our testing methodology is diverse, comprising both unit testing using **Pytest** in Python and integration testing. For unit testing, we leverage Pytest, a widely used testing framework in **Python**, in conjunction with 'MagicMock' from the 'unittest' module. This enables the creation of mock scenarios for controlled evaluations, offering a comprehensive understanding of the software's behavior without actual connections to external systems. On the other hand, integration testing involves the utilization of real data sourced from the test database that we added from real data, ensuring seamless integration with authentic data sources. This dual-pronged strategy enables us to gain in-depth insights into the software's functionality across a spectrum of scenarios.

Hardware:

Our testing environment is designed to be versatile, accommodating **any** computer setup. This inclusivity allows testers to execute and evaluate the system's performance across a range of hardware configurations. By prioritizing ease of testing and accessibility, we create an environment that closely mirrors real-world conditions, enhancing the reliability and relevance of our testing processes.

Testing Execution:

In the process of testing execution, accessing and running tests is designed to be simple. Testers navigate to the "tests" folder within the **tracker_server** directory and choose the specific folder corresponding to their testing preference, whether it be for **unit** or **integration** tests. The execution is initiated by simply writing "***pytest***" in the command line.

Unit Testing for Tracker Server:

In the testing domain, comprehensive documentation is vital to ensure clarity and precision in evaluating software functionalities. The provided table furnishes crucial information for understanding and executing tests. Each "**Test Case**" example is uniquely identified by a label such as TC-1 or TC-2, serving as a reference point. The "**Input Space**" column delineates the input parameters, providing insight into the varied values utilized during testing. Simultaneously, the "**Output Space**" column offers a high-level description of anticipated outcomes, specifically related to the test_check_goal() function in this context. For enhanced comprehension, the

"**Input**" and "**Expected Output**" columns present concrete examples, illustrating the specific input values and their corresponding expected results in the given test cases. This structured format facilitates a systematic approach to interpreting and implementing the outlined tests.

Green: Regular Case

Red: Edge Case

Recommender Test Cases:

Table 13: *Recommender-Method-1 Test Cases*

Test	Input Space	Output Space	Test case	Input	Expected Output
test_check_goal()	Month_energy ∈ R Goal ∈ R	[0, 25, 50, 75, 100, 125, 150, 175, 200]	**TC-1:** month_energy less than goal	month_energy: 50 goal: 100	50
			TC-2: month_energy equal to goal	month_energy: 100 goal: 100	100
			TC-3: month_energy greater than goal	month_energy: 150 goal: 100	150
			TC-4: goal is zero	month_energy: 50 goal: 0	0
			TC-5: goal is negative	month_energy: 50 goal: -1	0

Table 14: *Recommender-Method-2 Test Cases*

Test	Input Space	Output Space	Test case	Input	Expected Output
check_peak()	status ∈ boolean e_type ∈ EType types ∈ list of shiftable device types	[True, False]	**TC-1:** Shiftable device is on during peak time	status: True e_type: "Shiftable" types:["Type1","Type2"]	True
			TC-2: Shiftable device is off during peak time	status: False e_type: "Shiftable" types: ["Type1", "Type2"]	False
			TC-3: Non-shiftable device is on during peak time	status: True e_type: "NonShiftable" types: ["Type1", "Type2"]	False
			TC-4: Non-shiftable	status: False	False

			device is off during peak time	e_type: "NonShiftable" types: ["Type1", "Type2"]	
			TC-5: Minimum input values	status: False, e_type: "Shiftable", types: []	False
			TC-6: All shiftable types are included	status: True, e_type: "Shiftable", types: ["Type1", "Type2"]	True
			TC-7: All non-shiftable types are included	status: False, e_type: "NonShiftable", types: ["Type1", "Type2"]	False
			TC-8: Maximum input values	status: True, e_type: "Shiftable", types: ["Type1", ..., "TypeN"]	True
			TC-9: Non-shiftable device with no types specified	status: False, e_type: "NonShiftable", types: []	False
			TC-10: Shiftable device with no types specified	status: True, e_type: "Shiftable", types: []	False

Table 15: *Recommender-Method-3 Test Cases*

Test	Input Space	Output Space	Test case	Input	Expected Output
check_phantom()	Model ∈ KMeans power ∈ float status ∈ bool	[True, False]	**TC-1:** Device in phantom mode	model: valid KMeans, power: 50, status: True	True
			TC-2: Device not in phantom mode	model: valid KMeans, power: 50, status: False	False
			TC-3: Model is None	model: None, power: 50, status: True	False
			TC-4: Device is off	model: valid KMeans,	False

				power: 50, status: False	
			TC-5: ‘power` is the minimum possible value	model: valid KMeans, power: 0, status: True	False
			TC-6: `power` is the maximum possible value	model: valid KMeans, power: 100, status: True	True
			TC-7: `power` is a negative value	model: valid KMeans, power: -1, status: True	False
			TC-8: `power` is a large positive value	model: valid KMeans, power: 1000, status: True	True

Table 16: *Recommender-Method-4 Test Cases*

Test	Input Space	Output Space	Test case	Input	Expected Output
test_check_baseline()	energy∈ float, baseline∈ float	[True, False]	**TC-1:** Energy exceeds baseline	energy: 150, baseline: 100	True
			TC-2: Energy equals baseline	energy: 100, baseline: 100	False
			TC-3: Energy is below baseline	energy: 50, baseline: 100	False
			TC-4: Baseline is zero	energy: 50, baseline: 0	True
			TC-5: Baseline is negative	energy: 50, baseline: -50	False

FCM Test Cases:

Table 17: *FCM Test Cases*

Test	Input Space	Output Space	Test case	Input	Expected Output
test_notify()	type (NotifType) ∈ {NotifType.CREDS, NotifType.DISCON	{title, body}	**TC-1:** CREDS notification	NotifType.CREDS, {}	('Update Your Login', 'Please update

	NECTION, NotifType.GOAL, NotifType.PEAK, NotifType.PHANTOM, NotifType.BASELINE}				your login information for Meross.')
			TC-2: DISCONNECTION notification	NotifType.DISCONNECTION, {'app_name': 'Test Device'}	('Device Not Working', 'Your Test Device is currently not working. Check its connection and fix it for better suggestions.')
			TC-3: GOAL notification	NotifType.GOAL, {'percentage': 80}	('Monthly Usage Goal', "You're close to reaching 80% of your monthly usage goal.")
			TC-4: PEAK notification	NotifType.PEAK, {'app_name': 'Test Device'}	('Peak Usage Alert', 'Try not to use Test Device after 5 PM. Click here to turn it off.')
			TC-5: PHANTOM notification	NotifType.PHANTOM, {'app_name': 'Test Device'}	('Ghost Mode Active',

					'Test Device is in ghost mode. Click here to turn it off.')
			TC-6: BASELINE notification	NotifType.BASELINE, {'app_name': 'Test Device'}	('Using Too Much', 'Test Device used more than it should today. Try to use it less.')

Meross Test Cases:

Table 18: *Meross Test Cases*

Test	Input Space	Output Space	Test case	Input	Expected Output
test_inner_login() test_ login()	Email ∈ X Password ∈ X	Successful/Failed Login	**TC-1:** Valid email, valid password	Valid email, valid password	Successful login
			TC-2: TC-2: Valid email, invalid password	Valid email, invalid password	Login failure
			TC-3: Invalid email, valid password	Invalid email, valid password	Login failure
			TC-4: Empty email and password	Empty email, empty password	Login failure
			TC-5: Valid email, missing password	Valid email, missing password	Login failure

test_get_devices()	-	-	TC-1: Retrieve Meross devices	-	List of Meross devices
test_get_id()	Valid device object	Valid Meross device	TC-2: Get device identifier	Valid Meross device	Device identifier
test_get_info()	Valid device object	Valid Meross device	TC-3: Get device information	Valid Meross device	Device information
test_daily_update_creds()	Current timestamp	Update Meross Credentials / No Update	TC-4: Daily update triggered	Current timestamp exceeds prev + day	Update Meross Credentials
	Current timestamp within the same day as prev	Update Meross Credentials / No Update	TC-5: Daily update not triggered	Current timestamp within the same day as prev	No Update

Tuya Test Cases:

Table 19: *Tuya Test Cases*

Test	Input Space	Test case	Input	Expected Output
test_login()	API_KEY, API_SECRET ∈ str	TC-1: Successful Login	API_KEY: Valid strings API_SECRET: Valid strings Dev1: Valid device ID	True
test_get_devices()	API_KEY, API_SECRET ∈ str	TC-2: Get Tuya Devices	-	List of Tuya devices from the Tuya Cloud
test_get_id()	API_KEY, API_SECRET ∈ str	TC-3: Get Tuya Device ID	dev	Device identifier (string)
test_get_info()	API_KEY, API_SECRET ∈ str	TC-4: Get Tuya Device Information	dev	Tuple: (on_off (bool), connection_status

				(bool), power (float))

Checker Test Cases:

Table 20: *Checker Test Cases*

Test	Input Space	Output Space	Test case	Input	Expected Output
test_check_goal()	goal ∈ float, cur_energy ∈ float	[True, False]	TC-1: Goal reached	goal: 100, cur_energy: 150	True
			TC-2: Goal not reached	goal: 200, cur_energy: 150	False
			TC-3: Goal not set	goal: 0, cur_energy: 150	False
test_check_peak()	status ∈ bool, e_type ∈ str, shiftable ∈ list	[True, False]	TC-4: Peak hour, shiftable	status: True, e_type: "PHANTOM", shiftable: ["SHIFTABLE", "PHANTOM"]	True
			TC-5: Peak hour, not shiftable	status: True, e_type: "SHARED", shiftable: ["SHIFTABLE", "PHANTOM"]	False
			TC-6: Not peak hour, shiftable	status: True, e_type: "PHANTOM", shiftable: ["SHIFTABLE", "PHANTOM"]	False
			TC-7: Not peak hour, not shiftable	status: True, e_type: "SHARED", shiftable: ["SHIFTABLE", "PHANTOM"]	False

test_check_phantom()	model ∈ object, power ∈ dict, status ∈ bool	[True, False]	TC-8: Phantom mode, valid model	model: MockModel, power: {"power": 50}, status: True	True
			TC-9: Phantom mode, invalid model	model: None, power: {"power": 50}, status: True	False
			TC-10: Not phantom mode, valid model	model: MockModel, power: {"power": 50}, status: False	False
			TC-11: Not phantom mode, invalid model	model: None, power: {"power": 50}, status: False	False
test_check_baseline()	energy ∈ float, baseline ∈ float	[True, False]	TC-12: Energy exceeds baseline	energy: 150, baseline: 100	True
			TC-13: Energy equals baseline	energy: 100, baseline: 100	False
			TC-14: Energy is below baseline	energy: 50, baseline: 100	False
			TC-15: Baseline is zero	energy: 50, baseline: 0	True
			TC-16: Baseline is negative	energy: 50, baseline: -50	False

Scheduler Test Cases:

Table 21: *Scheduler Test Cases*

Test	Input Space	Output Space	Test case	Input	Expected Output
test_run()	-	-	TC-2: Run Scheduler	None	Master_job created
test_main()	-	-	TC-3: Main Function	None	Server is running (Scheduler started)

Collector Test Cases:

Table 22: *Collector Test Cases*

Test	Input Space	Output Space	Test Case	Input	Expected Output
test_get_appliances()	-	{Id, name, energy} Id∈ String Name∈ String Energy∈ float	TC-1: Get appliances list	-	Appliances list including: appliance, id, name, and energy.
test_to_energy()	Prev_energy∈ float Power∈float output∈float	Updated energy consumption.	TC-2: Calculate energy consumption with valid inputs	Previous energy: 3, Current power: None	Updated energy: 3
			TC-3: Calculate energy consumption with zero inputs	Previous energy: 0, Current power: 0	Updated energy: 0
			TC-4: Calculate energy consumption with non-zero inputs	Previous energy: 4.1, Current power: 0	Updated energy: 4.1
			TC-5: Calculate energy consumption with non-zero inputs and power	Previous energy: 0.71, Current power: 20	Updated energy: 2131 / 3000
test_check_disconnected()	connection status∈ boolean count∈ int flags before∈ [boolean] flags after∈boolean	Check disconnection status and flag update.	TC-6: Check disconnected device with initial flag set to False	Connection status: False Count: 1 Flags before: [False], Flag after: False	Notify once and set flag to False
			TC-7: Check disconnected device with initial flag set to True	Connection status: False, Count: 1, Flags before: [True, False], Flag after: False	Notify once and set flag to False
			TC-8: Check connected device with initial flag set to False	Connection status: True, Count: 1, Flags before: [False], Flag after: True	Set flag to True
			TC-9: Check disconnected device	Connection status: False,	Notify once and

			with multiple occurrences	Count: 2, Flags before: [True, True], Flag after: False	set flag to False
test_notify_disconnected()	List of disconnected devices ∈list	Notifications and updated flags.	TC-10: Notify disconnected devices	-	Notify users about disconnected devices and update flags accordingly.
test_get_doc_updates()	-	Document updates and appliance status updates.	TC-11: Get document and appliance updates	-	Document updates and appliance status updates
test_save_data()	Updates	Saved data in the database.	TC-12: Save data to the database	-	Saved data in the database with appropriate timestamp and user ID.
test_run()	-	FCM notifications and flag updates.	TC-13: Run with failed login, not notified before	Failed login, Not notified before	Notify user and set notified flag to True
			TC-14: Run with failed login, already notified	Failed login, and notified flag set	Do not notify again
			TC-15: Run with successful login, not notified before	Successful login, notified flag not set	Do not notify, notified flag remains False

Mongo Test Cases:

Table 23: *Mongo Test Cases*

Test	Input Space	Output Space	Test case	Input	Expected Output
test_get_doc()	-	dict or None	TC-1: Retrieve a Single Document	"users", {"username": "john"}, {"password": 0}, [("timestamp", pymongo.DESCENDING)]	Document or None
test_get_docs()	-	[Document List]	TC-2: Retrieve Multiple Documents	"posts", {"author": "Alice"}, {"title": 1, "content": 1}, [("timestamp", pymongo.ASCENDING)]	List of expected documents or None
test_insert_doc()	-	None	TC-3: Insert a Single Document	"products", {"name": "Laptop", "price": 1200, "brand": "Dell"}	None
test_insert_docs()	-	None	TC-4: Insert Multiple Documents	"products", [{"name": "Phone", "price": 800, "brand": "Samsung"}, {"name": "Tablet", "price": 500, "brand": "Apple"}]	None
test_update_appliances()	-	None	TC-5: Update Appliances	"users", "123", [{"456", {"status": "on", "power": 50}}, {"789", {"status": "off", "power": 30}}]	None
test_update()	-	None	TC-6: Update a Field in a Document	"orders", "987", "status", "shipped", [{"quantity": {"$gt": 10}}, {"quantity": {"$lte": 10}}]	None

Updater Test Cases:

Table 24: *Updater Test Cases*

Test	Input Space	Output Space	Test case	Input	Expected Output
test_get_powers()	Updater instance	DataFrame containing power data	TC-1: Retrieve Powers	updater_instance, mocked_db	DataFrame with powers data for a specific timestamp
test_yesterday_powers()	Updater instance	DataFrame containing power data	TC-2: Filter Yesterday Powers	updater_instance	DataFrame with filtered powers data from yesterday
test_yesterday_energys()	Updater instance	Dictionary containing appliance energies	TC-3: Yesterday Energys	updater_instance	Dictionary with energies for non-deleted appliances
test_update_energy()	Updater instance	Updated energy values	TC-4: Update Energy	updater_instance, date	Updated energy values based on specified date
test_dump_model()	Updater instance with mocked model	Model dumped to file system	TC-5: Dump Model	updater_instance, model	Model dumped to the file system
test_apply_cluster()	Updater instance with mocked methods	Cluster applied to powers	TC-6: Apply Cluster	updater_instance, monkeypatch	Cluster applied to powers with model dumped
test_apply_forecast()	Updater instance with mocked methods	Forecast applied to powers	TC-7: Apply Forecast	updater_instance, monkeypatch	Forecast applied to powers

					with model dumped
test_run()	Updater instance with mocked methods	Energys and models updated	TC-8: Run Updater	updater_instance	Energys and models updated based on yesterday 's data

Integration Testing for Tracker Server:

The architecture of the tracker server is built upon distinct components, ensuring that classes operate independently, and the main classes remain isolated from each other. Specifically, the master, collector, updater, and checker classes function autonomously, without the need for direct collaboration. Instead of emphasizing integration among the main tasks, our focus is on conducting integration tests that evaluate the seamless interaction between the primary tasks and the external dependencies.

As a result, we execute four dedicated integration tests, each focused on assessing a task's interaction with external dependencies. These tests involve running tasks with actual components, incorporating a live database, FCM, and cloud infrastructure. This meticulous approach ensures the harmonious functionality of all components within the system.

Usability Testing:

Test Objectives:

- Evaluate the effectiveness of the usage of PowerEye application by setting some test scenarios to be tested by the users.
- **Key questions:**
    - How easily can the users navigate through the application?
    - Are the users able to see their energy consumption analysis?
    - Are the users able to set a cost goal, modify it , and remove it?
    - Are the users able to add new appliances to their account?
    - Are the users able to create a new room?
    - Are the users able to add new appliances to an existing room?
    - Are the users able to edit appliance/room name?

- Are the users able to delete appliance/room?

Participants:

During the execution of this test, we have selected a sample of 5 participants, carefully chosen to represent our target audience and provide valuable insights into the usability of the product.

User characteristics:

- Male/Female
- Age > 18
- Have interest in Home Energy Monitoring Systems (HEMS)

Test procedure:

- **Pre-test:** We will commence the usability session by extending a warm welcome to the participant and providing a comprehensive introduction to the purpose and objectives of the usability testing. A concise overview of the application's features and functionalities will follow, accompanied by the confirmation of the participant's willingness to partake in the testing process.
- **During-test:** During the testing, meticulous task instructions will be provided, prompting participants to articulate their thoughts aloud while I record their on-screen interactions. As a moderator, we will offer guidance when necessary, ensuring participants remain uninfluenced in their decision-making processes.
- **Post-test:** We will engage participants in a constructive discussion to elicit feedback, capturing their overall impressions, likes, and dislikes. Probing questions will be employed to delve into specific issues, clarifying any ambiguities and expressing gratitude for their invaluable contributions.

Test Result:

- As our application needs some Meross credentials to create a new account, we'll skip the registration process and continue with login immediately.
- We won't let the user delete the account, since this is a test account that will be used multiple times.

Table 25: *Usability Test*

Id: 1 - **Participant:** R.B **Age:** 23 **Gender:** Female **Id:** 2 - **Participant:** A **Age:** 32 **Gender:** Male **Id:** 3 - **Participant:**L - **Age:** 20 - **Gender:** Female **Id:** 4 - **Participant:** J.N **Age:** 22 - **Gender:** Female

Id: 5 - Participant: J.A Age: 23 Gender: Female											
Test Scenario	**Time expected**	**Time taken**					**Test Result**				
		1	**2**	**3**	**4**	**5**	**1**	**2**	**3**	**4**	**5**
A user enters an incorrect password and receives an error message.	10 - 15 sec.	10 sec.	10 sec.	15 sec.	14 sec.	12 sec.	Pass	Pass	Pass	Pass	Pass
A user enters the correct login credentials and successfully logs into their account.	10- 20 sec.	12 sec.	20 sec.	14 sec.	14 sec.	15 sec.	Pass	Pass	Pass	Pass	Pass
A user navigates to the user information section, successfully views their existing information, and proceeds to edit and save it.	15 - 20 sec.	18 sec.	12 sec.	18 sec.	17 sec.	15 sec.	Pass	Pass	Pass	Pass	Pass
A user uploads a new profile picture successfully, and it reflects the changes on their profile.	30 - 35 sec.	33 sec.	15 sec.	27 sec.	35 sec.	32 sec.	Pass	Pass	Pass	Pass	Pass
A user sets a cost goal for their energy consumption successfully, providing the desired target cost value.	10 - 15 sec.	10 sec.	10 sec.	14 sec.	10 sec.	13 sec.	Pass	Pass	Pass	Pass	Pass

A user modifies an existing cost goal, adjusting the target value to a new value.	10 - 15 sec.	13 sec.	8 sec.	15 sec.	13 sec.	12 sec.	Pass	Pass	Pass	Pass	Pass
A user initiates the deletion of a cost goal but cancels the action before it is completed.	5 - 10 sec.	7 sec.	5 sec.	5 sec.	5 sec.	5 sec.	Pass	Pass	Pass	Pass	Pass
A user successfully deletes a cost goal, removing it from their goal list.	5 - 10 sec.	9 sec.	5 sec.	5 sec.	5 secs.	5 sec.	Pass	Pass	Pass	Pass	Pass
A user successfully adds a new appliance to their profile, providing the necessary details and saving it.	25 - 30 sec.	29 sec.	27 sec.	26 sec.	25 sec.	27 sec.	Pass	Pass	Pass	Pass	Pass
A user attempts to delete an appliance but cancels the deletion action before it is completed.	25 - 30 sec.	24 sec.	30 sec.	30 sec.	30 sec	25 sec.	Pass	Pass	Pass	Pass	Pass
A user tries to change the name of an appliance but cancels the action before it is completed.	10 -15 sec.	10 sec.	12 sec.	12 sec.	15 sec.	10 sec.	Pass	Pass	Pass	Pass	Pass

A user successfully changes the name of an appliance, providing a new desired name.	40 - 60 sec.	40 sec.	30 sec.	22 sec.	52 sec.	55 sec.	Pass	Pass	Pass	Pass	Pass
A user successfully deletes an appliance.	30 - 40 sec.	35 sec.	20 sec.	36 sec.	40 sec.	35 sec.	Pass	Pass	Pass	Pass	Pass
A user creates a new room successfully, specifying the room details and saving it.	15 - 20 sec.	19 sec.	15 sec.	17 sec.	15 sec.	16 sec.	Pass	Pass	Pass	Pass	Pass
A user adds appliances to an existing room, ensuring they are properly linked and displayed within the room.	40 - 45 sec.	43 sec.	40 sec.	40 sec.	44 sec.	41 sec.	Pass	Pass	Pass	Pass	Pass
A user attempts to change the name of a room but cancels the action before it is completed.	20 - 25 sec.	10 sec.	24 sec.	19 sec.	25 sec.	25 sec.	Pass	Pass	Pass	Pass	Pass
A user successfully changes the name of a room, providing a new desired name.	30 - 50 sec.	33 sec.	20 sec.	37 sec.	61 sec.	49 sec	Pass	Pass	Pass	Fail	Pass
A user successfully	25 - 35 sec.	31 sec.	20 sec.	28 sec.	32 sec.	32 sec.	Pass	Pass	Pass	Pass	Pass

Test findings:

Some questions will be asked to the user after the test done in order to know how was his experience:

- How was your experience using PowerEye?
- What did you like most?
- What did you dislike most?
- Do you have any recommendations we could change in order to make the app better?

User 1: R.B findings:

Upon successful experience R.B has gone through, where all the test scenarios have been passed. R.B found that PowerEye has a good visual design that enhances the user experience, such as the clear icons used. The app was smooth and the navigation between different pages was easy. She liked the analysis pages most, and she said they are so insightful. As a recommendation, R.B suggested that doing more interactivity on the app will make it more user-friendly, especially the buttons on the pop-up windows.

User 2: A findings:

The participants had a great and wonderful experience using the PowerEye system. He found it easy to identify how the features work, and the time it took was as expected, if not a bit less. When asked about his experience, User A expressed satisfaction, highlighting the goal achievement feature as his favorite, allowing him to specify and edit specific goals. However, he mentioned a dislike for the green color. Despite this, User A had no specific recommendations for improvement, stating that the app is good enough.

User 3: L findings:

The participant had an overall positive experience using PowerEye and appreciated its creative concept. The features the participant liked the most was the ability to view the analysis of my energy consumption and setting cost goals. On the other hand, one aspect that the participant found slightly inconvenient was the time it took to process the information. Also, the

participant recommend focusing on optimizing the app's performance to make it faster in processing the data.

User 4: J.N findings:

After the user acceptance test, the participant lauded PowerEye for its user-friendly interface, expressing ease of use. She particularly valued the ability to add multiple appliances within a single room, streamlining device management. However, the user has a preference for dark mode and expressed a desire for this feature. Implementing a dark mode option could enhance her user satisfaction and accommodate diverse preferences, aligning with PowerEye's commitment to delivering a customizable and enjoyable user experience.

User 5: J.A findings:

The user acceptance test for PowerEye yielded highly positive feedback from the user, reflecting an overall enjoyable and engaging experience. The user expressed great satisfaction with the application, particularly enjoying the interactive feature of remotely turning appliances on and off. Notably, there were no identified dislikes, highlighting the user's appreciation for the application's seamless functionality. The user's suggestion to integrate a bill payment option within the app reflects a valuable insight for potential future enhancements, aligning with the user's desire for increased convenience. This user acceptance test indicates a successful implementation of features and a high level of user satisfaction, with the valuable recommendation serving as a potential avenue for further improvement and user convenience.

Discussion:

The findings from the usability test of PowerEye indicate an overall positive response from the participants. Users appreciated the visual design, smooth navigation, and insightful analysis pages, highlighting the user-friendly nature of the app. The ability to set goals and manage appliances received praise, while some users expressed preferences for additional features such as dark mode and bill payment integration. The findings suggest that PowerEye has successfully delivered a satisfying user experience, with valuable recommendations for further enhancement to accommodate diverse user preferences and optimize performance. These findings validate the effectiveness of PowerEye in facilitating energy management and user engagement, positioning it as a promising solution in the market.

You can find participants records: Usability Test Videos

Software Testing Process:

First Iteration:

The first iteration's main goal is to plan for the testing phase and lay the groundwork. This involved defining the test objectives, levels, types, and Technique. To ensure that testing efforts are concentrated on crucial areas, the features that should be tested and those that shouldn't be tested are identified. This determines the test scope. Test logistics are also taken into account, including what has to be tested, who will conduct the testing, and when the testing will take place. In order to make well-informed judgments on the direction and completion of testing, the first iteration also includes creating test criteria, such as suspension and exit criteria. Lastly, the environments, test resources, and schedule estimation required to support testing operations are determined.

Second Iteration:

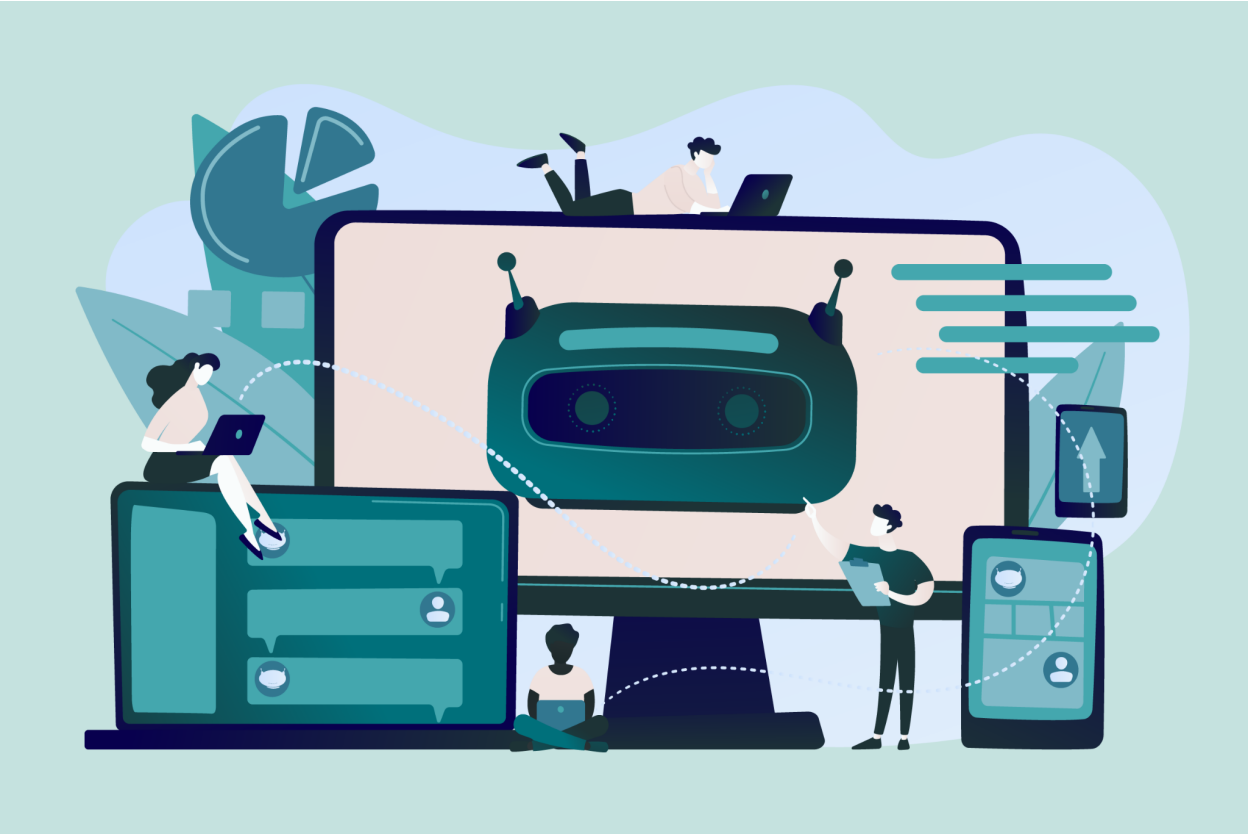
Building on the groundwork laid in the first iteration, the testing process is carried out in the second iteration. Comprehensive testing activities are performed, including unit testing, integration testing, and usability testing for the CS team. Other forms of testing can be visible in the SE team's documentation.

Conclusion:

Throughout the testing phase of the PowerEye project, several notable outcomes and insights emerged. The comprehensive testing activities, including unit testing, integration testing, and usability testing, resulted in achieving the required goals. Unit and integration testing successfully resolved bugs, validated PowerEye against defined criteria, and ensured seamless performance. Usability testing contributed to enhancing the overall user experience. The meticulous documentation of test cases and scenarios facilitated a systematic approach to testing, ensuring clarity and precision. However, challenges were encountered in the background of testing and documentation. Collaborative efforts between the CS and SE teams effectively addressed these challenges. Lessons learned include the importance of continuous collaboration between both teams, the value of participant feedback in usability testing, and the need for flexibility in addressing unexpected challenges during the testing process. For more details and a comprehensive set of test cases, please refer to the **Tracker Server Tests** on *GitHub*. These insights are instrumental in refining the PowerEye project's software development processes and ensuring the delivery of a high-quality, user-centric application.

DEPLOYMENT

Details about how the project is deployed and made accessible to end-users or stakeholders



Deployment Overview:

The deployment of our system adopts a hybrid approach, strategically leveraging both cloud and on-premises environments to optimize performance, scalability, and resource utilization. The web server, a critical component of our system, finds its home on the AWS cloud, harnessing the power and flexibility of cloud computing to ensure high availability and efficient resource allocation. On the other hand, the tracker server is deployed on Single Board Computer (SBC) Orange Pi, providing a cost-effective and compact solution for real-time tracking functionalities. Our database resides in the AWS cloud, benefitting from the scalability and managed services offered by cloud databases. Firebase, a key element in our ecosystem, is deployed on Google Cloud, taking advantage of its specialized features for real-time data synchronization and secure communication. Lastly, the React Native application, designed to run seamlessly on both Android and iOS devices, is deployed across a diverse array of devices, ensuring widespread accessibility and user engagement. This hybrid deployment approach enables us to harness the strengths of each platform, creating a robust and flexible system that meets the specific needs of our project.

Deployment Environment & Configurations:

Database:

In deploying PowerEye databases on Amazon servers through MongoDB Atlas, we specifically opt for Amazon Web Services (AWS) in the Bahrain region (me-south-1). Leveraging the M0 Sandbox cluster tier with shared RAM and 512 MB of storage, we ensure a cost-effective solution suitable for initial development and testing. The choice of AWS in the Bahrain region aligns with our strategy to provide low-latency access for users in the Middle East, enhancing the overall performance of our database operations. MongoDB Atlas streamlines the deployment process by managing administrative tasks, including backups, monitoring, and security configurations. This deployment on AWS allows us the flexibility to scale our database resources effortlessly as data volume requirements evolve. Combining MongoDB Atlas with AWS, we achieve a robust, scalable, and regionally optimized database solution.

Firebase:

Deploying the PowerEye project on Firebase Cloud Messaging (FCM) involves a setup for seamless communication across iOS, Android, and our server. Under the management of the *saraalshagawi@gmail.com* account, we initiated two applications—one for iOS and one for

Android—within the Firebase project. For the Android app, we configured SHA certificate fingerprints to establish secure communication with Android devices. Simultaneously, we implemented APN (Apple Push Notification) authentication keys for the iOS app, ensuring a secure connection with Apple devices. Additionally, a server key was generated to enable our server to communicate effectively with the Firebase infrastructure. This strategic setup on FCM allows PowerEye to leverage Firebase's robust cloud-based messaging services, facilitating efficient and reliable communication across diverse platforms within our project.

Web Server:

Deploying the PowerEye web server on Amazon Web Services (AWS) involved creating a highly efficient and cost-effective setup. Utilizing the AWS Free Tier, we generated an Amazon Machine Image (AMI) tailored for our web server. The AMI is specifically chosen to run on the Ubuntu Server 22.04 LTS operating system. The chosen instance type, t3.micro, belonging to the t3 family, provided 2 vCPUs and 1 GiB of memory, ensuring optimal performance within the free tier constraints. For storage configuration, we opted for 30 GB of General Purpose (SSD) Magnetic storage, aligning with our storage requirements. This deployment on AWS EC2 (Elastic Compute Cloud) with a thoughtfully chosen configuration allows the PowerEye web server to run seamlessly, leveraging the scalability and reliability of AWS services.

Tracker Server:

Deploying the PowerEye Tracker Server on the Orange Pi 5, which is a single board computer (SBC), offers a robust and efficient solution, leveraging the advanced hardware capabilities of the device. Powered by the Rockchip RK3588S, a cutting-edge 8-core 64-bit processor featuring a quad-core A76 and quad-core A55 configuration, the Orange Pi 5 provides substantial computing power. With 8GB of LPDDR4/4X RAM, it ensures smooth and responsive performance. The server operates on the Ubuntu 22.04.2 LTS operating system, offering a stable and well-supported environment. Storage is facilitated through a 32GB microSD card, providing ample space for data storage and server operations. This deployment harnesses the Orange Pi 5's hardware prowess to deliver a reliable and capable PowerEye Tracker Server solution for tracking and managing data effectively.

Mobile Application:

For the mobile application, we utilized React Native with Expo, leveraging Expo for deployment on both iOS and Android platforms. To cater to iOS, we generated an ad hoc provisioning profile with push notifications capabilities, coupled with an Apple Distribution

certificate. All designated devices, including our team's and users', were added to facilitate internal distribution. No specific configurations were required for Android, allowing for broad usage of the build. The successful generation of a preview build marked a significant milestone, allowing the execution of the application on mobile devices without reliance on the Metro server, streamlining the testing and previewing process for our team.

Monitoring and Logging:

In the tracker server, we implemented a comprehensive logging strategy to ensure effective monitoring and troubleshooting capabilities. This approach incorporates both real-time observation through the console and archival records stored in log files, offering a dual-pronged solution for monitoring system behavior. All logs are systematically organized and stored in a dedicated **logs** folder. Each log file is uniquely identified by the timestamp at which it is created, facilitating easy reference and traceability. The logging format adheres to a structured pattern: **"%(asctime)s - %(name)s - %(levelname)s - %(message)s"**. Here, the placeholders convey critical information:

- **asctime:** Represents the timestamp when the log entry was generated.
- **name:** Identifies the specific module responsible for logging the message.
- **levelname:** Indicates the log level, providing insights into the severity of the log entry (e.g., DEBUG, INFO, WARNING, CRITICAL, ERROR).
- **message:** Presents the detailed log message.

This approach offers valuable context and information about the system's state or events. This logging framework serves as a valuable tool for ongoing system monitoring, troubleshooting, and retrospective analysis.

File Systems and Storage:

While our primary data storage relies on databases, certain datasets are stored in the server file systems. This approach is particularly beneficial for handling large-sized data and ensuring rapid access to critical information.

Web Server Uploads:

To efficiently manage user profile images, we store them within the server in the designated **uploads/images** folder. The **uploads** folder serves as a versatile repository for potential future uploads, providing a scalable solution for handling diverse user-generated content.

Tracker Server Machine Learning Models:

Our tracker server incorporates machine learning models categorized into two types: cluster and forecasting. Within the **models_filesystem** directory of the tracker server, three subfolders exist—namely, **cluster_models**, **forecast_models**, and **test_models**,—contain user-specific folders. To prevent conflicts, each user is assigned a dedicated folder named with their unique ID. Inside these user-specific folders, we store pickle files containing the models, parameters, and other variables essential for the machine learning models.

Mobile Application Caching:

Our mobile application employs an effective caching strategy to optimize the retrieval and presentation of essential data. This includes caching for:

- Both user and FCM tokens
- User information
- Data for main pages

This caching mechanism significantly enhances the speed of data retrieval and presentation within the mobile application, contributing to a seamless user experience.

User Guide:

In the realm of mobile applications, a well-crafted User Guide and Frequently Asked Questions (FAQ) document play pivotal roles in enhancing the user experience. A User Guide provides users with a comprehensive resource for understanding the functionalities and navigation of the application, serving as a roadmap to unlock the full potential of the features it offers. On the other hand, an FAQ document anticipates common user queries, offering quick and accessible solutions. These documents collectively empower users by providing clarity, guidance, and troubleshooting tips, ultimately fostering a smoother and more satisfying user journey. To access our User Guide and Frequently Asked Questions Document for our mobile application, please refer to the following links:

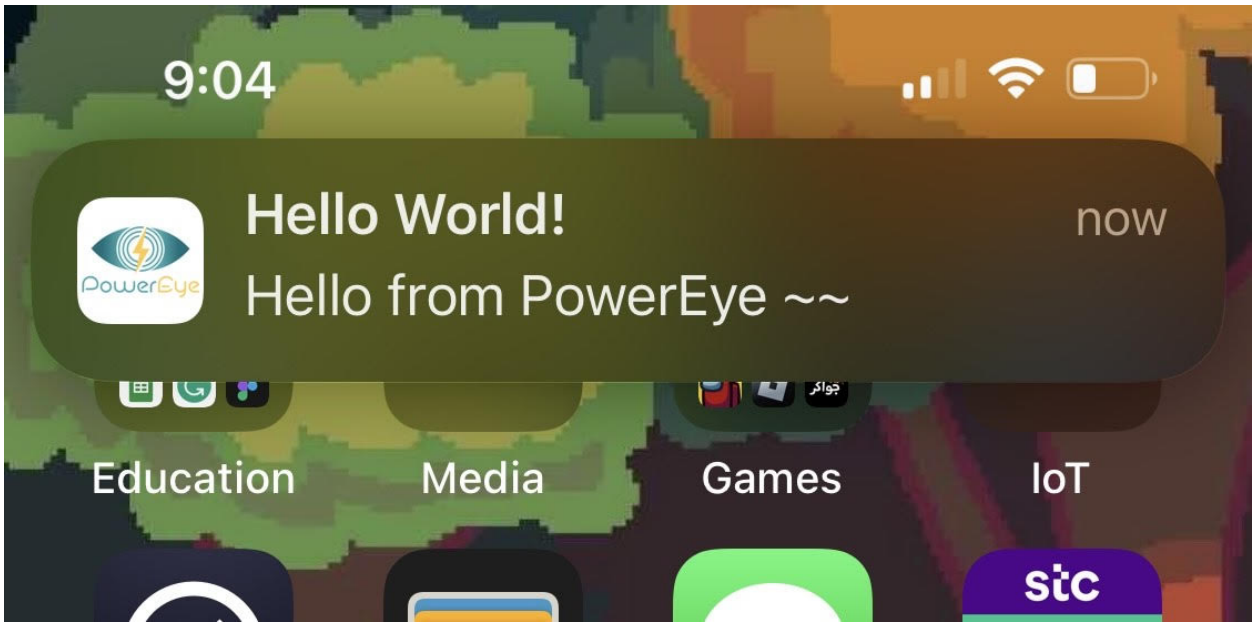
User Guide Document

Frequently Asked Questions Document

Screenshots:

Hello World:

Figure 18: *Hello World from PowerEye!*



Database:

Figure 19: *PowerEye Database on Atlas (Users Collection)*

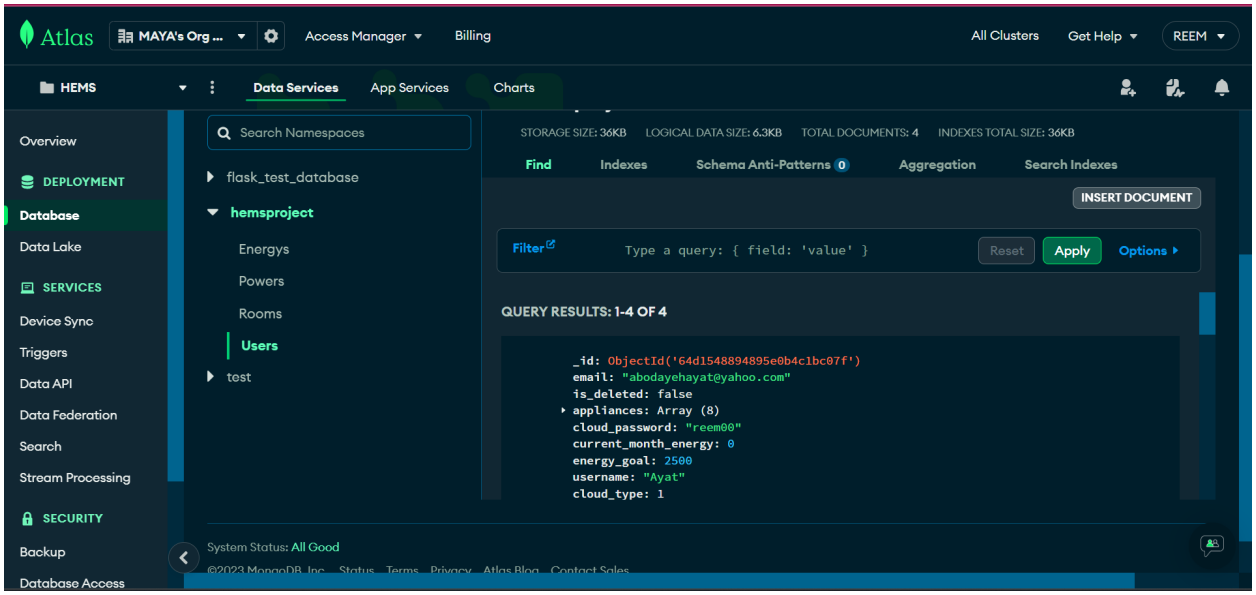
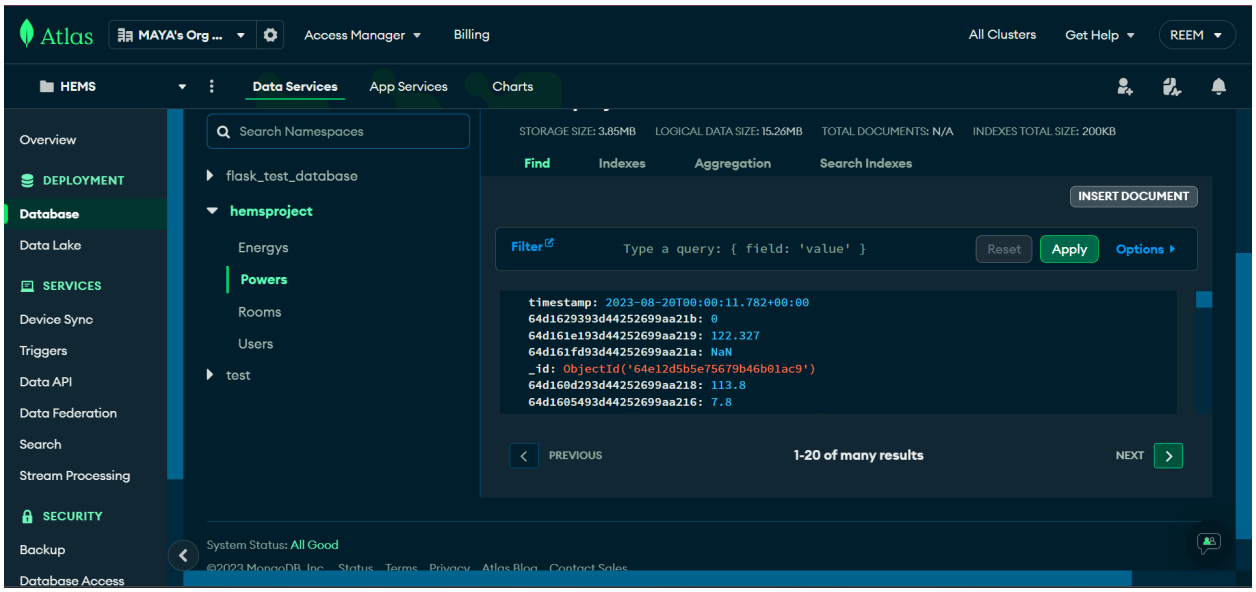


Figure 20: *PowerEye Database on Atlas (Powers Collection)*



Firebase:

Figure 21: *PowerEye Firebase Project Overview*

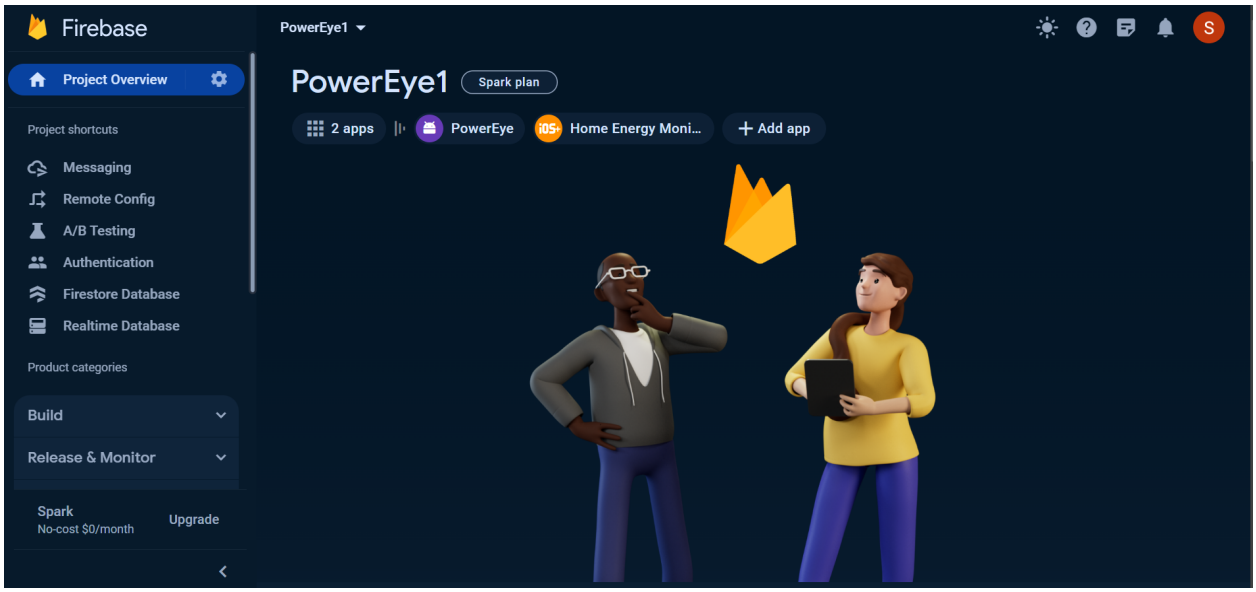
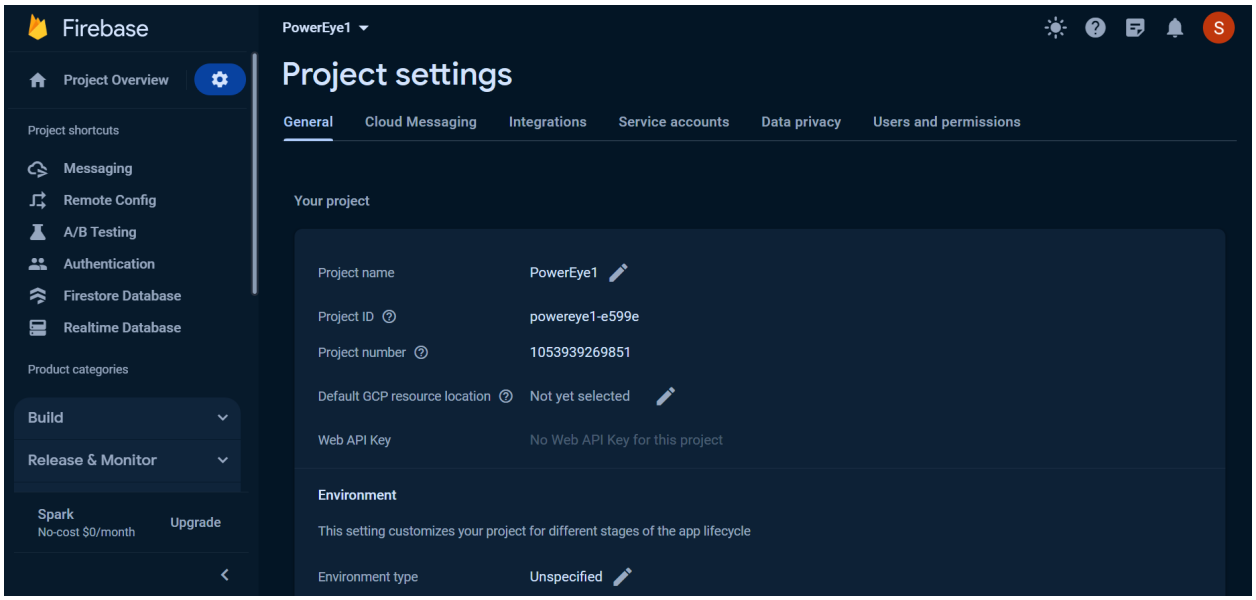


Figure 22: *PowerEye Firebase Project Information*



Web Server:

Figure 23: *PowerEye Web Server AWS Instance Information*

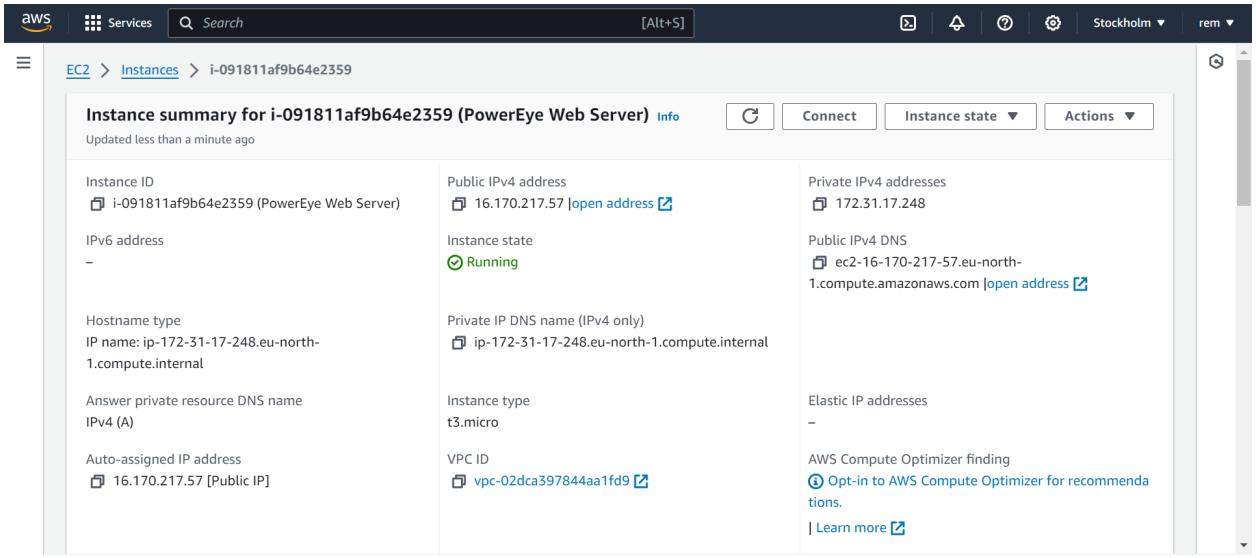
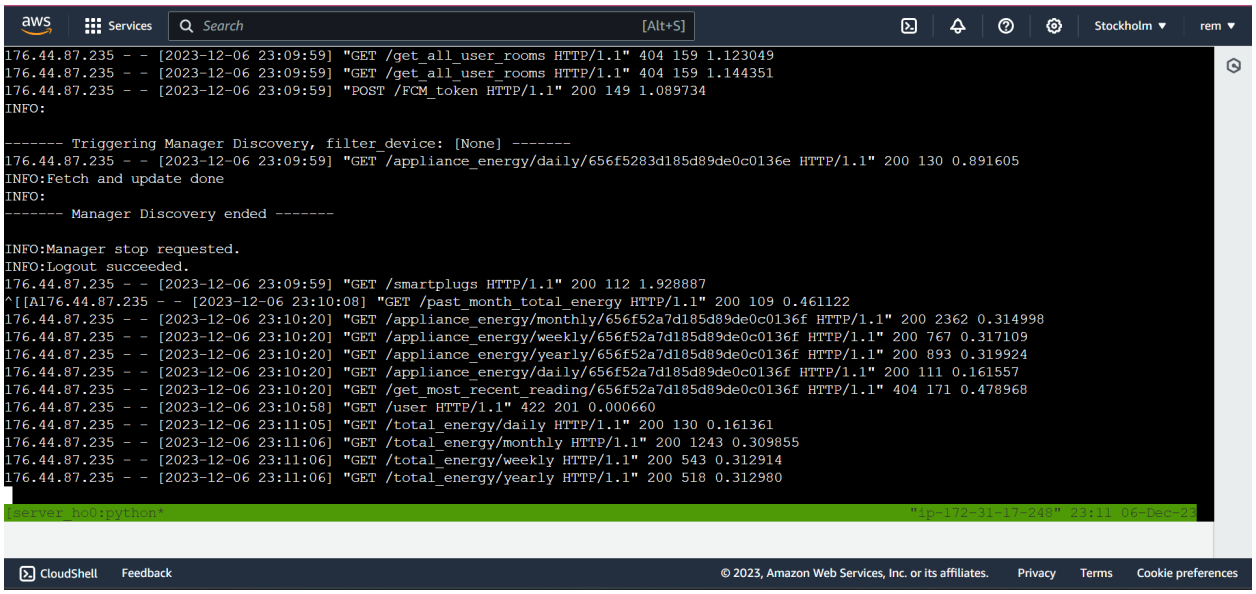


Figure 24: *PowerEye Web Server Console logs*



Tracker Server:

Figure 25: *PowerEye Tracker Server Physical Single Board Computer (Orange pi 5)*

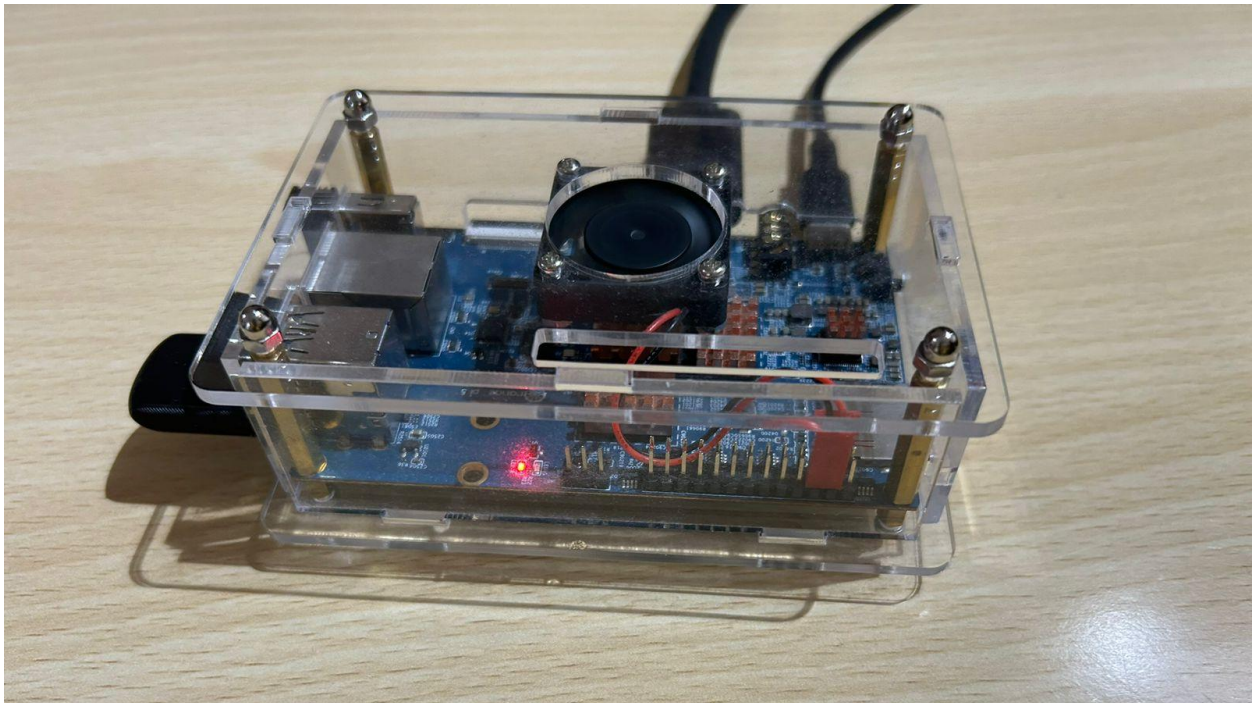
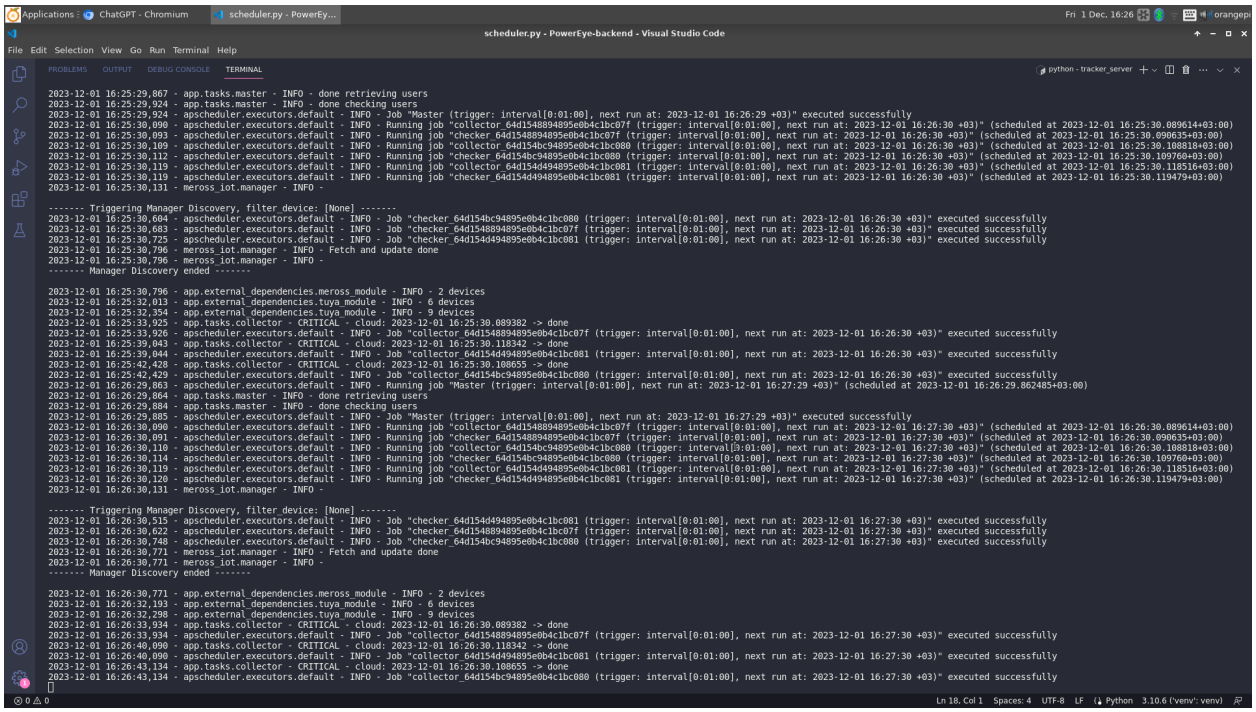


Figure 26: *PowerEye Tracker Server Console Logs*



Mobile Application:

Figure 27: *Mobile Application Interface on an Iphone and a Huawei Phone*

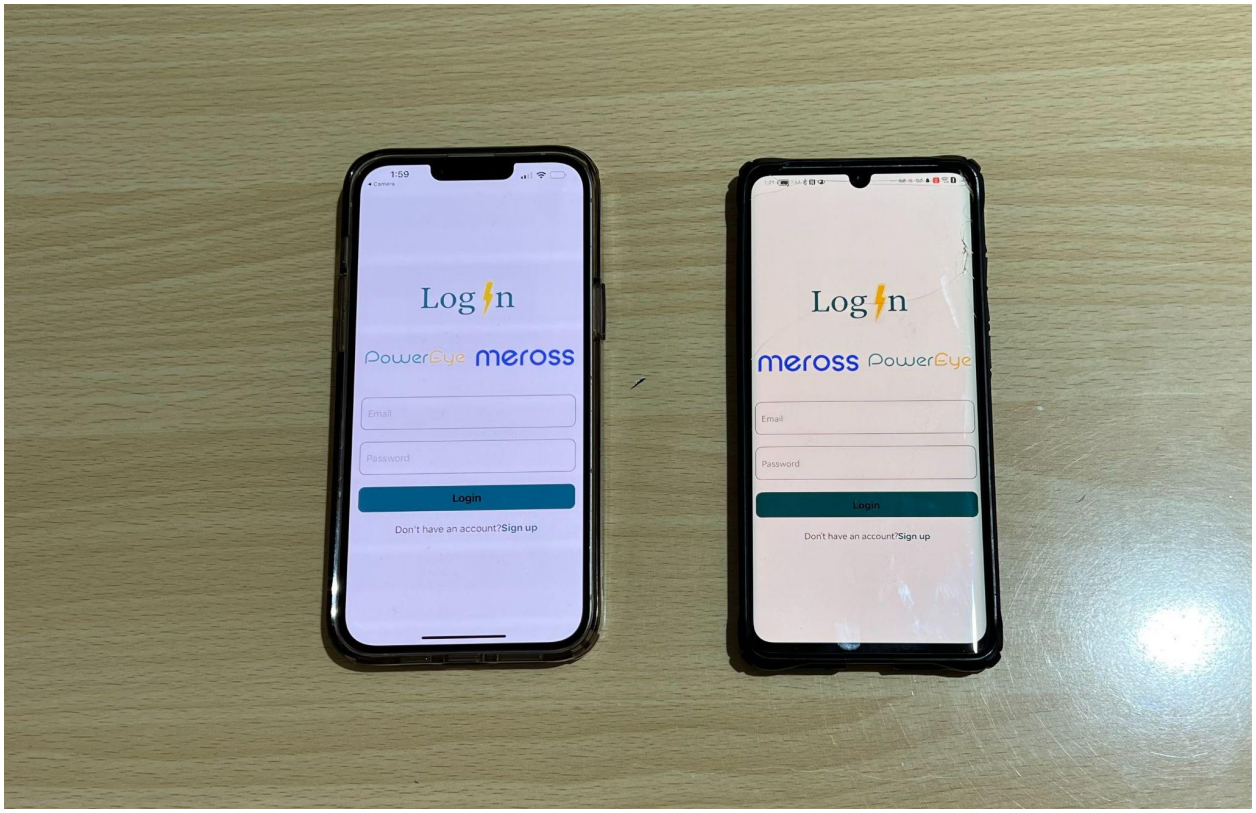
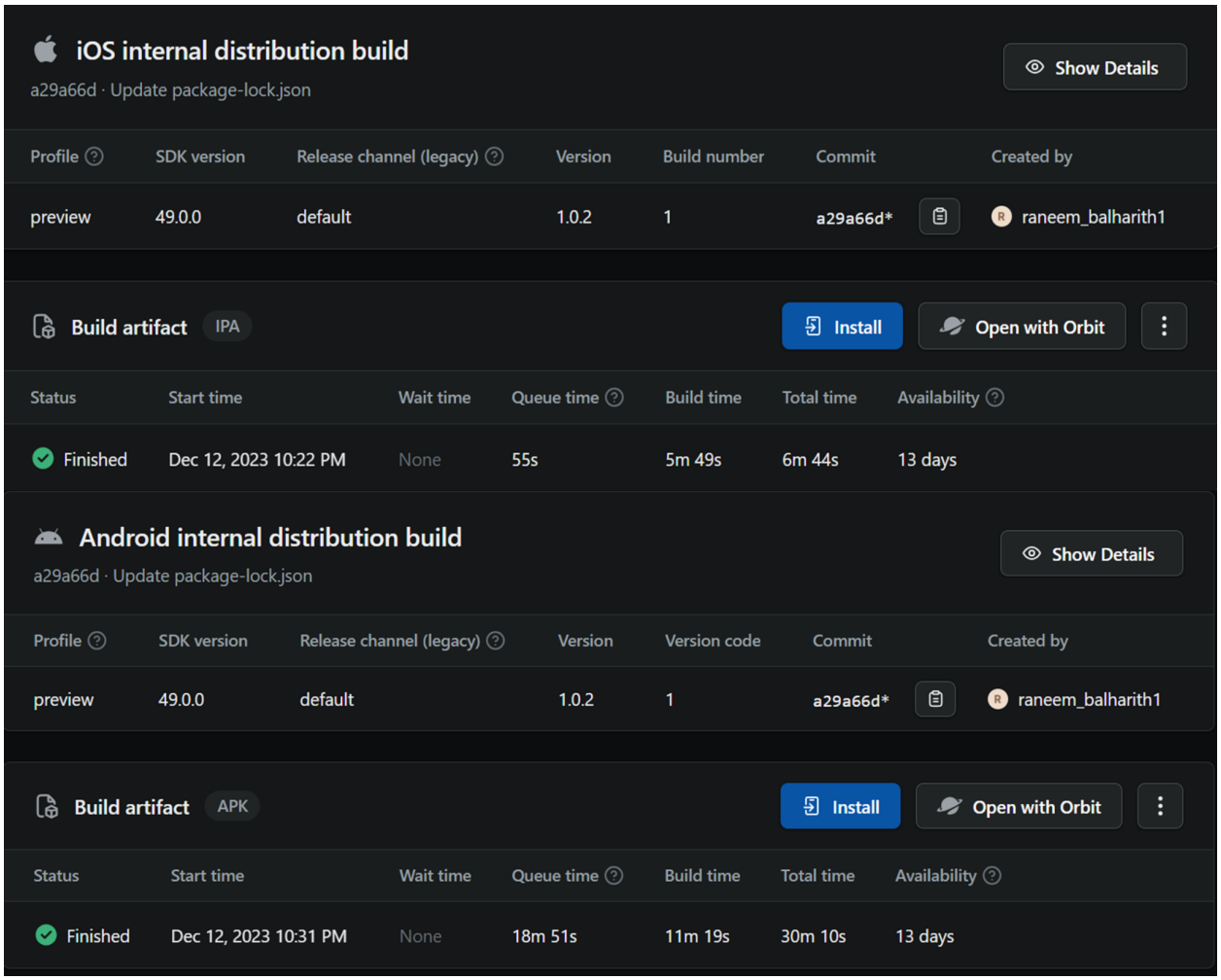


Figure 28: *Expo Preview Build for iOS & Android Mobile Application*



End-to-End System Workflow:

In this section, we depict the workflow of the *'Setting Cost Goal'* use case, extending it to the receipt of the energy goal notification. The demonstration is conducted within the context of the Qater Al Nada Account, highlighting the initial scenario where the goal was not set.

Figure 29: *User Set Goal*

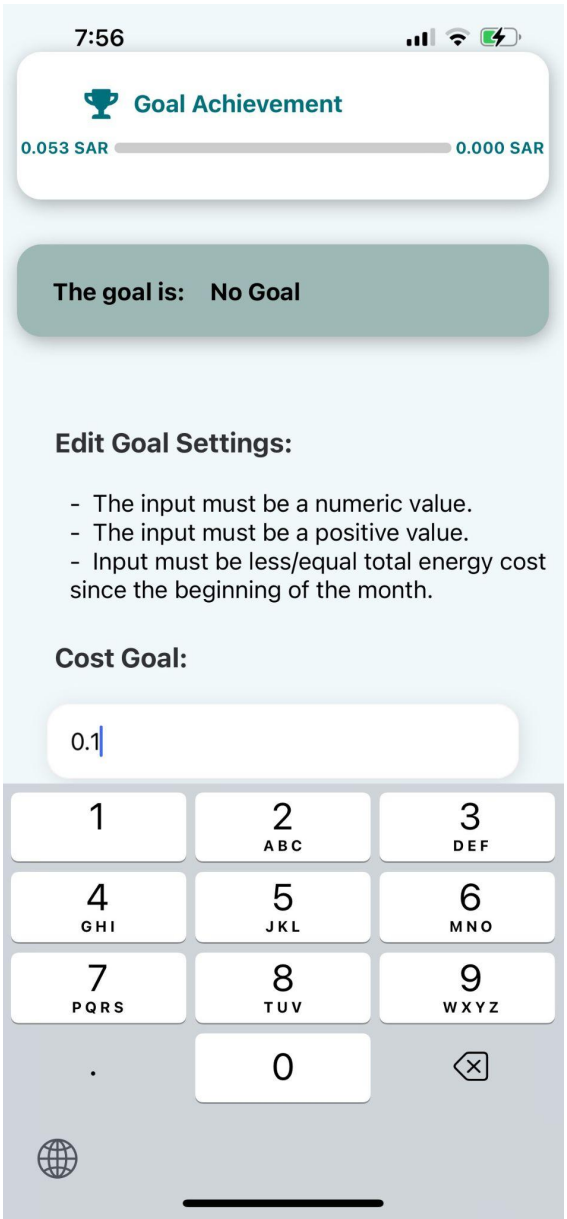


Figure 30: *Goal is Updated on The Application*

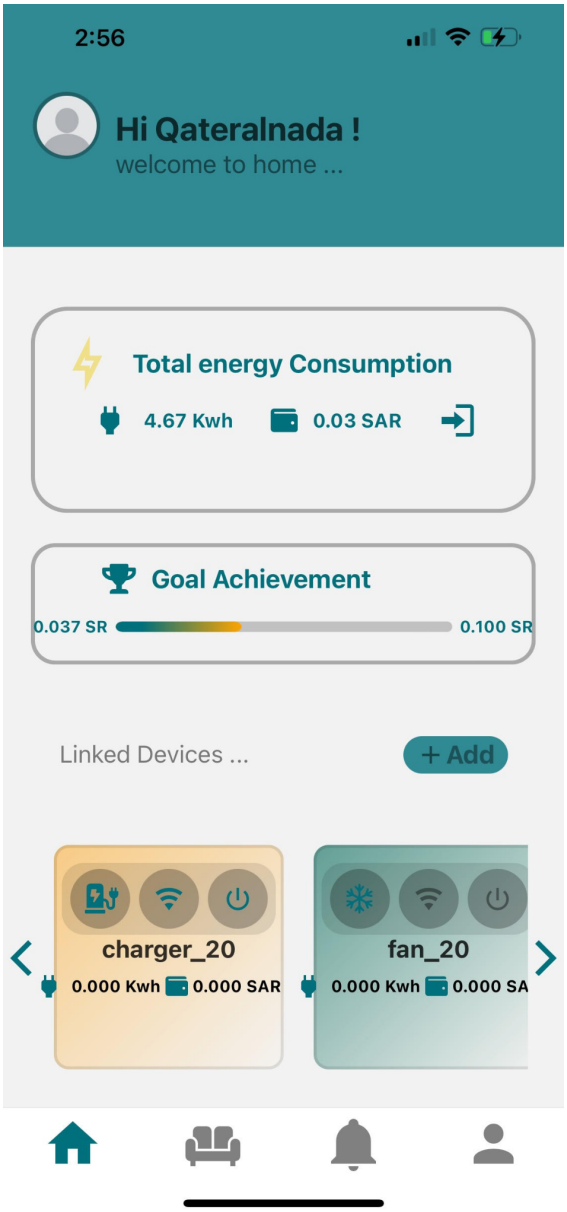


Figure 31: *Web Server Receives The Request*

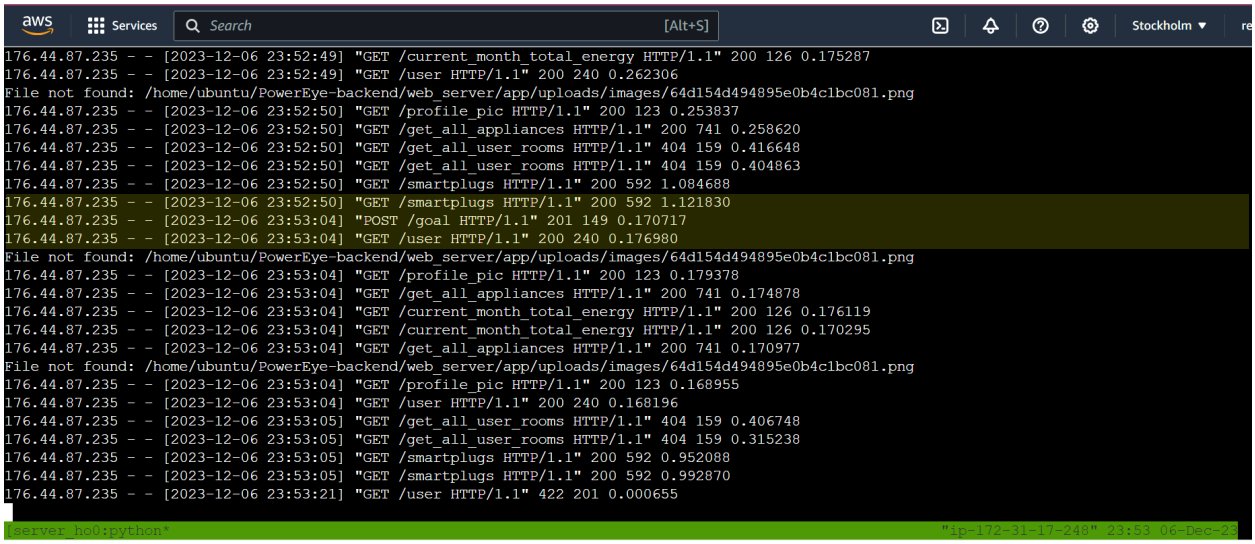


Figure 32: *Goal is Updated on The Database*

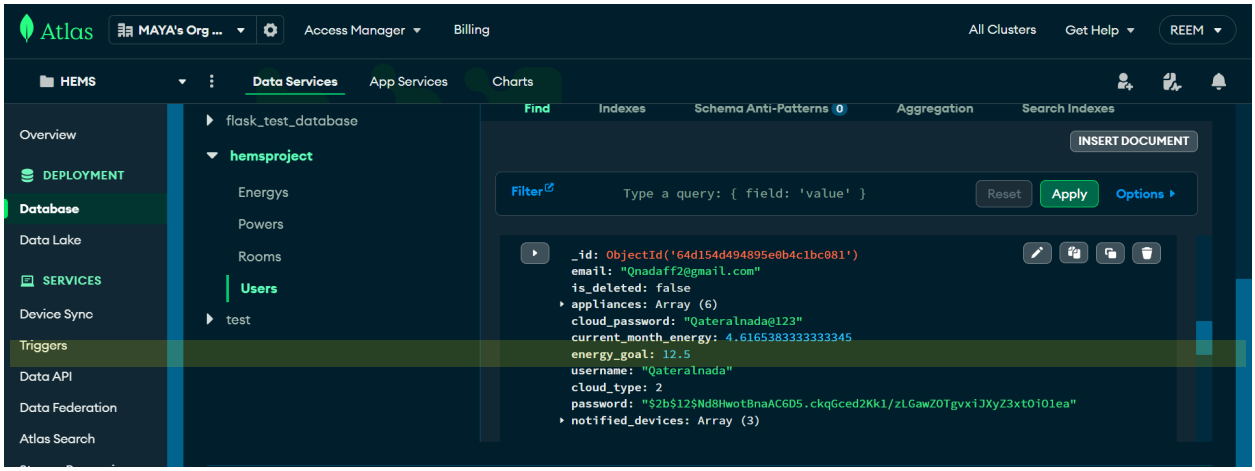


Figure 33: *Tracker Server Detect Energy Goal Percentage*

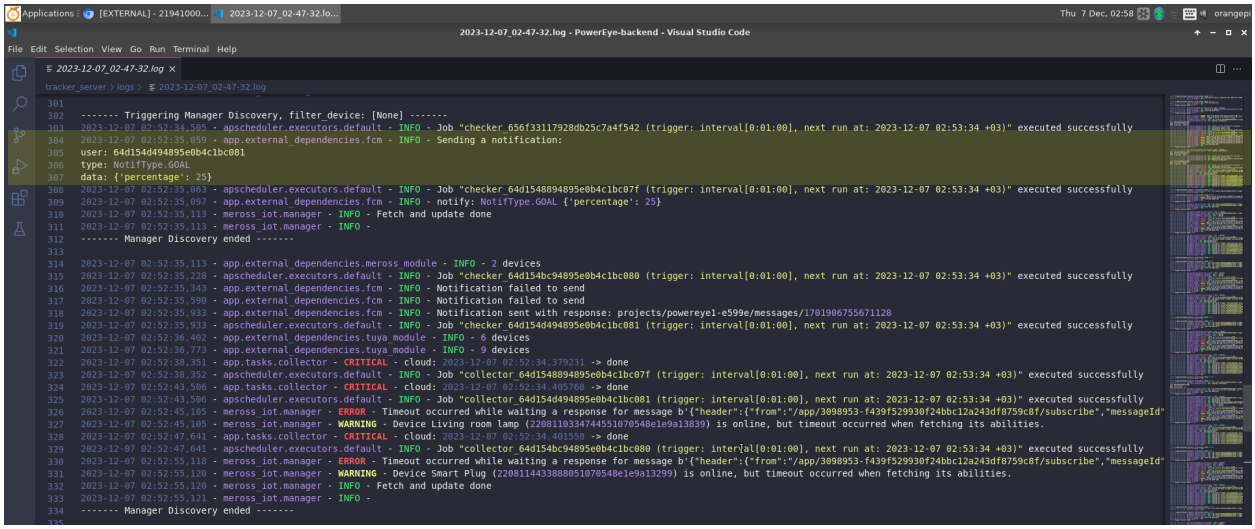
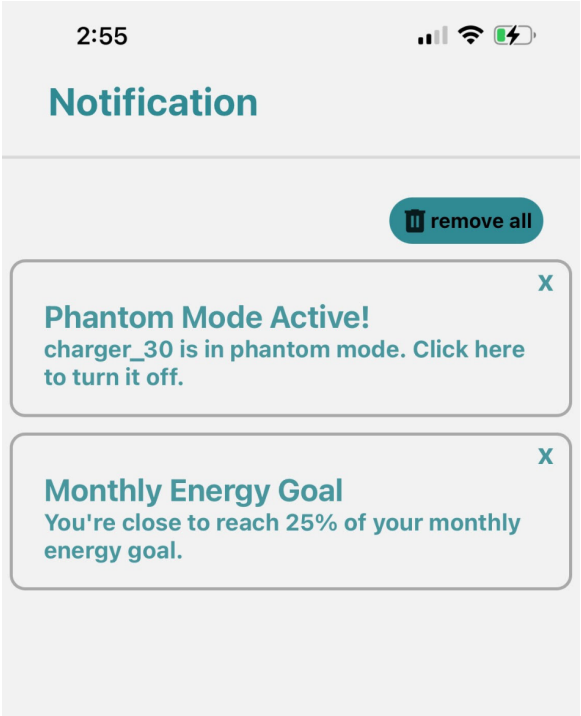


Figure 34: *Notification is Received*



Starting from ***Figure 29***, no goal was set for the Qatar Al Nada account. We subsequently set a goal of 0.1 SAR for the month. In ***Figure 30***, the mobile application displays the updated goal with a progress of 0.037/0.1 SAR. Moving to ***Figure 31***, the Web Server receives a post goal request for updating the goal. ***Figure 32*** illustrates the successful update of the goal in the database for the Qatar Al Nada document in the Users collection. Additionally, we converted the cost goal to an energy goal (0.1 SAR = 12.5 kWh) for simplified calculations on the Tracker Server. In ***Figure 33***, the tracker server detects a 25% completion, signifying that the user has achieved 25% of their energy consumption goal. The server logs the event and sends a notification to the user, as depicted in ***Figure 34***. The mobile app successfully receives the in-app notification.

Furthermore, demo videos showcasing the process of adding a new smart plug are available, demonstrating the integration with the Meross app and our application and other use cases. All videos are accessible in the Demo Videos folder.

EXPERIMENTAL RESULTS

Details about the experiment, discussion of the algorithms, and performance evaluation

Introduction:

In this phase, we will evaluate our system across four key aspects:

- **Algorithm Performance:** How much time is needed for the critical algorithms to run?
- **Data Accuracy:** How accurately does the energy data reflect the real data from their corresponding cloud data?
- **ML Models Accuracy:** How accurate and reliable are the predictions of the two ML models used in the EPR?
- **User Engagement:** How effective is the application from the user's point of view?
- **System Effectiveness:** Does the system meet the main goal of PowerEye, which is to help users optimize their energy consumption?

Answers to all these questions will be provided in the following sections.

Methodology and Evaluation Metrics:

Each aspect is evaluated using specific metrics:

- **Algorithm Performance:** The EPR's main algorithms are concentrated in the check functions within the checker class, impacting the system's overall performance due to their execution at short intervals. We will conduct an asymptotic analysis using big O notation and measure the execution time on a physical device.
- **Data Accuracy:** Data retrieval from the smart plugs cloud (Meross + Tuya) is compared to our data since the Tracker Server being deployed. Metrics such as RMSE or MAE will illustrate the average error.
- **ML Models Accuracy:** Two models are employed: k-mean clustering for phantom mode detection, measured using silhouette score, and XGBoost for baseline forecasting, assessed using RMSE and MAE for regression models.
- **User Engagement:** A one day experiment involving 3 users will gather feedback on their app experience. We will assess the relevance and helpfulness of recommendations, as well as how effectively the application aids the users in optimizing their energy consumption.
- **System Effectiveness:** Utilizing the same experiment, we will compare user energy consumption before and after the experiment, examining the reduction in energy consumption for the 3 users.

All these evaluations are summarized in the table below:

Table 26: *Evaluation Methodology, Metrics and Units.*

Aspect	Methodology	Metric(s) and unit(s)
Algorithm Performance	Asymptotic analysis + measure execution time	Time (s)
Data Accuracy	Comparison between real data and PowerEye data	MAE (kWh)
ML Models Accuracy	Model tests	Silhouette score RMSE/MAE (kWh)
User Engagement	One day experiment with 3 users to gather their feedback	User feedback
System Effectiveness	Energy consumption comparison before and after the experiment	Reduction percentage (%)

Experimental Setup:

To measure the execution time of algorithms and models, the following computer with the specified specifications is utilized:

Table 27: *Experimental Computer Specifications*

Processor	RAM	System Type	Operating System
Intel(R) Core(TM) i7-10510U CPU @ 1.80GHz 2.30 GHz	16.0 GB	64-bit operating system x64-based processor	Windows 11 Home

We have three actual users for our system, each with the following information:

Table 28: *Experiment Participants Information*

Name	Age	House Size	Rooms	Family Members	Smart Plugs
Ayat Abodayeh	21	big	10	9	4 Meross plugs
Qateralnada AlFaour	23	medium	8	7	6 Tuya plugs
Ward Al Najjar	21	medium	8	5	9 Tuya plugs

We began collecting power readings from users on August 20th, 2013. As of the submission of this document, the data collection is still ongoing. Initially, the collection process was managed by a dummy code until the deployment of the Tracker Server on November 27, 2023,

after which the server took over the collection process. All deployments were completed by the 7th of December, 2023. Subsequently, the three users installed the PowerEye application on their mobile devices, providing them access to their accounts and data. The experiment lasted for one day, constrained by time considerations.

Asymptotic Analysis:

Asymptotic analysis will be conducted on the primary algorithms within our system, primarily located in the EPR module. The focus here lies on the pseudocode outlined in the algorithms section above. The EPR module encompasses four key aspects: Energy goal, peak time, phantom mode, and energy baseline.

- **Energy goal:**
  check_goal() algorithm takes two numerical inputs, *month_energy* and *goal*. It consists of if statements and mathematical calculations, resulting in constant time complexity.
  notify_goal() also exhibits constant time complexity, involving only if statements without loops.
- **Peak time:**
  Both **check_peak()** and **notify_peak()** show constant run time, as indicated by their pseudocode.
- **Phantom mode:**
  check_phantom() contains a critical line: *model.predict()*. Since we're using K-means clustering, the time complexity of this function is $\in O(N \cdot K \cdot d)$, where N is the number of samples, K is the number of clusters, d is the number of dimensions of the samples. In our case N = 2, K = 2, d = 1, two samples (power and zero), two clusters (on/off), one dimension (power value). All of which are constant values in this context. The predict function measures the distance for each sample with each cluster center and determines the closest one, resulting in constant time complexity in our case. **notify_phantom()** also maintains constant time complexity.
- **Energy baseline:**
  As previously discussed, the check_baseline() algorithm is intricate, and detailed explanations can be found in the Jupyter notebook. The primary component involves training an XGBoost model, with a time complexity of $O(t \cdot d \cdot x \cdot \log n)$ (Washington et al., 2016), where t is the number of trees, d is the depth of the trees, x is the number of entries in the training data, and n is the number of features. All these variables are determined by the tuning function. All other preprocessing steps are bounded by O(x) or lower bounds. Therefore, the overall time complexity for the baseline is $O(t \cdot d \cdot x \cdot \log n + x)$.

As demonstrated, all algorithms exhibit $\in O(C)$. However, in our system, these functions are invoked for each appliance. Consequently, we can infer that the time complexity of the checker run() function is $\in O(N)$, where N represents the number of appliances. As mentioned earlier, the checker run() function is executed every minute, offering optimal performance with linear time complexity. However, the **check_baseline()** algorithm relies on other variables. Consequently, we might infer that the time complexity of the checker **run()** function is $\in O(N \cdot (t \cdot d \cdot x \cdot \log n + x))$, where N is the number of appliances.It's worth noting that **check_baseline()** is executed every hour, allowing more time for its execution. The validity of check_baseline() should be further validated in subsequent analyses.

Table 29: *Time Complexity for EPR Main Functions*

Function Name	Time Complexity
check_goal()	$\in O(C)$
notify_goal()	$\in O(C)$
check_peak()	$\in O(C)$
notify_peak()	$\in O(C)$
check_phantom()	$\in O(C)$
notify_phantom()	$\in O(C)$
check_baseline()	$\in O(t \cdot d \cdot x \cdot \log n + x)$
notify_baseline()	$\in O(t \cdot d \cdot x \cdot \log n + x)$
Checker.run() [min-interval]	$\in O(N)$
Checker.run() [hour-interval]	$\in O(N \cdot (t \cdot d \cdot x \cdot \log n + x))$

Results & Discussion:

Execution Run Time:

Here, we will measure the execution time of the **Checker.run()** function. Some adjustments have been made to this function to ensure that conditions are consistently satisfied, allowing us to run all check functions and achieve the worst-case scenario—consistent with our

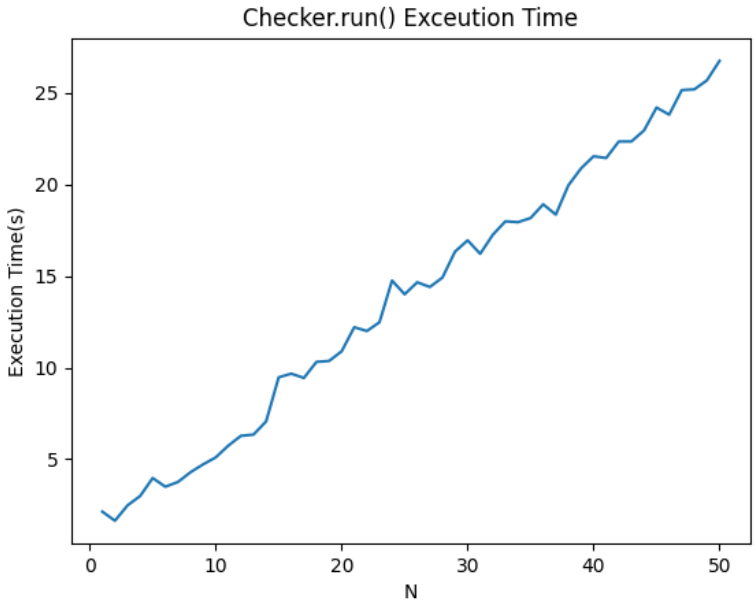
focus on asymptotic analysis. The test is conducted against varying values of N (number of appliances), with M being fixed (representing the number of power readings). The table below displays the execution times for the minimum, middle, and maximum values of N:

Table 30: *Execution Time for Min, Mid, and Max N*

	Min	Mid	Max
N	1	26	50
Time Execution (secs)	2.089	14.787	26.761

The figure below illustrates the time execution for each value of N. As anticipated, **Checker.run()** exhibits linear time complexity, which is optimal for the performance in the context of the Tracker Server.

Figure 35: *Time Execution For Checker.run()*



Energy Comparison:

Here, we present six datasets, with two for each user. The first dataset is exported from the original smart plug application (Meross/Tuya), while the second is obtained from the PowerEye database's Energys collection. Both datasets cover the same time period, from November 27,

2023, to December 6, 2023. The table below summarizes the Mean Absolute Errors for each appliance, user, and the total.

Table 31: *MAE for User 1: Ayat*

office_strip_10	tv_11
0.011	0.001

Table 32: *MAE for User 2: Qatar Alnada*

fridge_20	coffee_maker_20	lamp_20	charger_20	fan_20	hair_dryer_20
0.073	0.0	0.002	0.001	0.022	0.009

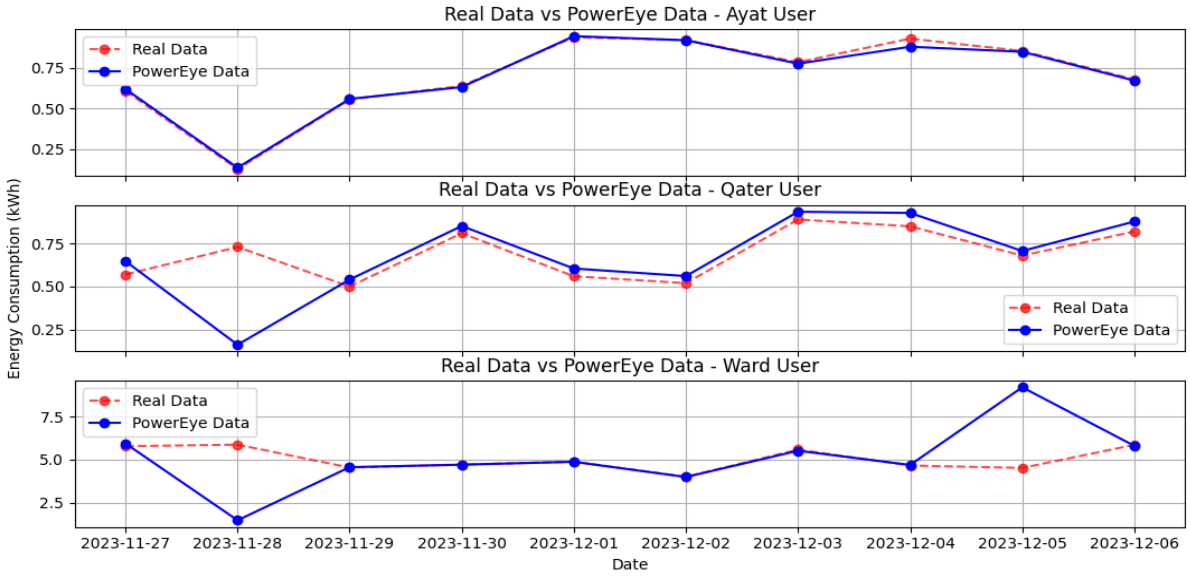
Table 33: *MAE for User 3: Ward*

cooler_30	toaster_30	charger_30	lamp_30	air_fryer_30
0.567	0.022	0.005	0.005	0.016
camera_30	**speaker_30**	**tv_30**	**office_strip_30**	
0.0	0.0	0.376	0.008	

Table 34: *Total MAE for All Users Appliances*

User 1: Ayat	User 2: Qatar Alnada	User 3: Ward	Total: All Users
0.011	0.018	0.111	0.066

Figure 36: *Real Data vs PowerEye Data*



Errors may occur for various reasons. The Tracker Server, deployed on a Single-Board Computer (SBC) rather than a dedicated server, may face challenges in collecting power readings accurately, especially in cases of unstable network connections. This issue could be addressed by providing a more stable environment for the Tracker Server. Additionally, the decision to collect power readings on a minute-by-minute basis is influenced by limitations in hardware resources. Increasing these resources could potentially allow for a reduction in the interval for collecting power readings, leading to improved accuracy. Afterall, a mean absolute error of 0.066 kWh is considered acceptable.

Machine Learning Models:

Here, we present the Silhouette score for K-means clustering models applied to phantom appliances. Additionally, we provide Root Mean Squared Error (RMSE) and Mean Absolute Error (MAE) for XGBoost models corresponding to each appliance. The models and hyperparameters were selected from the last tuning and training session conducted by the Tracker Server on December 3, 2023. The tables below summarize all the scores. If an appliance did not meet the requirements for either the K-means model or the XGBoost model, the score is left empty in the respective tables.

Table 35: *MAE for User 1: Ayat*

Score	office_strip_10	tv_11
Silhouette	0.571	0.788
RMSE	-	0.114
MAE	-	0.069

Table 36: *MAE for User 2: Qatar Alnada*

Score	fridge_20	coffee_maker_20	lamp_20	charger_20	fan_20	hair_dryer_20
Silhouette	-	-	-	-	-	-
RMSE	0.676	0.009	0.453	-	0.738	-
MAE	0.527	0.002	0.150	-	0.488	-

Table 37: *MAE for User 3: Ward*

Score	cooler_30	toaster_30	charger_30	lamp_30	air_fryer_30
Silhouette	-	-	-	-	-

RMSE	-	-	-	0.618	0.182
MAE	-	-	-	0.253	0.051

Score	camera_30	speaker_30	tv_30	office_strip_30
Silhouette	-	0.915	0.608	0.954
RMSE	0.666	0.000	0.489	0.734
MAE	0.258	0.000	0.333	0.187

As demonstrated in the tables above, certain models meet the eligibility criteria for employing XGBoost or k-means clustering, while others do not. The optimal application for k-means clustering is identified in *'office_strip_30,'* exhibiting a silhouette score of 0.954. Conversely, the results for XGBoost vary significantly across different appliances, potentially influenced by distinct usage patterns. It is noteworthy that some appliances yield a low RMSE (< 0.01), suggesting excellent results; however, this outcome is attributed to the minimal usage of these appliances, resulting in predominantly zero readings. On the other hand, an RMSE of 0.114 for *'tv_11'* and 0.453 for *'lamp_20'* are considered reasonable for a normalized dataset.

Energy Reduction:

PowerEye's main goal is to assist users in optimizing their energy consumption by providing them with energy consumption analysis and personalized recommendations. This approach aims to empower users to make more informed decisions regarding energy consumption. In this section, our focus is to assess the effectiveness of our system in achieving this goal.

Due to time constraints, we conducted a brief experiment with our existing users for a single day. It's important to note that this limited timeframe may not yield entirely accurate or representative results; a more extensive experiment is necessary for comprehensive insights. The experiment took place on December 8, 2023. In the tables and figures presented below, we will compare the data from the experiment day (2023/12/8) with the preceding day (2023/12/7).

Table 38: *Total Energy Consumption Before and After the Experiment*

Date	Ayat Total Energy Consumption
2023-12-07	0.768
2023-12-08	0.768

Date	Qater Alnada Total Energy Consumption
2023-12-07	0.720
2023-12-08	0.682
Date	**Ward Total Energy Consumption**
2023-12-07	4.374
2023-12-08	5.582

Figure 37: *Ayat Appliances Energy Consumption*

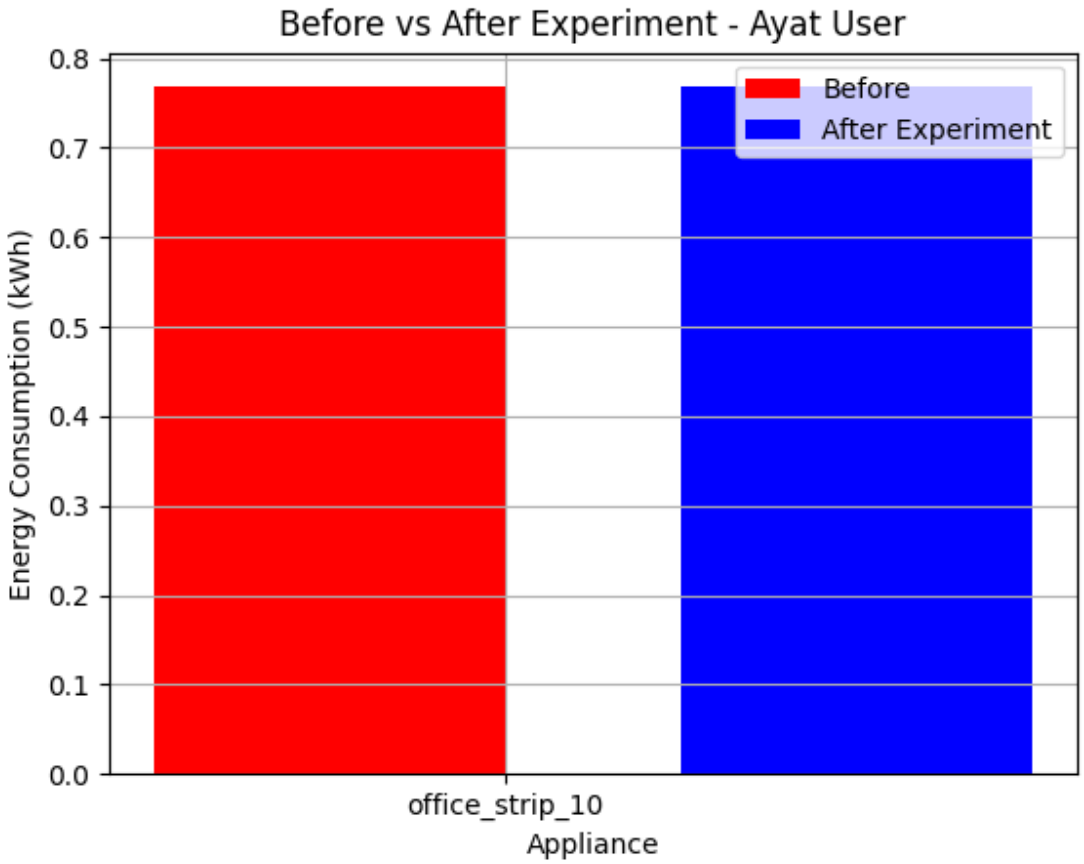


Figure 38: *Qater Alnada Appliances Energy Consumption*

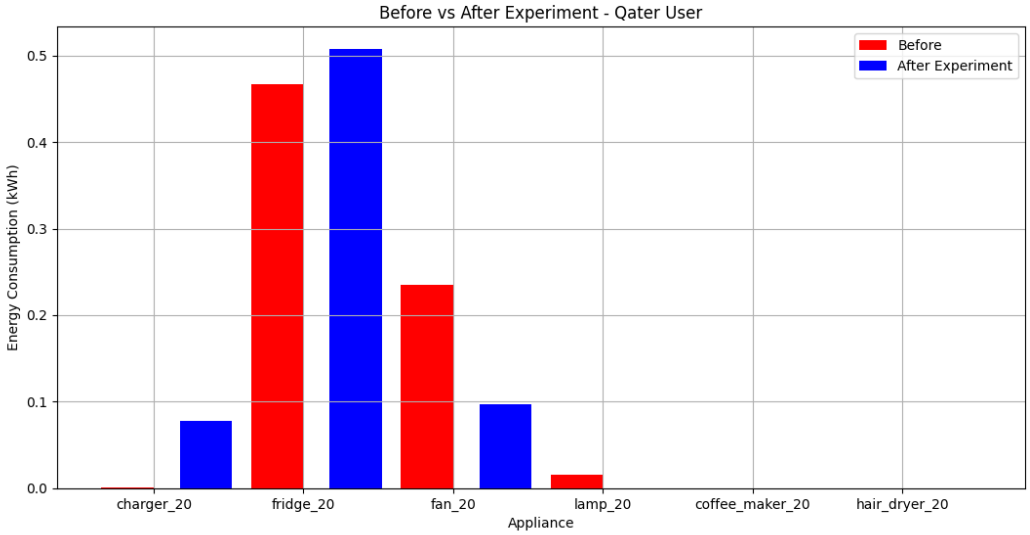
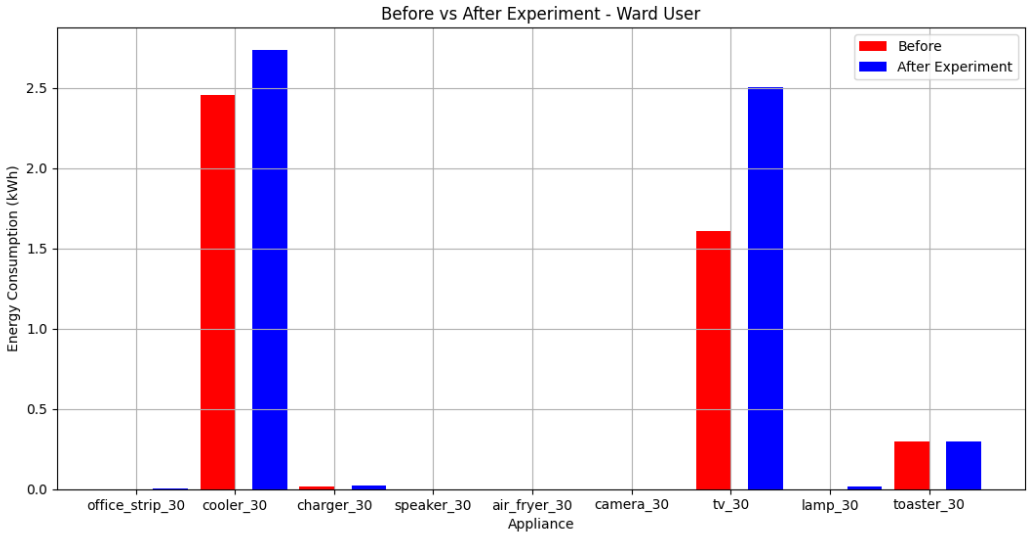


Figure 39: *Ward Appliances Energy Consumption*



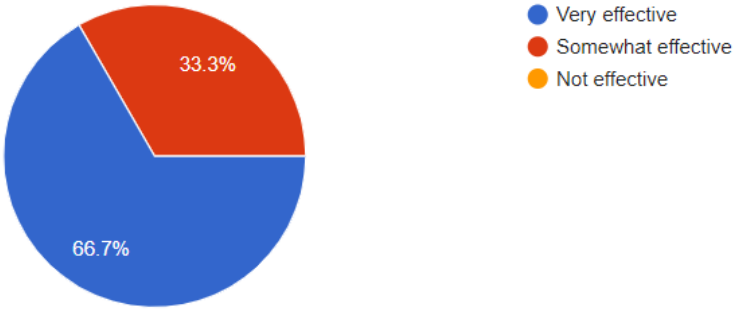
Upon observation, there wasn't a significant difference between the two days. Ayat exhibited similar energy consumption on both days. Qater Alnada showed a slight decrease in energy consumption, particularly for fan_20 and lamp_20. However, Ward's energy consumption increased for most appliances. It's worth noting that these preliminary findings emphasize the need for a more extended experiment to validate our hypothesis. Such a comprehensive study is planned for the near future, providing a more conclusive assessment.

Users Feedback Visualizations:

In order to ensure the satisfaction of our users, we have created a survey that contains a couple of questions regarding their experience while using the PowerEye application.

- **How effective was the automation in controlling your appliances through the application?**

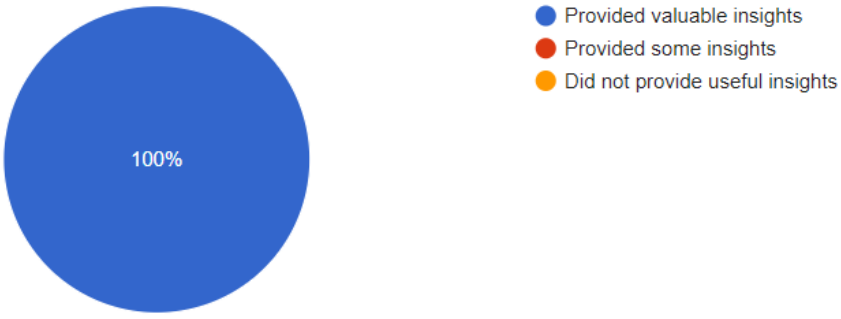
Figure 40: *result of first question in survey*



As the results show, most of the users, 66.7%, involved in the experiment find the using of PowerEye effective in controlling the appliances from the app. Only one finds the using of the app somehow effective in controlling the appliances.

- **To what extent did the energy consumption analysis pages (appliance/room/total) assist you in gaining insights?**

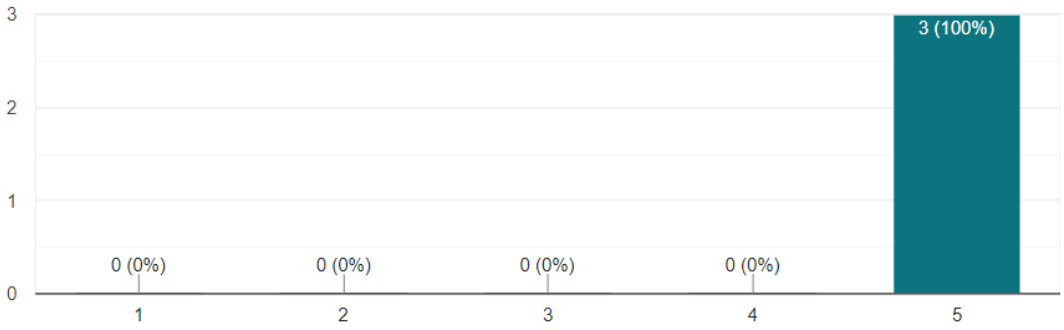
Figure 41: *result of second question in survey*



In the above chart, it's clearly shown that all of the users found the analysis provided by PowerEye system to be effective and it let them get a valuable insight about their energy consumption.

- **Did the energy consumption analysis help you in adjusting a reasonable cost goal?**

Figure 42: *result of third question in survey*



All of the users of the PowerEye system benefited from the analysis section in a way that they are able to adjust a value for the cost goal that is reasonable and within their limit.

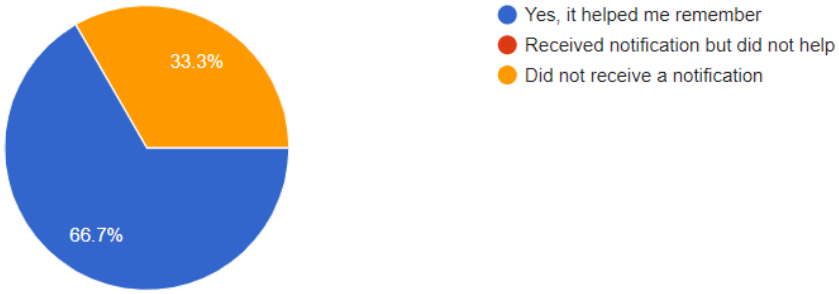
- **How accurately did the status and other information of the smart plugs reflect reality?**

Figure 43: *result of fourth question in survey*

Regarding the accuracy of the energy data analyzed by PowerWye System, all the users found that the information provided was real and accurate, so they were able to know their energy usage.

- **In case your smart plug was disconnected without your notice, did you receive a Disconnection Notification, and did it help you remember to reconnect your device?**

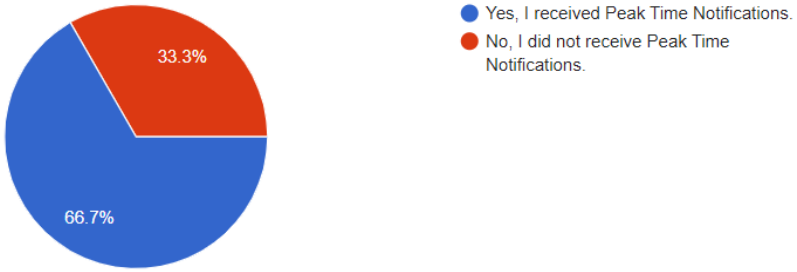
Figure 44: *result of fifth question in survey*



Upon the result provided above, 66.7% of the users were able to monitor their appliance status and get notified when some are disconnected . However, 33.3% haven't received this type of notifications,

- **Did you receive Peak Time Notifications during high-demand periods?**

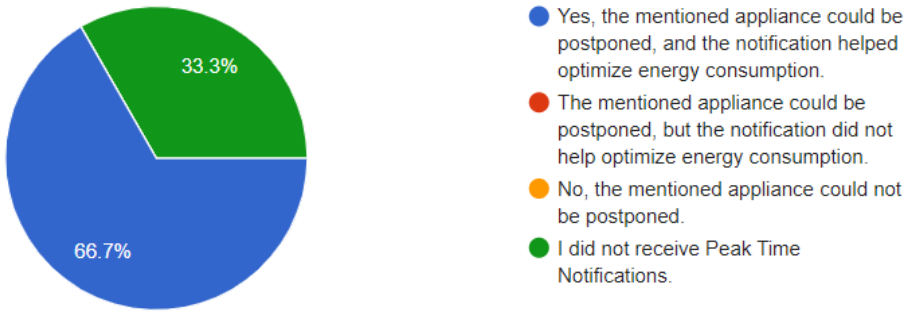
Figure 45: *result of sixth question in survey*



As shown in the result above 66.7% of users received the peak time notification, while 33.3% haven't received anything.

- **If you received Peak Time Notifications, did the mentioned appliance in the notification align with one you could postpone using? Did this notification help you optimize your energy consumption?**

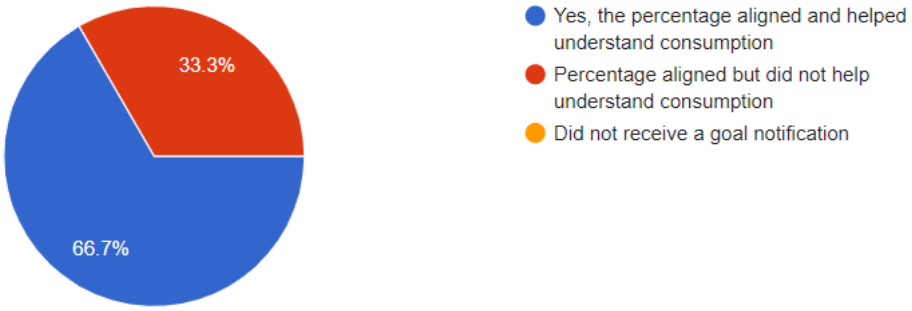
Figure 46: *result of seventh question in survey*



The result shows that the Peak time notification provided most of the users with clear and accurate information about the appliance causing that energy which assists the user in optimizing their energy.

- **If you received a goal notification, did the mentioned percentage reflect the same percentage displayed in the app? Did this notification assist you in understanding your energy consumption?**

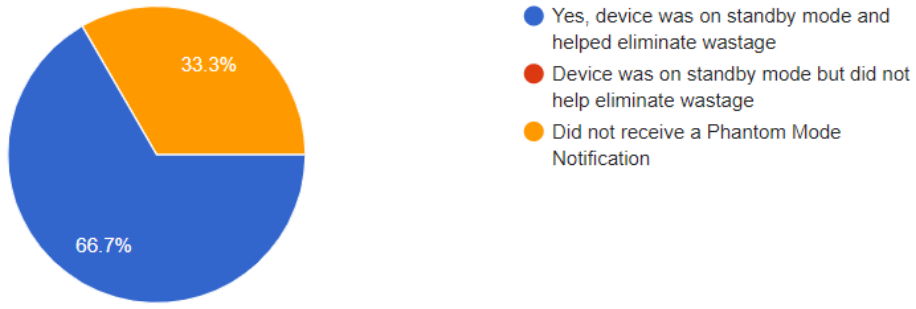
Figure 47: *result of eighth question in survey*



Depending on the result shown in the chart above, all of the users found that the mentioned goal percentage in the notification is aligned with the percentage in the app. However, 33.3% don't find that notification helpful in the way of understanding the energy consumption.

- **If you received a Phantom Mode Notification, was your device actually on standby mode? Did this notification assist you in eliminating energy wastage?**

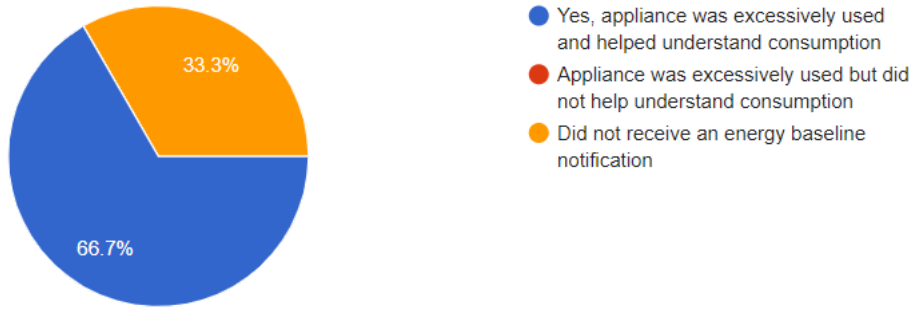
Figure 48: *result of ninth question in survey*



The chart represents that 66.7% of the users have received the phantom mode notification when their device was on standby mode which helped them eliminate wastage, while 33.3% haven't received this notification.

- **If you received an energy baseline notification, do you think the appliance mentioned in the notification was excessively used during that hour? Did it help you understand your energy consumption more?**

Figure 49: *result of tenth question in survey*



As shown in the chart above, 66.7% of the users have received the baseline notification and the appliance mentioned was excessively used at that time, while 33.3% haven't received this notification.

- **What insights did you gain from the analysis of your energy consumption?**

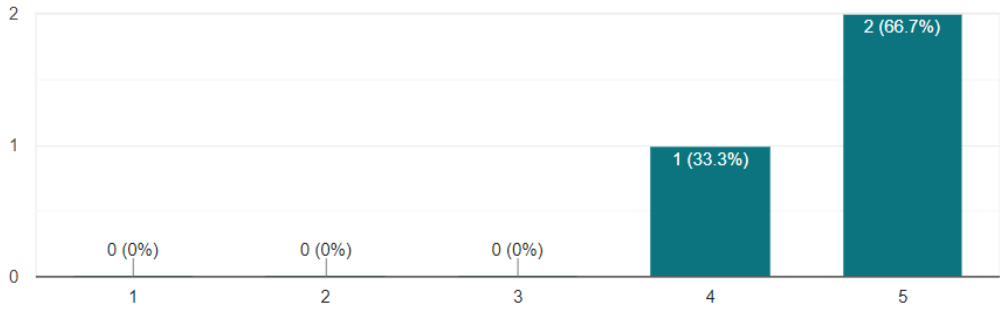
Figure 50: *result of eleventh question in survey*

I gained insights about the costs of using my appliances.
How to use the appliances efficiently
I have gained valuable insight on my personal consumption of energy, it helped me limit unnecessary use of electricity.

In the above image, you can see the insight of our users.

- **Rate your experience using PowerEye.**

Figure 51: *result of twelveth question in survey*



You can find in the chart above our users rating for PowerEye application.

- **Do you have any specific suggestions or recommendations for improvement regarding the overall application?**

Figure 52: *result of thirteenth question in survey*

I haven't received any notifications from the mobile application. Other than that, it's a great application that provided valuable insights through its dashboards. I especially liked the addition of rooms with my components.
I think connecting the application to Siri
The application works great! Maybe in the future add a predictive analysis feature for helping users plan ahead regarding their energy reduction.

As the figure above represents the recommendations given by our users, they liked the addition of rooms to the application where they can see the analysis of different chosen appliances at the same time. Also, they recommended integrating our application to Siri system

for more ease of use, as well as adding predictive analysis features to help the users plan regarding their energy consumption.

To conclude, this survey gives us a great insight about how our users experience while using the application. This survey makes us as developers of the application indicate where the issues are, so we can fix or modify them in a way that fits with the users preferences. The satisfaction of our users is the most important part that should be met before the app is published.

CONCLUSION

Summary of the report, project limitations and constraints, ethical and social implications

Summary:

The Home Energy Monitoring System, PowerEye, stands as a senior project at Prince Sultan University, symbolizing a collaborative initiative by a team from Software Engineering and Computer Science. Utilizing smart plugs, PowerEye ingeniously integrates aspects of both Intrusive Load Monitoring (ILM) and Non-Intrusive Load Monitoring (NILM), providing a user-friendly solution for advanced energy consumption monitoring. Progressing through well-defined phases and employing an iterative Waterfall model, PowerEye pioneers as the first locally developed system in the Arabian Gulf. It distinguishes itself with features such as energy sustainability tips, AI-driven data analysis, and a comprehensive scope covering user account management, device control, and insightful energy consumption analysis. The team's dedication to deliver an impactful and innovative PowerEye solution is evident in their systematic approach, addressing contemporary challenges and the limited availability of dedicated solutions in the region. The project encompasses gathering and refining system requirements in the Requirements Specification phase, creating architectural and detailed designs in the Design phase, and developing high and low-priority features in the Implementation phase. Testing involves unit, integration, and usability testing, while the Deployment phase centers on system deployment and configuration. The project concludes with Experimental Results, to assess the system's algorithm performance, data accuracy, ML model reliability, and user engagement. We aim to determine if PowerEye effectively optimizes user energy consumption. This structured process ensures a comprehensive and iterative approach to crafting a robust and user-friendly PowerEye solution.

Limitations:

The project encountered several limitations during its process. Hardware resources posed challenges, impacting the scalability and overall performance of the Home Energy Monitoring System. Data limitations due to a small user base and the availability of only small datasets, potentially affecting the system's ability to generate comprehensive insights. The use of smart plugs introduced a constraint, as the team reached for two brands for smart plugs instead of a potentially more expensive single option like Meross, addressing budget limitations. Additionally, time and budget constraints were notable challenges, influencing project scope and resource allocation decisions. Security measures were affected by financial constraints, as the team faced limitations in adopting security tools due to associated costs as all tools needed payments.

Future Work:

The project exhibits incredible potential, both on a small scale and a larger one. In a smaller scope, the system's adaptability shines through when working seamlessly with our unified smart plugs future enhancements could focus on extending compatibility to various brands, eliminating limitations and ensuring flexibility for users with different smart plug preferences. The addition of diverse energy monitoring devices, such as smart switches and meters, further enhances the system's capabilities. Integration with cloud services promises improved scalability and accessibility. Advanced features like collaborative filtering for the Energy Prediction and Recommendation (EPR) system, reinforcement learning for user feedback, and exploring superior forecasting models demonstrate the system's potential for continuous improvement.

On a larger scale, the project's application extends to factories and industrial projects, showcasing its versatility in tracking energy consumption and contributing to sustainable energy practices. This expansion positions PowerEye as a valuable tool for industries seeking efficient energy management solutions. The project's potential impact is evident, ranging from personalized home energy monitoring to broader applications in industrial settings, highlighting its versatility and scalability for diverse user needs. Additionally, a collaborative approach with the Saudi Electricity Company could further enhance the project's impact by providing home-based strategies to save on electrical bills and contribute to sustainability, aligning with the Vision 2030.

As well as broader deployment through mobile applications. Consideration could be given to publishing the application on popular platforms such as the App Store and Google Play. This would enhance user accessibility and engagement, making the PowerEye readily available to a wider audience, applicable with the Saudi Electricity Company collaboration.

Extensibility and Scalability:

PowerEye is capable of extensive scalability as seen in future works. The system can adopt both horizontal and vertical scaling strategies to cater to evolving needs. Horizontal scaling involves deploying additional servers, implementing load balancers, and utilizing clustering techniques, thereby enhancing the system's capacity to manage increased workloads. On the other hand, vertical scaling focuses on enhancing existing resources, such as upgrading hardware or increasing processing power, to meet growing demands more efficiently. This adaptability ensures HEMS can seamlessly grow, accommodating the changing requirements of users. Moreover, HEMS demonstrates its capacity to integrate with various smart plug brands and extend

beyond residential applications, potentially integrating with Industrial Internet of Things (IoT) devices and protocols to address industrial requirements.

Ethical & Social Implications:

Ethical considerations and social implications that require careful attention for PowerEye include the following points:

- **Privacy Concerns:** The collection and analysis of personal energy consumption data may raise privacy concerns, highlighting the need for encryption to safeguard user privacy.
- **Transparency and Trust**: Ensuring transparency about data security measures is crucial for building and maintaining user trust. Ethical considerations extend to addressing algorithmic biases, emphasizing fairness and transparency to avoid perpetuating social inequalities.
- **Accessibility:** Accessibility is a key ethical imperative, requiring it to be inclusive and accessible to individuals with disabilities or other statuses.
- **Environmental Impact:** HEMS projects want to align with environmental ethics, considering the long-term environmental impact of energy usage.

Appendix:

Glossary:

Table 39: *Appliances Types*

Term	Definition
Intrusive Load Monitoring (ILM)	A type of Home Energy Monitoring System (HEMS) that involves the installation of dedicated sensors directly on individual appliances or loads to monitor their energy usage.
Non-Intrusive Load Monitoring (NILM)	Another type of HEMS that allows the identification and tracking of individual electrical appliances within a household without requiring additional sensors or intrusive measurements.
Tree-cli	A robust tool for generating detailed tree-like project directory structures used for codebase illustration.
Github	A web-based platform and service that provides hosting for software development

	version control using Git.
Energy Personalized Recommender (EPR)	A component of the Tracker Server responsible for tracking energy goals, devices phantom mode, and providing personalized recommendations.
Tuya cloud	An integrated cloud service within PowerEye for data collection from Tuya smart plugs.
SHA Certificate Fingerprints	Secure Hash Algorithm certificate fingerprints used for secure communication between the Firebase Cloud Messaging infrastructure and Android devices.
APN (Apple Push Notification) Authentication Keys	Authentication keys used for secure communication between the Firebase Cloud Messaging infrastructure and iOS devices.
Amazon Machine Image (AMI)	A pre-configured virtual machine image used for creating instances (virtual servers) on AWS EC2, specifically tailored for the PowerEye web server.
Asymptotic Analysis	A method for analyzing the efficiency of algorithms as the input size approaches infinity. This is often expressed using big O notation, providing an upper bound on the algorithm's time complexity.
Big O Notation	A mathematical notation used to describe the upper bound on the time complexity of an algorithm. It provides insights into the algorithm's efficiency and scalability.
RMSE (Root Mean Squared Error)	A metric for measuring the average magnitude of the errors between predicted and actual values, providing an indication of the accuracy of a forecasting or regression model.
MAE (Mean Absolute Error)	A metric for measuring the average absolute differences between predicted and actual values, offering insights into the accuracy of predictions.
K-means Clustering	A machine learning algorithm used for partitioning a dataset into clusters. In this context, it is applied for phantom mode detection.
Silhouette Score	A metric used to calculate how similar an object is to its own cluster compared to other clusters in a clustering algorithm, providing

	insights into the quality of clustering.
XGBoost	An open-source machine learning library that provides an efficient and effective implementation of the gradient boosting framework. Used in PowerEye for baseline forecasting.

Appliances Types:

Type: the type is shown to the user.

Energy Type: the type is used in the recommender, which relies on the energy consumption patterns of the appliances.

Note: all phantom appliances are shiftable as well but not vice versa.

Table 40: *Appliances Types*

Type	Energy Type	Appliances
Coolers	None	- Fridge - Freezer - Mini Fridge - Water cooler - Fan
Lightings		- Table Lamp - Desk Lamp - Floor Lamp - Mirror light
Heaters		- Microwave - Toaster - Stove - Kettle
Cookers/Makers		- Coffee Maker - Popcorn Maker - Air Fryer - Slow Cooker
Blenders		- Blender - Food Processor
NO CATEGORY		- Clock/Alexa - Hair Dryer - Camera
Washing Machines	Shiftable	- Washer - Dishwasher - Clothes Dryer

NO CATEGORY		- Iron - Vacuum Cleaner - Air Purifier
Games	Phantom	- PlayStation - Wii - Xbox
Displayers		- PC - Monitor - Projector - TV
Audio Output		- Speaker - Radio
NO CATEGORY		- Printer - Charger - Receiver - Sports Machine

Notifications types:

Table 41: *Notifications Types*

Notification	Description
CREDS	Sent when the user updates their credentials on the original smart plugs application (Meross) but doesn't update them on our app
DISCONNECTION	Sent when an appliance becomes unreachable due to internet or electricity disconnection
GOAL	Sent when the user's monthly energy consumption reaches a certain percentage (25-200%) of their goal
PEAK	Sent when a shiftable appliance is detected to be active during peak times
PHANTOM	Sent when a phantom appliance is detected to be in phantom mode (standby)
BASELINE	Sent when an appliance surpasses its daily energy consumption baseline

Recommender System Types:

Recommender systems are broadly categorized into three main types: collaborative filtering, content-based, and hybrid.

- **Collaborative filtering systems** suggest items by considering the preferences or ratings of other users. For instance, you can begin with general recommendations and refine them by collecting user feedback to provide more relevant suggestions to similar users. **Neighborhood-based methods** are commonly employed here. One approach involves clustering users based on their appliances and preferences, then dividing each cluster into subgroups according to their electricity bills. Recommendations can then be tailored from the low-bill group's practices to the high-bill group (Luo et al., 2021).

- **Content-based systems** recommend items based on the characteristics of the items themselves. For example, suggesting energy-efficient appliances based on the user's existing appliances or preferences (Luo et al., 2017). **Case-based systems** are a subtype of content-based systems that rely on historical cases to make recommendations. For example, predicting energy reductions based on factors like temperature, humidity, date, and time (Pinto et al., 2019). **Knowledge-based systems** (rule-based) are another type of content-based systems. They recommend items by using a knowledge base of items and their relationships. One approach involves using predefined rules to provide recommendations based on user appliances, personal information, or preferences (Shigeyoshi et al., 2013). Another approach, using sequential association rules to uncover patterns in user energy consumption and then suggesting device usage changes based on these relationships (Osadchiy et al., 2018).

- **Hybrid systems** combine both collaborative filtering and content filtering. While collaborative filtering techniques are effective, they face challenges like the cold start problem. To address this, combining content-based approaches with collaborative filtering helps mitigate the issue.

External Links:

- Software Requirements Specification (SRS)
- Software Design Document (SDD)
- Software Testing Document
- Energy Personalized Recommender Jupyter Notebook
- Demo Videos
- Usability Test Videos
- PowerEye-frontend Github Repository
- PowerEye-backend Github Repository
- User Guide Document

- Frequently Asked Questions Document
- iOS mobile app build
- Android mobile app build

References:

- Altrabalsi, H., Stankovic, V., Liao, J., & Stankovic, L. (2016, January 5). Low-complexity energy disaggregation using appliance load modeling. AIMS Energy. https://doi.org/10.3934/energy.2016.1.1
- Merossiot Library's documentation. https://albertogeniola.github.io/MerossIot/index.html
- TinyTuya: Python module to interface with Tuya WiFi smart devices. https://github.com/jasonacox/tinytuya
- *Consumption tariffs*. Saudi Electricity Company. (n.d.). https://www.se.com.sa/en/Ourservices/ColumnC/Bills-and-Consumption/ConsumptionTariffs/
- Abdulkareem, A., & Ellaboudy, A. (2023, January 10). *Saudi Electricity Company*. Climate Scorecard. https://www.climatescorecard.org/2023/01/saudi-electricity-company/#:~:text=However%2C%20the%20emission%20intensity%20from,per%20kilowatt%2Dhour%20in%202021
- Techlabs, M. (2021, August 18). *What are the types of recommendation systems?*. Medium. https://medium.com/mlearning-ai/what-are-the-types-of-recommendation-systems-3487cbafa7c9
- *Consumption Rationalization, Industrial sector*. Saudi Electricity Company. (n.d.). https://www.se.com.sa/en/Electricity-Information/Consumption-Rationalization/Industrial-Sector/
- Pinto, T., Faia, R., Navarro-Caceres, M., Santos, G., Corchado, J. M., & Vale, Z. (2019). Multi-agent-based CBR recommender system for Intelligent Energy Management in buildings. *IEEE Systems Journal,* 13(1), 1084–1095. https://doi.org/10.1109/jsyst.2018.2876933
- Osadchiy, T., Poliakov, I., Olivier, P., Rowland, M., & Foster, E. (2018). Recommender system based on Pairwise Association rules. *Expert Systems with Applications,* 115, 535–542. https://doi.org/10.1016/j.eswa.2018.07.077
- Shigeyoshi, H., Tamano, K., Saga, R., Tsuji, H., Inoue, S., & Ueno, T. (2013). Social Experiment on advisory recommender systems for energy-saving. *Human Interface and the Management of Information. Information and Interaction Design*, 545–554. https://doi.org/10.1007/978-3-642-39209-2_61

- F. Luo, G. Ranzi, W. Kong, G. Liang and Z. Y. Dong, "Personalized Residential Energy Usage Recommendation System Based on Load Monitoring and Collaborative Filtering," in *IEEE Transactions on Industrial Informatics,* vol. 17, no. 2, pp. 1253-1262, Feb. 2021. https://doi.org/10.1109/TII.2020.2983212
- Luo, F., Ranzi, G., Kong, W., Dong, Z.Y., Wang, S. and Zhao, J. (2017), Non-intrusive energy saving appliance recommender system for smart grid residential users. *IET Gener. Transm. Distrib*. 11: 1786-1793. https://doi.org/10.1049/iet-gtd.2016.1615
- Washington, T. C. U. of, Chen, T., Washington, U. of, Washington, C. G. U. of, Guestrin, C., Ibm, Bosch, Amazon, Baidu, & Metrics, O. M. A. (2016, August 1). *XGBoost: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and data mining.* ACM Conferences. https://dl.acm.org/doi/10.1145/2939672.2939785