A PROJECT REPORT

ON

"SMART HOME ENERGY MANAGEMENT SYSTEM"

Dissertation submitted to Telangana University, Nizamabad. In the partial fulfillment of the requirements for the Award of Post Graduate Degree of

MASTER OF COMPUTER APPLICATIONS

By

SHAIK NASEERODDIN

(5174-23-862-089)

Under the esteemed guidance of

P.SHIREESHA

M.TECH (CSE)

Assistant Professor in CS

Department of

Master of Computer Applications

NISHITHA DEGREE COLLEGE

(Approved by AICTE & Affiliated to TU Nizamabad)

Near S.P Office, Kanteshwar Road, Nizamabad. (T.S)

A PROJECT REPORT

ON

"SMART HOME ENERGY MANAGEMENT SYSTEM"

Dissertation submitted to Telangana University, Nizamabad. In the partial fulfillment of the requirements for the Award of Post Graduate Degree of

MASTER OF COMPUTER APPLICATIONS

By

SHAIK NASEERODDIN

(5174-23-862-089)

Internal Guide

P.SHIREESHA

M.TECH (CSE)

Assistant Professor

Head of the Department

P.PRASAD

MCA

Assistant Professor

Department of Master of Computer Applications

NISHITHA DEGREE COLLEGE

(Approved by AICTE & Affiliated to TU Nizamabad)

Near S.P Office, Kanteshwar Road, Nizamabad. (T.S)

Myra IT Technologies

CERTIFICATE

This is to certify that **SHAIK NASEERODDIN** bearing Hall ticket No. **5174-23-862-089** from **NISHITHA DEGREE COLLEGE**, student of M.C.A has successfully completed the project entitled as **"SMART HOME ENERGY MANAGEMENT SYSTEM "**part of the course curriculum in our Organization**.**

He has done project during the period for **3 Months** under the guidance of **Mr. Kasarla Shanthan Reddy**, Project coordinator, in **MYRA IT TECHNOLOGIES, HYDERABAD**

During the period of his project work with us, we found his conduct and character to be Good.

Yours Sincerely,
From Myra IT Technologies.

For MYRA IT TECHNOLOGIES,
Partner.

Kasarla Shanthan Reddy
Managing Director.

Myra IT Technologies

518, Annapurna Block, Adithya Enclave,Ameerpet, Hyderabad, Telangana

NISHITHA DEGREE COLLEGE

(Approved by AICTE & Affiliated to TU Nizamabad)
Near S.P Office, Kanteshwar Road, Nizamabad. (T.S)

DEPARTMENT OF MASTER OF COMPUTER APPLICATIONS

CERTIFICATE

This is to certify that the project report entitled **"SMART HOME ENERGY MANAGEMENT SYSTEM"**, being submitted by **SHAIK NASEERODDIN** bearing Roll No. **5174-23- 862-089**. In partial fulfillment of the requirements for the award of the degree of **Master of Computer Applications** by Telangana University, Nizamabad is a record of the bonafide work carried out by the student under our guidance and supervision during the period 2023-25.

This project has been completed with the collaboration of **"MYRA IT TECHNOLOGIES, HYDERABAD".**

INTERNAL GUIDE
P.SHIREESHA
M.TECH (CSE)
Assistant Professor

PRINCIPAL
DR. MOHD AZHER PARVEZ

EXTERNAL EXAMINER

BONAFIDE CERTIFICATE

This is to certify that this thesis titled **"SMART HOME ENERGY MANAGEMENT SYSTEM"**, is the bonafide work of **SHAIK NASEERODDIN** bearing Roll No. **5174-23-862-089**, who carried out by the research under my supervision, certified further, that to the best of my knowledge the work reported here in does not form part of any other thesis or dissertation on the basis of which a degree or award was conferred on an earlier occasion of thesis of any other candidate.

SUPERVISOR

Signature :

Name	: P.SHIREESHA
Academic Designation	: ASSISTANT PROFESSOR
Department	: MASTER OF COMPUTER APPLICATIONS
University/College/ Organization with Address	: Nishitha Degree College (Affiliated to TU) Near S.P. Office, Kanteshwar Road, Nizamabad

DECLARATION

I hereby declare that this project entitled "**SMART HOME ENERGY MANAGEMENT SYSTEM**" embodies our project word carried out during final year "**MASTER OF COMPUTER APPLICATIONS**" 2023-2025 and has not been submitted previously for the award of any other degree or diploma by me or any others to any institute or university according to the best of my knowledge.

SHAIK NASEERODDIN

(5174-23-862-089)

ACKNOWLEDGEMENT

I express my sincere thanks to all those who have guided me in successful completion of my project work.

Firstly, I would like to record my deep sense of gratitude towards our principal **DR.MOHD AZHER PARVEZ** sir for his concern and co operation.

I convey our deep gratitude to the management of **NISHITHA EDUCATIONAL ACADEMY,** Nizamabad for giving us opportunity to carry out this project work.

I would like to thank is my **INTERNAL GUIDE – P.SHIREESHA**, Assistant Professor of Master of Computer Application Department, **Nishitha Degree College, Nizamabad**. His understanding, encouragement and personal guidance have provided the basis for this thesis. He is a source of inspiration for innovative ideas and his kind support is well known to all his students and colleagues.

I wish to thank all the **Faculty and Staff** of department of **Master of Computer Applications** for their in time help to complete this work.

Finally I would thank **my parents**, all **my friends** and **well wishers** who directly or indirectly helped me in accomplishing the project work successfully.

SHAIK NASEERODDIN
(5174-2-862-089)

ABSTRACT

The Smart Home Energy Management System is a full-stack web application designed to monitor, manage, and forecast electricity usage in a smart home environment. With the growing importance of energy conservation, this system enables users to track power consumption and optimize their usage patterns effectively.

The frontend is developed using HTML, CSS, and JavaScript, enhanced with Chart.js for visualizations. The backend is powered by Node.js, and data is stored in MongoDB, which allows efficient storage and retrieval of device and energy data. Energy usage records are automatically logged, and device statuses are monitored and updated in real- time.

To forecast future electricity usage, the system integrates a Python-based ARIMA (Auto Regressive Integrated Moving Average) model. This time-series forecasting algorithm analyzes historical data from MongoDB and predicts upcoming energy consumption, allowing users to make informed decisions.

Users can interact with the system through a clean dashboard, view monthly statistics, see device-wise usage, and test prediction functionality. The project is modular, scalable, and deployable on cloud platforms like Render.com, making it accessible from anywhere.

This solution helps reduce energy wastage, lower costs, and promotes smarter living by combining web development with machine learning and IoT-like functionality. It demonstrates how intelligent software systems can contribute to sustainable energy practices in daily life

CONTENTS

Introduction

In today's technologically advancing world, energy conservation and efficient energy usage are more critical than ever. With the rise in electricity consumption due to smart appliances and connected devices, managing energy usage in homes has become essential. The smart home energy management system aims to provide a digital solution to monitor, predict, and optimize energy usage using real-time data and forecasting models.

This project utilizes Node.js for backend development, MongoDB for flexible data storage, and ARIMA (Auto Regressive Integrated Moving Average) as the core forecasting model to predict future energy consumption. The system captures energy usage data, displays it visually, and recommends optimization strategies to reduce wastage.

By offering functionalities like real-time energy tracking, device-wise control, and energy consumption prediction, this system contributes towards reducing electricity bills and supports a greener environment. The web-based interface is user-friendly and accessible from any device connected to the internet, ensuring convenience and control for homeowners.

This documentation outlines the system design, technical implementation, software requirements, and testing process involved in developing the Smart Home Energy Management System.

2. System Analysis

2.1 Existing System

In traditional households, energy usage is typically monitored through monthly electricity bills, offering limited visibility into real-time consumption patterns or specific device contributions. Users lack control over energy-hungry devices and cannot forecast future usage or costs. Manual monitoring and absence of analytics lead to inefficiencies, energy wastage, and higher utility expenses. Additionally, there's no system in place to identify unusual energy spikes or optimize device operation based on predicted consumption.

2.2 Proposed System

The proposed Smart home energy management system introduces a modern, automated solution for tracking and forecasting home energy usage. It uses sensors and software to collect device-wise energy data and stores it in a MongoDB database. The backend, powered by Node.js, processes this data and makes it available through APIs to the frontend interface.

Key features of the proposed system include:

- **Real-time Monitoring**: Users can view current energy usage and device statuses.
- **Forecasting with ARIMA**: Predicts future energy consumption based on historical data.
- **Device-wise Analysis**: Helps users understand which appliances consume the most power.
- **Optimization Suggestions**: Recommends when to turn devices on/off to save power.
- **Admin Panel**: Allows administrators to add, remove, and control smart devices.

3. System Requirement Specification

This section outlines the necessary functional and non-functional requirements of the Smart Home Energy Management System (SHEMS). It includes both hardware and software requirements essential for the successful development and deployment of the project. The system is designed to monitor, record, and predict energy usage in a smart home environment using real-time data and ARIMA time-series forecasting.

3.1 Modules Description

The system is divided into the following major modules:

1.User Interface Module

- Developed using HTML, CSS, and JavaScript.
- Provides a dashboard for visualizing energy usage, device control, and test predictions.
- Displays historical and real-time energy consumption data in chart formats.

2. Device Management Module

- Allows users to add, update, or remove devices in the system.
- Maintains device-specific data like wattage, location, status (ON/OFF), and type.
- Sends device data to backend for usage calculations.

3. Energy Recording Module

- Captures real-time total energy usage (in watts) from active devices.
- Stores data in MongoDB with timestamp information.
- Enables energy usage tracking and prediction accuracy.

4. Prediction Module (ARIMA)

- Uses historical energy usage stored in MongoDB to forecast upcoming consumption.
- Implemented in Python using the ARIMA model.
- Backend (Node.js) communicates with the Python script to retrieve predictions.

5. Database Module

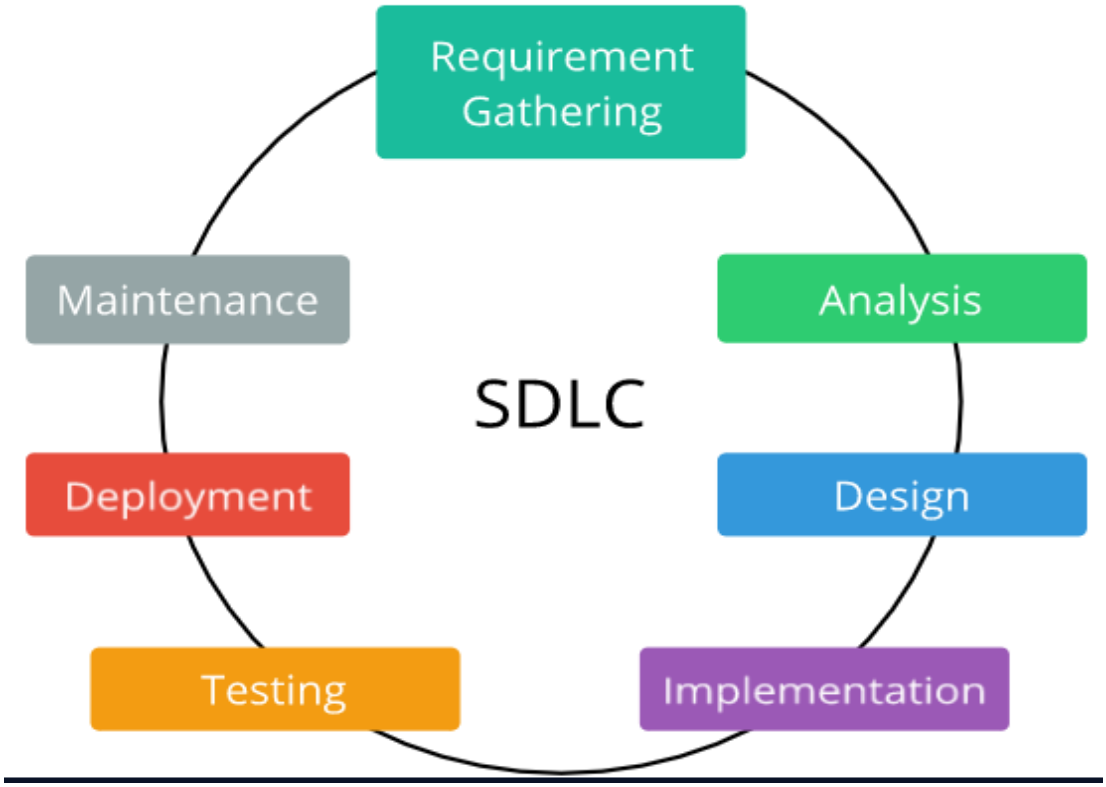
- Built using MongoDB.
- Stores energy usage logs, device metadata, and prediction history.
- Ensures data integrity and supports analytics.

6. Admin Control Module

- Provides access to manage system settings like energy rate (₹/kWh), peak hours, and notifications.
- Controls device statuses and optimization strategies.

3.2 SDLC Methodology

For the development of the Smart Home Energy Management System (SHEMS), the Waterfall Model of the Software Development Life Cycle (SDLC) has been adopted. This model provides a structured and linear approach to software development, where each phase must be completed before the next begins. The phases involved in this project are as follows:



1. Requirement Analysis

- Identified the problem of uncontrolled energy usage in smart homes.
- Gathered requirements such as real-time monitoring, device-wise control, forecasting future usage, and database integration.

2. System Design

- Created high-level and detailed designs for system architecture, user interface, database schema, and ARIMA prediction integration.
- Decided on technology stack: Node.js, MongoDB, HTML/CSS/JS (frontend), and Python for ARIMA.

3. Implementation

- Developed frontend dashboard in HTML, CSS, and JavaScript.
- Built backend using Node.js and Express.js.
- Created ARIMA prediction script in Python.
- Connected MongoDB to store energy data and devices.

4. Testing

- Conducted unit testing for each module.
- Performed integration testing between frontend, backend, and database.
- Validated prediction outputs and system performance.

5. Deployment

- Deployed backend to Render.com.
- Used MongoDB Atlas for cloud database storage.
- Ensured system accessibility from any location.

6. Maintenance

- Designed system to be scalable and easy to maintain.
- Future improvements can include adding user authentication, LSTM model integration, or mobile compatibility

3.3 SOFTWARE REQUIREMENTS:

Software Component	Description
Operating System	Windows 10 or higher
Backend Server	Node.js
Front End	HTML, CSS, JavaScript
Database	MongoDB (local or MongoDB Atlas)
Scripting Language	JavaScript
Machine Learning Model	Python (ARIMA using statsmodels)
Python Libraries	pandas, stats models, pymongo
Deployment Platform	Render.com

3.4 HARDWARE REQUIREMENTS:

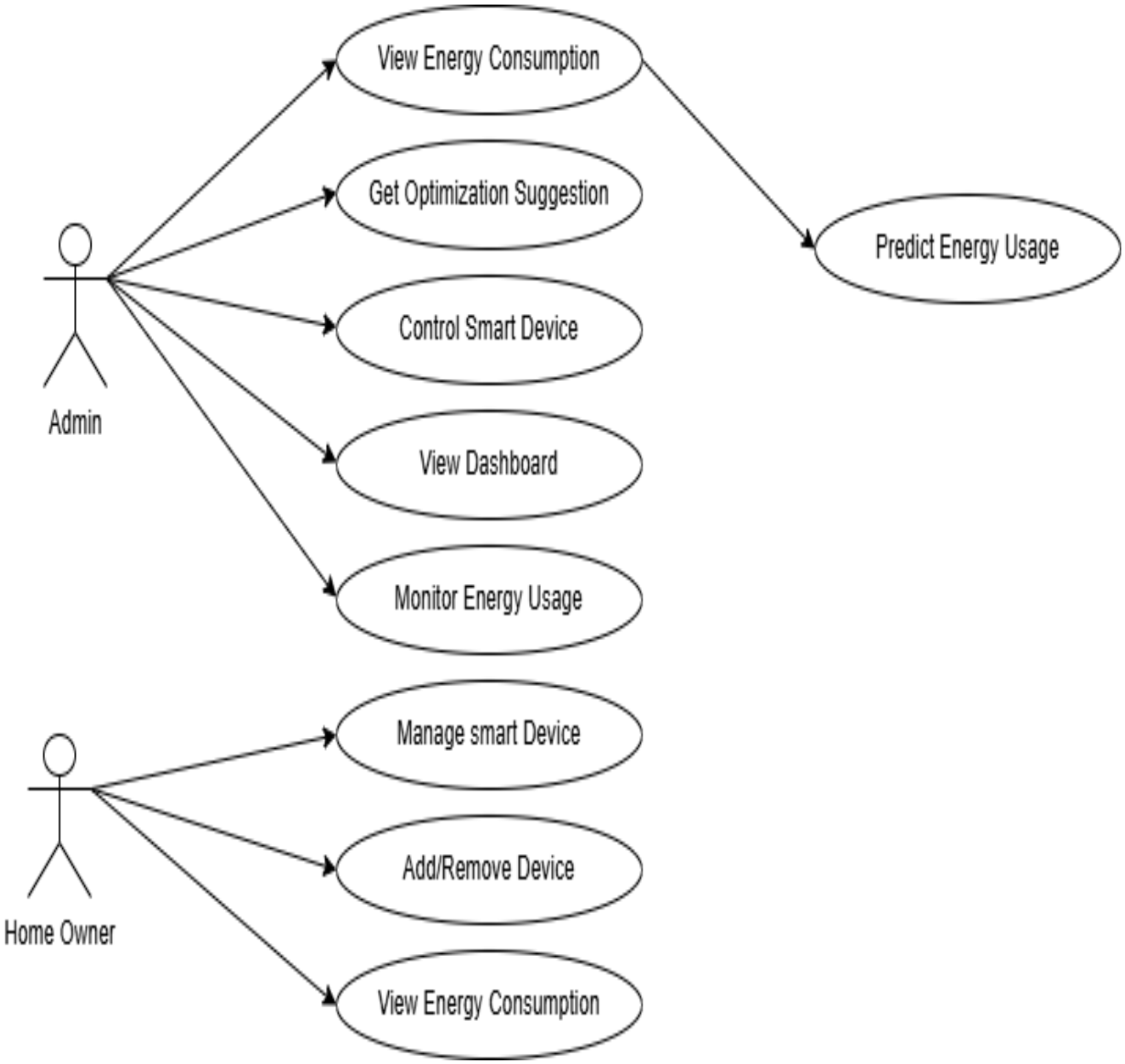
Hardware Component	Description
Operating System	Windows 10 or higher
Processor	Intel Core i3 or higher
RAM	4 GB or above
Hard Disk	20 GB or above
Internet Connection	Required for cloud database and deployment

4. System Design

The System Design phase is crucial for translating requirements into a structured solution architecture. The Smart Home Energy Management System is designed using a modular and layered approach that ensures smooth interaction between the user interface, backend, database, and machine learning prediction logic.

4.1 Use Case Diagram:

The Use Case Diagram represents the interactions between users (actors) and the system functionalities in the Smart Home Energy Management System. This system supports two primary actors: Admin and Home Owner. Each actor has specific permissions and functionalities they can access.



Actors and Use Cases:

Admin:

View Energy Consumption:

Allows the admin to monitor the energy usage data across all connected devices.

Get Optimization Suggestion:

Admin receives suggestions based on the forecasted usage and real-time monitoring.

Control Smart Device:

Enables turning devices on or off or adjusting settings for better energy management.

View Dashboard:

Displays a summarized visual dashboard of system statistics and alerts.

Monitor Energy Usage:

Continuously tracks the overall energy consumption for all registered devices.

Predict Energy Usage:

Based on the energy consumption view, the system predicts future usage using ARIMA.

Home Owner:

Manage Smart Device:

Homeowners can interact with the system to control or update device settings.

Add/Remove Device:

Provides the ability to add new smart devices or remove unused ones.

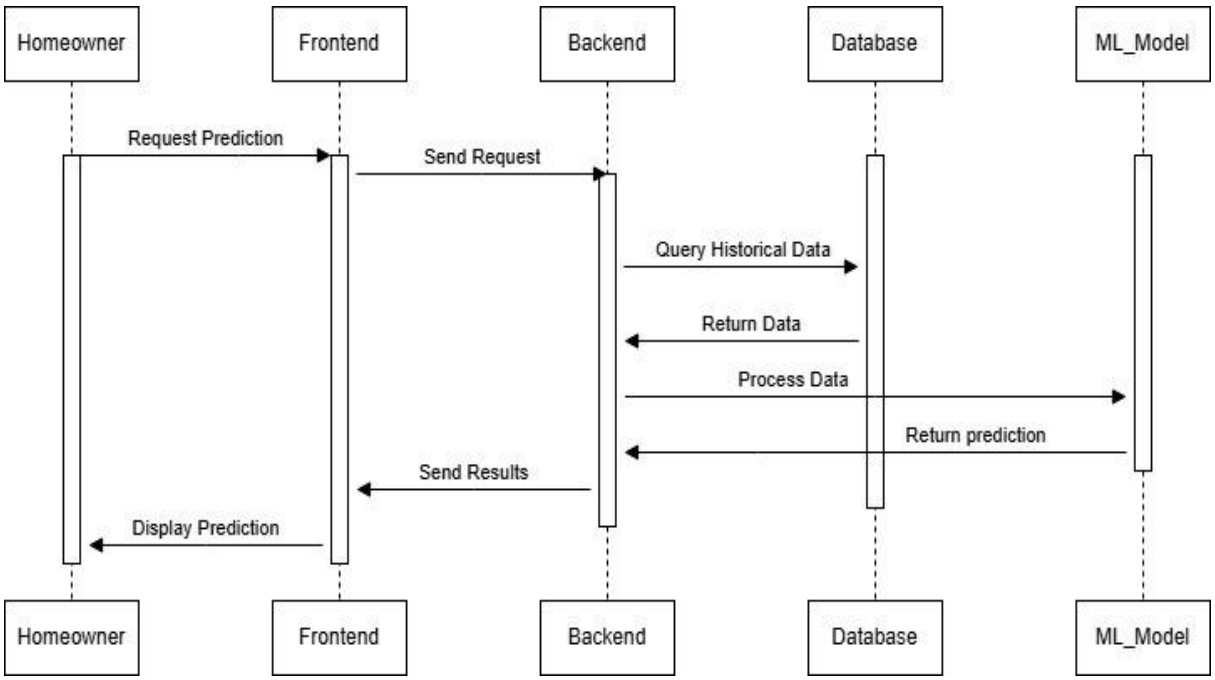
View Energy Consumption:

Homeowners can monitor their own usage to stay informed and efficient.

This diagram effectively showcases the roles and permissions in the system and how users interact with energy management features such as forecasting, optimization, and device control.

4.2 Sequence Diagram

A sequence diagram shows how objects or components in the system interact with each other over time. It focuses on the order of messages passed between various elements in the system to carry out specific functions such as energy prediction, device control, or data logging.



Actors & Components:

Homeowner: Initiates the prediction request.

Frontend: The UI through which the request is sent and results are displayed.

Backend: The Node.js server that handles logic and routes.

Database: MongoDB that stores historical energy usage data.

ML_Model: ARIMA model that forecasts future energy usage.

Flow Description:

Request Prediction (Homeowner → Frontend)

The homeowner clicks a "Send Test Prediction" button on the UI to request a forecast.

Send Request (Frontend → Backend)

The frontend sends a POST request to the backend via an API endpoint (/predict).

Query Historical Data (Backend → Database)

The backend queries the MongoDB database to fetch energy usage history.

Return Data (Database → Backend)

MongoDB returns the historical energy data to the backend.

Process Data (Backend → ML_Model)

The backend invokes the ARIMA model (predict.py) and passes the historical data for processing.

Return Prediction (ML_Model → Backend)

The ML model returns the forecasted energy usage value.

Send Results (Backend → Frontend)

The backend sends the prediction result back to the frontend.

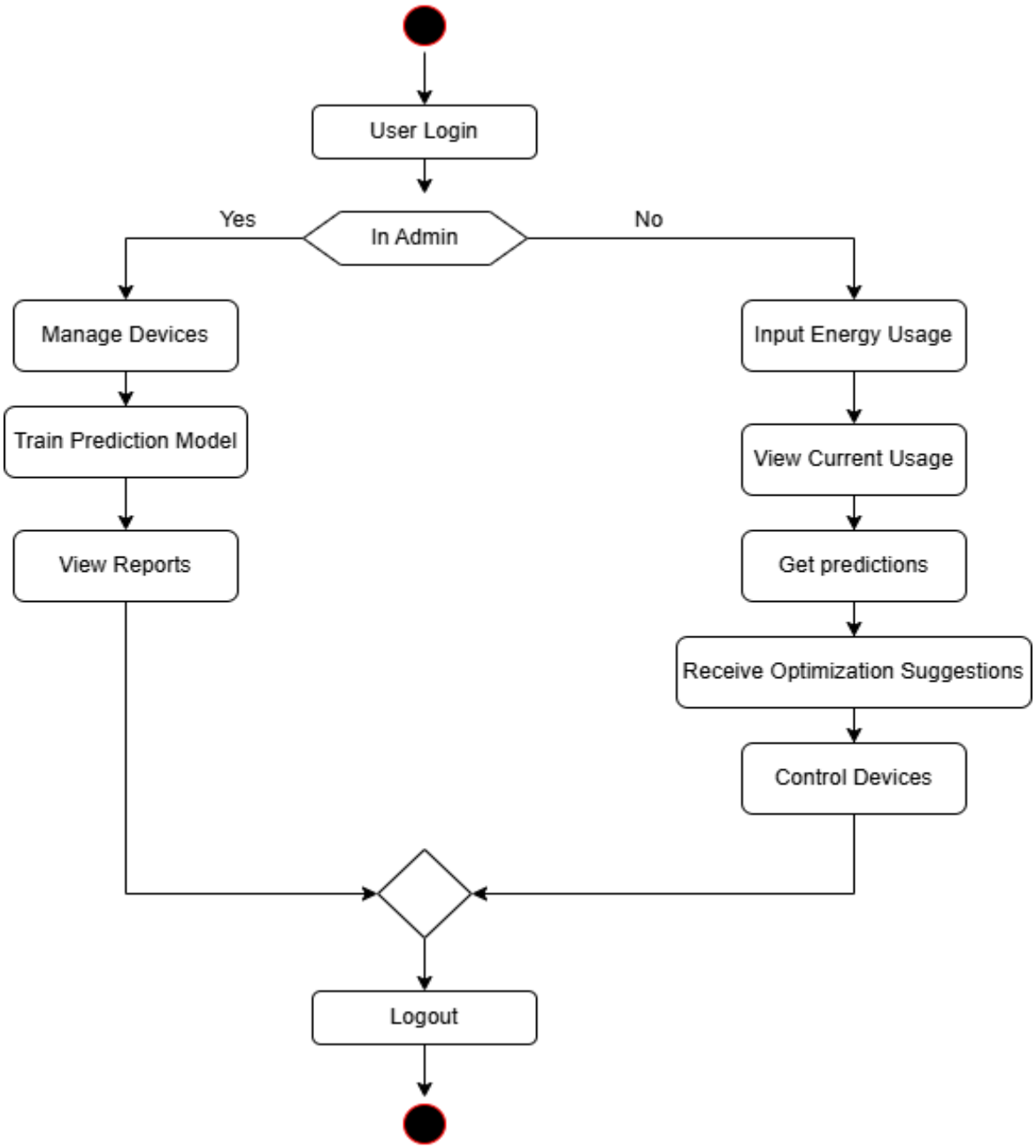
Display Prediction (Frontend → Homeowner)

The frontend displays the predicted energy consumption (e.g., in kWh) to the user via an alert or dashboard.

This interaction sequence ensures the system provides timely and accurate energy forecasts, helping homeowners manage consumption more efficiently.

4.3 Activity Diagram

This activity diagram models the sequence of actions based on user roles (Admin vs. Homeowner) from login to logout.



Flow Description:

1.User Login:

Every session begins with user authentication.

2.Decision: Is Admin?

The system checks the role of the logged-in user.

If Admin:

Manage Devices: Admin can view, add, remove, or update smart devices.

Train Prediction Model: Admin initiates the ARIMA model for forecasting based on collected energy data.

View Reports: Admin can monitor system performance, usage trends, and prediction results.

If Homeowner:

Input Energy Usage: Homeowner manually or automatically submits energy usage data.

View Current Usage: View device-wise or total consumption from the dashboard.

Get Predictions: Frontend sends a request to backend, which invokes the ARIMA model for prediction.

Receive Optimization Suggestions: System suggests actions to save energy (e.g., turning off unused devices).

Control Devices: Homeowner can turn devices on or off via the interface.

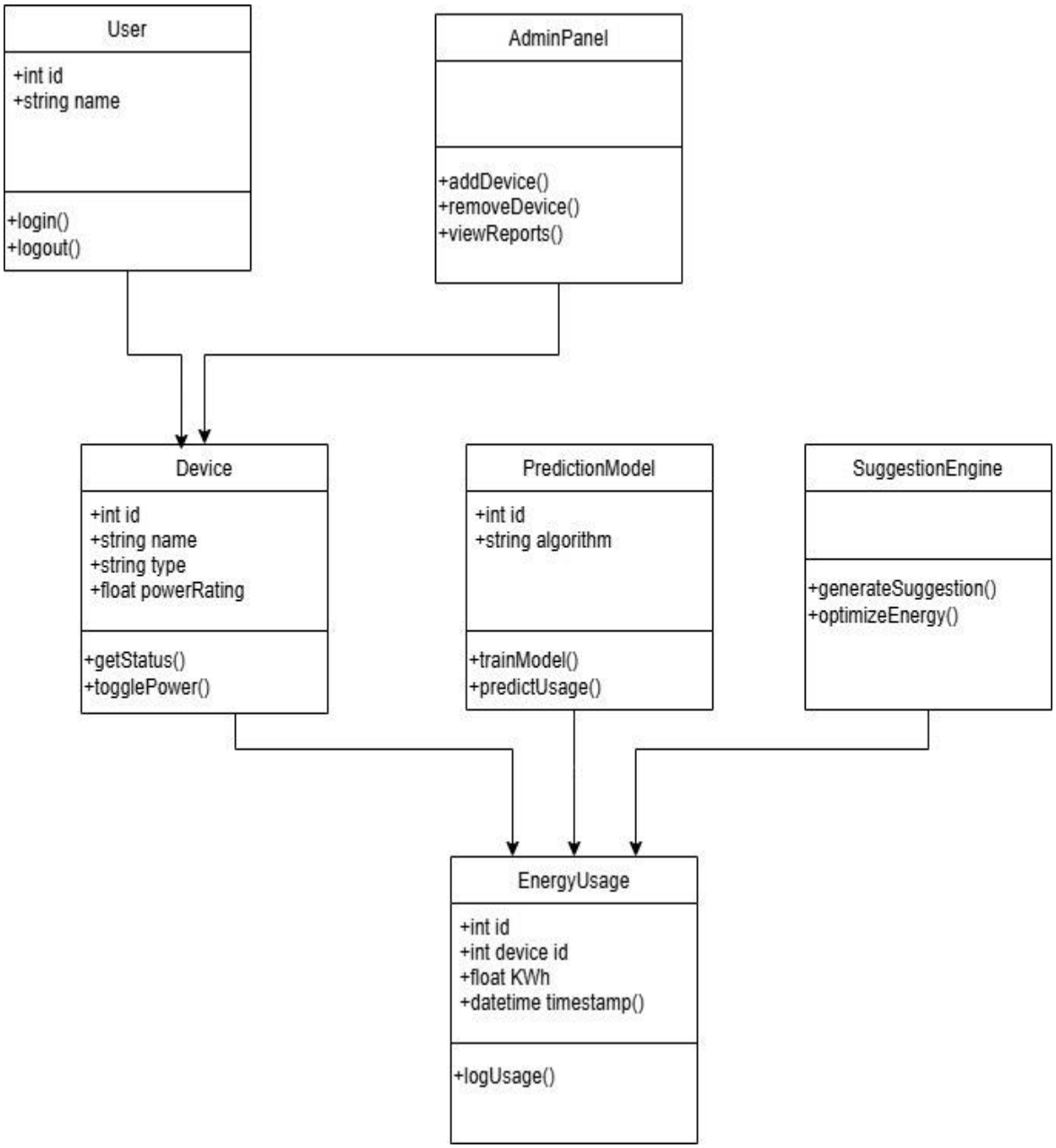
Logout:

After operations, user logs out and the session ends.

This diagram visualizes the entire operational flow based on user roles, ensuring clarity on how system functionality differs between admin and homeowner access.

4.4 Class Diagram

The class diagram outlines the static structure of the system by defining key classes, their attributes, and the operations they perform.



1. User

- **Attributes**:
  - id: Integer – Unique identifier for each user.
  - name: String – Name of the user.
- **Methods**:
  - login(): Authenticates the user.
  - logout(): Ends the user session.

2. AdminPanel

- **Methods**:
  - addDevice(): Adds a new smart device to the system.
  - removeDevice(): Deletes an existing device.
  - viewReports(): Generates usage and prediction reports.

3. Device

- **Attributes**:
  - id: Integer – Unique device identifier.
  - name: String – Name of the device.
  - type: String – Type/category (e.g., AC, Light).
  - powerRating: Float – Power consumption of the device in watts.
- **Methods**:
  - getStatus(): Checks whether the device is ON or OFF.
  - togglePower(): Switches the device ON or OFF.

4. PredictionModel

- **Attributes**:
  - id: Integer – Identifier for the model.
  - algorithm: String – Type of forecasting used (ARIMA).
- **Methods**:
  - trainModel(): Trains the model using historical energy usage.
  - predictUsage(): Predicts upcoming energy usage based on data.

5. EnergyUsage

- **Attributes**:
  - id: Integer – Record ID.
  - deviceId: Integer – Associated device's ID.
  - kWh: Float – Energy consumed in kilowatt-hours.
  - timestamp: Datetime – Timestamp when data is logged.
- **Methods**:
  - logUsage(): Logs new energy usage data.

6. SuggestionEngine

- **Methods**:

generateSuggestion(): Creates recommendations based on predictions.

optimizeEnergy(): Applies logic to reduce unnecessary consumption.

This class diagram encapsulates both the user-facing and backend components, clearly showing how the prediction model, data logging, and device control come together in the system.

5.Technology Description

This project integrates multiple technologies across frontend, backend, database, and machine learning components to achieve a real-time energy monitoring and prediction system. Below is an in-depth explanation of each technology used:

1. HTML (HyperText Markup Language)

HTML is the core language used to structure content on the web. It defines how data and interface elements like buttons, inputs, labels, tables, charts, etc., appear on the screen.

Basic HTML Tags Used

Tag	Purpose
<html>	Root element of the HTML document
<head>	Contains metadata, title, and script/style links
<title>	Sets the title of the web page tab
<body>	Contains all visible content of the page
<h1>–<h6>	Headings (<h1> = largest, <h6> = smallest)
<p>	Paragraph text
	Generic container to group elements
<button>	Defines a clickable button (e.g., Send Test Prediction)
<form>	Defines a user input form
<input>	Collects user input (text, numbers, etc.)
<label>	Labels input fields
 	Line break
<canvas>	Draws charts and graphs (used with Chart.js)
<script>	Includes JavaScript code
<style>	Includes internal CSS

Common HTML Attributes

Attribute	Description	Example
id	Unique identifier for an element	
class	Assigns a CSS class for styling or scripting	<button class="submit-btn">
type	Specifies type of input/button	<input type="text">
value	Sets the default value of input fields or buttons	<input value="Admin">
name	Name of input used for form submissions	<input name="email">
placeholder	Text shown inside input as a hint	<input placeholder="Enter data">
onclick	Executes JS function when clicked	<button onclick="submit()">
src	Specifies the source of external files (images, scripts, etc.)	<script src="main.js">
href	Used for links (<a>)	Home

Example from Your Project:

<button id="testPredictionBtn">Send Test Prediction</button>

<canvas id="monthlyStatsChart" height="200"></canvas>

- <button> creates a clickable button.
- id="testPredictionBtn" helps JS to find and add event to the button.
- <canvas> is used to draw graphs with Chart.js.

Usage in Project:

Creates the entire frontend interface including login page, dashboard, device status panel, and prediction controls.

5.2 Cascading Style Sheets (CSS)

CSS (Cascading Style Sheets) is a stylesheet language used to describe the presentation of HTML documents. It controls how HTML elements appear on the screen, including layout, colors, fonts, spacing, and visual effects.

It separates the content (HTML) from presentation (CSS), making web pages cleaner, reusable, and easier to maintain.

Purpose of CSS:

CSS is used to:

- Design attractive and professional interfaces
- Maintain consistency in design across all pages
- Make websites responsive for different devices
- Apply animations and transitions
- Control positioning, alignment, and visibility

Types of CSS:

1. **Inline CSS**

   Applied directly inside an HTML tag using the style attribute.

   Example:

```
<p style="color: red;">This is red text</p>
```

2. **Internal CSS**

   Placed within a <style> tag inside the <head> of the HTML document.

   Example:

```
<style>
  body { background-color: lightblue; }
</style>
```

3. **External CSS**

   Written in a separate .css file and linked via <link rel="stylesheet">.

   Example:

```
<link rel="stylesheet" href="style.css">
```

In your project, **Inline CSS** was mostly used due to single-file HTML format.

CSS Syntax:

```
selector {
  property: value;
}
```

Example:

```
h1 {
  color: blue;
  font-size: 28px;
}
```

Common CSS Selectors:

Selector	Description
*	Selects all elements
#id	Selects an element by ID (#header)
.class	Selects elements by class (.card)
Element	Selects all of that element (p, div)
element1, element2	Group selector (applies to both)
element element	Descendant selector

Common CSS Properties (Attributes):

Property	Purpose
Color	Text color
background-color	Background fill
font-size	Size of the text
font-family	Font style
Margin	Outer spacing
Padding	Inner spacing
Border	Adds border
width / height	Element size
Display	Layout behavior
Position	Positioning (absolute, relative)
text-align	Aligns text
box-shadow	Adds shadow to boxes
Transition	Smooth changes in properties

Advanced Properties (Used in Your Project):

Property	Purpose
flex, grid	Used for layout and structure
z-index	Stacks elements (used in modals/popups)
border-radius	Rounds corners
Overflow	Controls content overflow (scroll, hidden)
hover,active	Adds interactivity (ex: button color on hover)

Example: Dashboard Card Styling (Used in Your Project):

```
<div style="border: 1px solid #ccc; padding: 10px; margin: 10px; background-color: #f8f8f8;">
  <h3 style="color: #333;">Energy Usage Today</h3>
  <p style="font-size: 18px;">2.5 kWh</p>
</div>
```

Usage in Smart Home Energy Management System:

In your project, CSS is used to:

- Design dashboards, charts, and tables
- Style input fields, login panels, buttons
- Make the layout visually appealing and readable
- Highlight alerts and prediction results
- Maintain professional structure without using external files

5.3 JavaScript

JavaScript is a lightweight, interpreted programming language used to create dynamic and interactive effects within web browsers. In this project, JavaScript is used in the frontend to handle user actions, update the user interface in real-time, and communicate with the backend server using HTTP requests.

Purpose in This Project:

JavaScript helps your Smart Home Energy Management System in:

- Handling button clicks (e.g., "Send Test Prediction")
- Sending and receiving data using fetch()
- Updating charts (with Chart.js)
- Rendering dynamic dashboards

Important Features/Tags/Concepts Used:

1. Event Listeners

Used to trigger actions when a user interacts with the page (clicks, loads, etc.).

```
document.getElementById("btn").addEventListener("click", function () {
  alert("Button Clicked!");
});
```

2. DOM Manipulation

Used to change HTML elements dynamically.

```
document.getElementById("status").innerHTML = "Device is ON";
```

3. Fetch API

Used to communicate with the backend using GET, POST, etc.

```
fetch("http://localhost:5000/api/energy/record", {
  method: "POST",
  headers: { "Content-Type": "application/json" },
  body: JSON.stringify({ totalWattage: 1000 })
});
```

4. JSON Handling

Used for sending/receiving structured data.

```
const jsonData = JSON.stringify({ key: "value" });
```

5. Chart.js Integration

Used to draw bar and line charts in your dashboard.

```
new Chart(document.getElementById("myChart"), {
  type: "bar",
  data: {
    labels: ["Jan", "Feb"],
    datasets: [{
      label: "Energy (kWh)",
      data: [30, 45]
    }]
  }
});
```

5.4Node.js

Node.js is an open-source, cross-platform runtime environment that executes JavaScript code on the server side. It uses the event-driven, non-blocking I/O model, making it lightweight and efficient. In your project, Node.js powers the backend server, handles data APIs, and triggers machine learning predictions.

Purpose in This Project:

Node.js manages:

- APIs for energy data and device data
- Communication with MongoDB
- Running the ARIMA model via Python
- Sending prediction results to frontend

Key Features / Modules Used:

1. Express.js

A web framework to create API endpoints easily.

```
const express = require("express");
const app = express();
app.use(express.json());
```

2. CORS

Used to allow cross-origin requests (frontend to backend).

```
const cors = require("cors");
app.use(cors());
```

3. Mongoose (MongoDB ODM)

Used to define schemas and interact with MongoDB.

```
const mongoose = require("mongoose");
mongoose.connect("mongodb://localhost:27017/smartenergy");
```

4. API Routes (GET/POST)

To receive or send data to frontend.

```
app.post("/api/energy/record", async (req, res) => {
  // save data
});
```

5. Child Process (spawn)

To run Python scripts from Node.js.

```
const { spawn } = require("child_process");
const py = spawn("python", ["ml/predict.py"]);
```

6. Schema Definitions

Used to define database structure.

```
const EnergyData = mongoose.model("EnergyData", {
  timestamp: Date,
  totalWattage: Number
});
```

7. Error Handling

Used to return error messages properly.

```
try {
  // logic
} catch (err) {
  console.error(err);
  res.status(500).send("Server Error");
}
```

5.5 MongoDB – Database

MongoDB is a popular NoSQL database that stores data in flexible, JSON-like documents instead of traditional tables. It is schema-less, making it ideal for applications that require scalability, flexibility, and rapid development.

In the Smart Home Energy Management System, MongoDB is used to store and retrieve all energy usage data, device statuses, and prediction results. It allows seamless integration with Node.js through the Mongoose library.

Key Features:

- Document-oriented (uses BSON/JSON format)
- Scalable and high-performance
- Schema-less (no fixed structure)
- Easy to integrate with Node.js via Mongoose

Collections Used:

In this project, MongoDB contains the following **two key collections**:

1. energydatas:

Stores records of energy usage over time.

Field	Type	Description
_id	ObjectId	MongoDB auto-generated ID
timestamp	Date	Time when the usage was recorded
totalWattage	Number	Total energy consumption in watts

Example Document:


```
{
  "_id": "64db4f5c7a4b22a1f1e3d9c0",
  "timestamp": "2025-07-10T10:30:00.000Z",
  "totalWattage": 1234
}
```

2. devices

Stores information about each smart device.

Field	Type	Description
deviceId	String	Unique identifier for the device
label	String	Human-readable name (e.g., "TV")
wattage	Number	Power usage of the device in watts
status	String	Device state: "on" or "off"

Example Document:

json

CopyEdit

```
{
  "deviceId": "tv-1",
  "label": "Television",
  "wattage": 250,
  "status": "on"
}
```

Data Types Used:

- **String** – for device names and IDs
- **Number** – for wattage and predicted usage
- **Date** – to store timestamps
- **ObjectId** – automatically assigned by MongoDB to each document

Usage in Project:

- Record new energy usage (/api/energy/record)
- View monthly stats (/api/energy/monthly)
- Device-wise energy usage summaries
- Fetch historical data for ARIMA prediction
- Store prediction results

5.6 Chart.js

Chart.js is an open-source JavaScript library that allows you to easily draw interactive and visually appealing charts using the <canvas> HTML element.

Used in the dashboard to visualize:

Daily energy consumption

Device-wise energy usage

Monthly usage summary

Makes complex data easy to understand using pie charts, bar graphs, and line charts.

Key Chart.js Features Used:

Feature	Purpose
new Chart()	Initializes a chart instance
type	Defines the chart type: 'bar', 'line', etc.
data	Contains labels and datasets
options	Configures chart behavior (like tooltips)

Common Chart Types Used in Your Project:

Line Chart – For energy usage trends over time

Bar Chart – For device-wise energy usage comparison

Pie Chart – For distribution of energy among devices

5.7 Python

Python is a high-level, interpreted, general-purpose programming language known for its readability, simplicity, and vast library ecosystem. It supports object-oriented, procedural, and functional programming paradigms.

In your project, Python is used to run the ARIMA forecasting model (predict.py) to estimate future energy usage based on historical data stored in MongoDB.

Key Uses in Your Project:

- Data fetching from MongoDB
- Data preprocessing with Pandas
- Forecasting with ARIMA from statsmodels
- Returning prediction output to the backend

Common Python Constructs ("Tags"):

Statement/Function	**Purpose**
Import	Used to include external libraries like pandas, pymongo, statsmodels
Def	Defines a function
if __name__ == "__main__":	Entry point for execution
print()	Outputs data to console (used to return forecast result)
for / while	Loops through data
try-except	Handles runtime errors gracefully
With	Used to manage file/database connections
exit()	Stops script execution
range()	Generates a range of numbers for iteration
input()	Accepts user input (not used in your case)

Commonly Used Data Types:

Data Type	**Description**	**Example**
int	Integer numbers	x = 10
float	Decimal numbers	y = 2.55
str	Textual data	"Smart Home"
bool	Boolean value: True or False	status = True
list	Ordered collection of items	[1, 2, 3]
dict	Key-value pair collection	{"name": "AC"}
datetime	Date and time handling	datetime.now()
DataFrame	Tabular data structure from Pandas	pd.DataFrame()

Key Libraries Used in Your Project:

Library	**Purpose**
pandas	Reading, cleaning, and transforming data from MongoDB
pymongo	Connecting and retrieving data from MongoDB
statsmodels	Building ARIMA model for forecasting
sys	Reading command-line arguments (if needed)

6.code:

Frontend code:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Smart Home Energy Management System</title>
  <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.0.0-
beta3/css/all.min.css">
  <script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
  <script src="https://cdn.jsdelivr.net/npm/chartjs-plugin-zoom"></script>
<script src="https://cdn.jsdelivr.net/npm/axios/dist/axios.min.js"></script>
  <style>
    :root {
      --primary-color: #4CAF50;
      --secondary-color: #45a049;
      --accent-color: #3e8e41;
      --dark-color: #333;
      --light-color: #f4f4f9;
      --warning-color: #ff9800;
      --danger-color: #f44336;
    }
    body {
      font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
      margin: 0;
      padding: 0;
      background-color: var(--light-color);
      color: var(--dark-color);
      line-height: 1.6;
    }
    header {
      background-color: var(--primary-color);
      color: white;
      padding: 1.5rem;
      text-align: center;
      box-shadow: 0 2px 5px rgba(0,0,0,0.1);
    }
    nav ul {
      list-style: none;
      padding: 0;
      display: flex;
      justify-content: center;
      gap: 2rem;
      margin-top: 1rem;
    }
    nav ul li a {
      color: white;
      text-decoration: none;
      font-weight: bold;
```

```
  padding: 0.5rem 1rem;
  border-radius: 4px;
  transition: all 0.3s ease;
}
nav ul li a:hover, nav ul li a.active {
  background-color: rgba(255,255,255,0.2);
}
main {
  padding: 2rem;
  max-width: 1200px;
  margin: 0 auto;
}
.dashboard-grid {
  display: grid;
  grid-template-columns: repeat(auto-fit, minmax(300px, 1fr));
  gap: 2rem;
  margin-bottom: 2rem;
}
section {
  background-color: white;
  border-radius: 8px;
  padding: 1.5rem;
  box-shadow: 0 2px 10px rgba(0,0,0,0.05);
  margin-bottom: 2rem;
}
h2 {
  color: var(--primary-color);
  border-bottom: 2px solid var(--primary-color);
  padding-bottom: 0.5rem;
  margin-top: 0;
  display: flex;
  align-items: center;
  gap: 0.5rem;
}
h2 i {
  font-size: 1.2em;
}
.chart-container {
  position: relative;
  height: 300px;
  margin: 1rem 0;
}
.card {
  background-color: white;
  border-radius: 8px;
  padding: 1.5rem;
  box-shadow: 0 2px 10px rgba(0,0,0,0.05);
  margin-bottom: 1rem;
}
.card h3 {
  margin-top: 0;
  color: var(--primary-color);
```

```
}
#suggestions-list {
  list-style-type: none;
  padding: 0;
}
#suggestions-list li {
  padding: 0.75rem;
  margin-bottom: 0.5rem;
  background-color: #f8f9fa;
  border-left: 4px solid var(--primary-color);
  border-radius: 4px;
  display: flex;
  align-items: center;
  gap: 0.75rem;
}
#suggestions-list li i {
  color: var(--primary-color);
}
.suggestion-critical {
  border-left-color: var(--danger-color);
  background-color: #fdecea;
}
.suggestion-critical i {
  color: var(--danger-color);
}
.suggestion-warning {
  border-left-color: var(--warning-color);
  background-color: #fff8e1;
}
.suggestion-warning i {
  color: var(--warning-color);
}
form {
  max-width: 500px;
  margin: 0 auto;
  display: flex;
  flex-direction: column;
  gap: 1rem;
}
.form-group {
  display: flex;
  flex-direction: column;
  gap: 0.5rem;
}
form label {
  font-weight: bold;
  color: var(--dark-color);
}
form input, form select, form textarea {
  padding: 0.75rem;
  font-size: 1rem;
```

```
  border: 1px solid #ddd;

  border-radius: 4px;
  transition: border 0.3s ease;
}
form input:focus, form select:focus, form textarea:focus {
  border-color: var(--primary-color);
  outline: none;
  box-shadow: 0 0 0 2px rgba(76, 175, 80, 0.2);
}
form button {
  padding: 0.75rem;
  background-color: var(--primary-color);
  color: white;
  border: none;
  border-radius: 4px;
  cursor: pointer;
  font-weight: bold;
  transition: all 0.3s ease;
}
form button:hover {
  background-color: var(--secondary-color);
}
.devices-grid {
  display: grid;
  grid-template-columns: repeat(auto-fill, minmax(250px, 1fr));
  gap: 1rem;
  margin-top: 1rem;
}
.device-card {
  background-color: white;
  border-radius: 8px;
  padding: 1rem;
  box-shadow: 0 2px 5px rgba(0,0,0,0.1);
  border-left: 4px solid var(--primary-color);
  transition: transform 0.3s ease;
}
.device-card:hover {
  transform: translateY(-3px);
}
.device-card h3 {
  margin-top: 0;
  display: flex;
  justify-content: space-between;
  align-items: center;
}
.device-status {
  display: inline-block;
  padding: 0.25rem 0.5rem;
  border-radius: 4px;
  font-size: 0.8rem;
  font-weight: bold;
```

```
}


.status-on {
  background-color: #e8f5e9;
  color: var(--primary-color);
}
.status-off {
  background-color: #ffebee;
  color: var(--danger-color);
}
.device-details {
  margin-top: 0.5rem;
  font-size: 0.9rem;
  color: #666;
}
.device-actions {
  display: flex;
  gap: 0.5rem;
  margin-top: 1rem;
}
.btn {
  padding: 0.5rem 1rem;
  border-radius: 4px;
  border: none;
  cursor: pointer;
  font-weight: bold;
  transition: all 0.3s ease;
}
.btn-primary {
  background-color: var(--primary-color);
  color: white;
}
.btn-primary:hover {
  background-color: var(--secondary-color);
}
.btn-danger {
  background-color: var(--danger-color);
  color: white;
}
.btn-danger:hover {
  background-color: #d32f2f;
}
footer {
  background-color: var(--dark-color);
  color: white;
  text-align: center;
  padding: 1.5rem;
  margin-top: 2rem;
}
.energy-stats {
  display: grid;
```

```
  grid-template-columns: repeat(auto-fit, minmax(200px, 1fr));
  gap: 1rem;


  margin-bottom: 2rem;
}

.stat-card {
  background-color: white;
  border-radius: 8px;
  padding: 1.5rem;
  box-shadow: 0 2px 10px rgba(0,0,0,0.05);
  text-align: center;
}
.stat-card h3 {
  margin-top: 0;
  color: #666;
  font-size: 1rem;
}
.stat-card .value {
  font-size: 2rem;
  font-weight: bold;
  color: var(--primary-color);
  margin: 0.5rem 0;
}
.stat-card .unit {
  color: #666;
  font-size: 0.9rem;
}
.comparison-container {
  display: flex;
  gap: 1rem;
  margin-top: 1rem;
}
.comparison-badge {
  display: flex;
  align-items: center;
  gap: 0.25rem;
  font-size: 0.9rem;
}
.comparison-up {
  color: var(--danger-color);
}
.comparison-down {
  color: var(--primary-color);
}
@media (max-width: 768px) {
  nav ul {
    flex-direction: column;
    gap: 0.5rem;
  }
```

```
    .dashboard-grid {
      grid-template-columns: 1fr;



    }
    .energy-stats {
      grid-template-columns: 1fr 1fr;
    }
  }
  /* Login System Styles */
#login-overlay {
  position: fixed;
  top: 0;
  left: 0;
  width: 100%;
  height: 100%;
  background: rgba(0,0,0,0.9);
  display: flex;
  justify-content: center;
  align-items: center;
  z-index: 9999;
}

.login-box {
  background: white;
  padding: 2rem;
  border-radius: 8px;
  width: 300px;
}

.login-box h2 {
  color: var(--primary-color);
  margin-top: 0;
}

.login-box input, .login-box select {
  width: 100%;
  padding: 10px;
  margin: 8px 0;
  border: 1px solid #ddd;
  border-radius: 4px;
}

.login-box button {
  width: 100%;
  padding: 10px;
  background: var(--primary-color);
  color: white;
  border: none;
  border-radius: 4px;
  cursor: pointer;
```

```
  margin-top: 10px;
}


#login-message {
  color: var(--danger-color);
  margin-top: 10px;
  height: 16px;
}

#user-info {
  position: absolute;
  top: 20px;
  right: 20px;
  color: white;
  display: none;
  align-items: center;
  gap: 10px;
}

#logout-btn {
  background: var(--danger-color);
  color: white;
  border: none;
  padding: 5px 10px;
  border-radius: 4px;
  cursor: pointer;
}

  </style>
</head>
<body>
  <!-- Login Overlay -->
<div id="login-overlay">
  <div class="login-box">
    <h2>Smart Energy Login</h2>
    <select id="login-role">
      <option value="homeowner">Homeowner</option>
      <option value="admin" selected>Admin</option>
    </select>
    <input type="text" id="login-username" placeholder="Username" value="admin">
    <input type="password" id="login-password" placeholder="Password">
    <button id="login-btn">Login</button>
    <div id="login-message"></div>
  </div>
</div>

<!-- User Info -->
<div id="user-info">
  <span id="current-user">Welcome, Admin</span>
  <button id="logout-btn">Logout</button>
</div>
```

```
<header>
  <h1><i class="fas fa-home"></i> Smart Home Energy Management System</h1>
  <nav>


    <ul>
      <li><a href="#dashboard" class="active"><i class="fas fa-tachometer-alt"></i>
Dashboard</a></li>
      <li><a href="#predictions"><i class="fas fa-chart-line"></i> Predictions</a></li>
      <li><a href="#optimization"><i class="fas fa-lightbulb"></i> Optimization</a></li>
      <li><a href="#admin"><i class="fas fa-cog"></i> Admin Panel</a></li>
    </ul>
  </nav>
</header>

<main>
  <!-- Dashboard Section -->
  <section id="dashboard">
    <h2><i class="fas fa-tachometer-alt"></i> Energy Usage Dashboard</h2>

    <div class="energy-stats">
      <div class="stat-card">
        <h3>Current Usage</h3>
        <div class="value">1.25</div>
        <div class="unit">kW</div>
        <div class="comparison-container">
          <span class="comparison-badge comparison-up">
            <i class="fas fa-arrow-up"></i> 12% from yesterday
          </span>
        </div>
      </div>

      <div class="stat-card" id="live-cost-card">
        <h3>Real-Time Cost</h3>
        <div class="value" id="live-cost-value">₹0.00</div>
        <div class="unit">per hour</div>
      </div>

      <div class="stat-card">
        <h3>Today's Consumption</h3>
        <div class="value">8.7</div>
        <div class="unit">kWh</div>
        <div class="comparison-container">
          <span class="comparison-badge comparison-up">
            <i class="fas fa-arrow-up"></i> 5% from average
          </span>
        </div>
      </div>

      <div class="stat-card">
        <h3>Monthly Consumption</h3>
        <div class="value">265</div>
```

```
        <div class="unit">kWh</div>
        <div class="comparison-container">
          <span class="comparison-badge comparison-down">
            <i class="fas fa-arrow-down"></i> 8% from last month


          </span>
        </div>
      </div>

      <div class="stat-card">
        <h3>Estimated Cost</h3>
        <div class="value">₹3,299</div>
        <div class="unit">this month</div>
        <div class="comparison-container">
          <span class="comparison-badge comparison-down">
            <i class="fas fa-arrow-down"></i> ₹311 saved
          </span>
        </div>
      </div>
    </div>

    <div class="chart-container">
      <canvas id="energyUsageChart"></canvas>
    </div>

    <div class="card">
      <h3>Active Devices</h3>
      <div class="card">

      </div>

      <div class="devices-grid" id="dashboard-devices">
        <!-- Devices will be dynamically inserted here -->
      </div>
    </div>
  </section>

  <!-- Predictions Section -->
  <section id="predictions">
    <h2><i class="fas fa-chart-line"></i> Energy Consumption Predictions</h2>
    <div class="chart-container">
      <canvas id="energyPredictionChart"></canvas>
    </div>

    <div class="card">
      <h3>Prediction Insights</h3>
      <p>Based on your usage patterns and weather forecasts, we predict higher energy
consumption in the upcoming summer months due to increased AC usage. Consider these
adjustments to optimize your energy use:</p>
      <ul id="prediction-insights">
        <li><i class="fas fa-check-circle"></i> Pre-cool your home during off-peak hours
```

```
(before 2 PM)</li>
        <li><i class="fas fa-check-circle"></i> Use ceiling fans to complement your AC</li>
        <li><i class="fas fa-check-circle"></i> Schedule regular AC maintenance for optimal
efficiency</li>
      </ul>


    </div>
  </section>

  <!-- Optimization Suggestions Section -->
  <section id="optimization">
    <h2><i class="fas fa-lightbulb"></i> Energy Optimization Suggestions</h2>
    <ul id="suggestions-list">
      <li class="suggestion-critical">
        <i class="fas fa-exclamation-circle"></i>
        <span>Living Room AC has been running continuously for 4 hours. Consider turning it
off or increasing the temperature by 2°F.</span>
      </li>
      <li class="suggestion-warning">
        <i class="fas fa-exclamation-triangle"></i>
        <span>Kitchen lights are on during daylight hours. Switch to natural light or install
motion sensors.</span>
      </li>
      <li>
        <i class="fas fa-info-circle"></i>
        <span>Your refrigerator is 8 years old. Replacing it with an ENERGY STAR model
could save you ₹5,810/year.</span>
      </li>
      <li>
        <i class="fas fa-info-circle"></i>
        <span>15% of your energy is used during peak rate hours (4-9 PM). Shift non-essential
usage to other times.</span>
      </li>
      <li>
        <i class="fas fa-info-circle"></i>
        <span>Consider installing solar panels. Based on your roof size, you could offset 65%
of your energy needs.</span>
      </li>
      <li class="suggestion-warning">
        <i class="fas fa-exclamation-triangle"></i>
        <span>3 devices are in standby mode, consuming 45W continuously. Use smart plugs
to completely power them off.</span>
      </li>
    </ul>

    <div class="card">
      <h3>Potential Savings</h3>
      <p>Implementing these suggestions could save you approximately
<strong>₹1,826/month</strong> or <strong>₹21,912/year</strong> on your energy bills.</p>
      <button class="btn btn-primary"><i class="fas fa-calendar-alt"></i> Schedule
Optimization Plan</button>
```

```
  </div>
</section>

<!-- Admin Panel Section -->
<section id="admin">
  <h2><i class="fas fa-cog"></i> Admin Panel</h2>


  <button onclick="sendPrediction()" style="
  padding: 10px 20px;
  font-size: 16px;
  font-weight: bold;
  color: white;
  background-color: #28a745;
  border: none;
  border-radius: 8px;
  cursor: pointer;
  transition: background-color 0.3s ease;
">Send Test Prediction</button>


  <div class="dashboard-grid">
    <div class="card">
      <h3>Add New Device</h3>
      <form id="add-device-form">
        <div class="form-group">
          <label for="device-name">Device Name:</label>
          <input type="text" id="device-name" name="device-name" required>
        </div>

        <div class="form-group">
          <label for="device-type">Device Type:</label>
          <select id="device-type" name="device-type" required>
            <option value="">Select type</option>
            <option value="lighting">Lighting</option>
            <option value="appliance">Appliance</option>
            <option value="hvac">HVAC</option>
            <option value="electronics">Electronics</option>
            <option value="other">Other</option>
          </select>
        </div>

        <div class="form-group">
          <label for="device-wattage">Wattage (W):</label>
          <input type="number" id="device-wattage" name="device-wattage" required>
        </div>

        <div class="form-group">
          <label for="device-location">Location:</label>
          <input type="text" id="device-location" name="device-location">
        </div>
```

```
          <div class="form-group">
            <label for="device-notes">Notes:</label>
            <textarea id="device-notes" name="device-notes" rows="3"></textarea>
          </div>

          <button type="submit" class="btn btn-primary"><i class="fas fa-plus"></i> Add
Device</button>


        </form>
      </div>

      <div class="card">
        <h3>System Settings</h3>
        <form id="system-settings-form">
          <div class="form-group">
            <label for="energy-rate">Energy Rate (₹/kWh):</label>
            <input type="number" id="energy-rate" name="energy-rate" step="0.01"
value="12.45" required>
          </div>

          <div class="form-group">
            <label for="peak-hours">Peak Hours:</label>
            <select id="peak-hours" name="peak-hours" required>
              <option value="4-9">4 PM - 9 PM</option>
              <option value="3-8">3 PM - 8 PM</option>
              <option value="5-10">5 PM - 10 PM</option>
              <option value="none">No Peak Hours</option>
            </select>
          </div>

          <div class="form-group">
            <label for="notifications">Notifications:</label>
            <select id="notifications" name="notifications" required>
              <option value="enabled">Enabled</option>
              <option value="disabled">Disabled</option>
            </select>
          </div>

          <div class="form-group">
            <label>
             <input type="checkbox" id="energy-alerts" name="energy-alerts"> Enable energy spike alerts
            </label>
          </div>

          <button type="submit" class="btn btn-primary"><i class="fas fa-save"></i> Save Settings</button>
        </form>
      </div>
    </div>

    <div class="card">
      <h3>Device Management</h3>
      <div class="devices-grid" id="device-manager">
```

```
        <!-- Devices will be dynamically inserted here -->
       </div>
     </div>
   </section>
  </main>

  <footer>


    <p>&copy; 2025 Smart Home Energy Management System | <a href="#" style="color:
white;">Privacy Policy</a> | <a href="#" style="color: white;">Terms of Service</a></p>
  </footer>

  <script>
    // Login System
const users = {
  homeowner: { username: "home", password: "home", name: "Homeowner" },
  admin: { username: "admin", password: "admin", name: "Administrator" }
};

document.getElementById('login-btn').addEventListener('click', function() {
  const role = document.getElementById('login-role').value;
  const username = document.getElementById('login-username').value;
  const password = document.getElementById('login-password').value;
  const messageEl = document.getElementById('login-message');

  messageEl.textContent = '';

  if (users[role] && username === users[role].username && password ===
users[role].password) {
    // Successful login
    document.getElementById('login-overlay').style.display = 'none';
    document.getElementById('user-info').style.display = 'flex';
    document.getElementById('current-user').textContent = `Welcome, ${users[role].name}`;

    // Toggle Admin Panel based on role
    const adminTab = document.querySelector('nav ul li:nth-child(4)');
    const adminSection = document.getElementById('admin');

    if (role === 'admin') {
      adminTab.style.display = 'block';
      adminSection.style.display = 'block';
    } else {
      adminTab.style.display = 'none';
      adminSection.style.display = 'none';
    }
  } else {
    messageEl.textContent = 'Incorrect password';
  }
});

document.getElementById('logout-btn').addEventListener('click', function() {
```

```
  // Reset UI
  document.getElementById('login-overlay').style.display = 'flex';
  document.getElementById('user-info').style.display = 'none';
  document.getElementById('login-password').value = '';

  // Restore Admin Panel for next login
  document.querySelector('nav ul li:nth-child(4)').style.display = 'block';
  document.getElementById('admin').style.display = 'block';
});

// Initialize
document.addEventListener('DOMContentLoaded', function() {
  document.getElementById('login-overlay').style.display = 'flex';
  document.getElementById('user-info').style.display = 'none';
});
    // Add this to your frontend code (before other scripts)
const API_URL = "http://localhost:5000/api";

// 1. Load Devices on Startup
async function loadDevices() {
  try {
    const response = await fetch(`${API_URL}/status`);
    const data = await response.json();
    console.log("Connection Status:", data);

    // Update your UI here (replace with your actual rendering function)
    document.getElementById('status').innerText = `Backend: ${data.status}, DB:
${data.database}`;
  } catch (err) {
    console.error("Connection Failed:", err);
  }
}

// 2. Test Data Submission (Add to Admin Panel)
async function testAddDevice() {
  await fetch(`${API_URL}/energy/predict`, {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({
      actualUsage: 2.4,
      devices: [{
  deviceId: "1",
  power: 500,
  duration: 2,
  label: "Washing Machine - 30 min Cycle"
}]

    })
  });
  alert("Test data sent!");
}
```

```
async function loadMonthlyStats() {
  try {
    const res = await fetch('http://localhost:5000/api/energy/summary/monthly');
    const stats = await res.json();

    if (!stats || stats.days === 0) {
      return alert("📉 No usage data this month.");
    }


alert(` Monthly Stats:
Total: ${stats.total} kWh
Average per Day: ${stats.avgPerDay} kWh
Max in a Day: ${stats.maxUsage} kWh
Days recorded: ${stats.days}`);
  } catch (err) {
    console.error("Monthly stats fetch error:", err);
    alert(" Failed to load monthly stats.");
  }
}

// Initialize
document.addEventListener('DOMContentLoaded', loadDevices);
    // Device storage array
    let devices = [
      {
        id: 1,
        name: "Living Room AC",
        type: "hvac",
        wattage: 1200,
        location: "Living Room",
        status: "on",
        notes: "Main cooling unit"
      },
      {
        id: 2,
        name: "Kitchen Lights",
        type: "lighting",
        wattage: 300,
        location: "Kitchen",
        status: "on",
        notes: "LED strip lights"
      },
      {
        id: 3,
        name: "Refrigerator",
        type: "appliance",
        wattage: 150,
        location: "Kitchen",
        status: "on",
        notes: "10-year old model"
      },
```

```
    {
      id: 4,
      name: "Home Office",
      type: "electronics",
      wattage: 800,
      location: "Office",
      status: "off",
      notes: "Computer and peripherals"
    }
  ];

  // Function to render devices in both dashboard and device manager
  function renderDevices() {
    const dashboardGrid = document.querySelector('#dashboard-devices');
    const deviceManagerGrid = document.querySelector('#device-manager');

    dashboardGrid.innerHTML = '';
    deviceManagerGrid.innerHTML = '';

    devices.forEach(device => {
      const deviceCard = document.createElement('div');
      deviceCard.className = 'device-card';
      deviceCard.innerHTML = `
        <h3>${device.name} <span class="device-status status-
${device.status}">${device.status.toUpperCase()}</span></h3>
        <div class="device-details">
          <div><i class="fas fa-bolt"></i> ${device.wattage}W</div>
          <div><i class="fas fa-home"></i> ${device.location}</div>
          <div><i class="fas fa-tag"></i> ${device.type.charAt(0).toUpperCase() +
device.type.slice(1)}</div>
          ${device.notes ? `<div><i class="fas fa-sticky-note"></i> ${device.notes}</div>` : ''}
        </div>
        <div class="device-actions">
          <button class="btn btn-primary" onclick="toggleDeviceStatus(${device.id})">
            <i class="fas fa-power-off"></i> Turn ${device.status === 'on' ? 'Off' : 'On'}
          </button>
        </div>
      `;

      // Add to both dashboard and device manager
      dashboardGrid.appendChild(deviceCard.cloneNode(true));
      deviceManagerGrid.appendChild(deviceCard);
    });
  }

  // Device form submission
  document.getElementById('add-device-form').addEventListener('submit', function(event) {
    event.preventDefault();

    const newDevice = {
      id: devices.length > 0 ? Math.max(...devices.map(d => d.id)) + 1 : 1,
      name: document.getElementById('device-name').value,
```

```
        type: document.getElementById('device-type').value,
        wattage: parseInt(document.getElementById('device-wattage').value),
        location: document.getElementById('device-location').value,
        status: 'on', // default to on when added
        notes: document.getElementById('device-notes').value
      };

      devices.push(newDevice);
      renderDevices();



    // Show success message
      alert(`Device "${newDevice.name}" added successfully!`);

      // Reset form
      this.reset();
    });

    // Toggle device status
    function toggleDeviceStatus(deviceId) {
      const device = devices.find(d => d.id === deviceId);
      device.status = device.status === 'on' ? 'off' : 'on';
      renderDevices();
    }
// Add this directly below
function updateLiveCost() {
  const energyRate = parseFloat(document.getElementById('energy-rate')?.value || 12.45); //
₹/kWh
  const activeDevices = devices.filter(d => d.status === 'on');
  const totalWatts = activeDevices.reduce((sum, d) => sum + d.wattage, 0);
  const kWh = totalWatts / 1000;
  const cost = kWh * energyRate;

  document.getElementById('live-cost-value').textContent = `₹${cost.toFixed(2)}`;
}
    // System settings form
    document.getElementById('system-settings-form').addEventListener('submit',
function(event) {
      event.preventDefault();
      const energyRate = document.getElementById('energy-rate').value;
      const peakHours = document.getElementById('peak-hours').value;

      alert(`System settings updated!\nEnergy Rate: ₹${energyRate}/kWh\nPeak Hours:
${peakHours}`);
    });

    // Chart.js configuration for energy usage (6 months)
    const energyUsageCtx = document.getElementById('energyUsageChart').getContext('2d');
    const energyUsageChart = new Chart(energyUsageCtx, {
      type: 'line',
      data: {
```

```
      labels: ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun'],
      datasets: [{
        label: 'Energy Usage (kWh)',
        data: [320, 290, 310, 280, 260, 275],
        borderColor: 'rgba(75, 192, 192, 1)',
        backgroundColor: 'rgba(75, 192, 192, 0.1)',
        borderWidth: 2,
        fill: true,
        tension: 0.3
      },
```

```
{
        label: 'Target Usage',
        data: [300, 280, 290, 270, 250, 260],
        borderColor: 'rgba(255, 99, 132, 1)',
        backgroundColor: 'rgba(255, 99, 132, 0.1)',
        borderWidth: 2,
        borderDash: [5, 5],
        fill: true,
        tension: 0.3
      }]
    },
    options: {
      responsive: true,
      plugins: {
        zoom: {
          zoom: {
            wheel: {
              enabled: true,
            },
            pinch: {
              enabled: true
            },
            mode: 'xy',
          },
          pan: {
            enabled: true,
            mode: 'xy',
          }
        },
        tooltip: {
          mode: 'index',
          intersect: false,
        },
        legend: {
          position: 'top',
        }
      },
      scales: {
        y: {
```

```
          beginAtZero: false,
          title: {
            display: true,
            text: 'Energy Usage (kWh)'
          }
        },
        x: {
          title: {
            display: true,
            text: 'Month'
          }
        }
      },

      interaction: {
        mode: 'nearest',
        axis: 'x',
        intersect: false

      }
    }
  });

  // Chart.js configuration for energy predictions (6 months)
  const energyPredictionCtx =
document.getElementById('energyPredictionChart').getContext('2d');
  const energyPredictionChart = new Chart(energyPredictionCtx, {
    type: 'bar',
    data: {
      labels: ['Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec'],
      datasets: [{
        label: 'Predicted Energy Usage (kWh)',
        data: [310, 340, 290, 270, 250, 280],
        backgroundColor: 'rgba(153, 102, 255, 0.6)',
        borderColor: 'rgba(153, 102, 255, 1)',
        borderWidth: 1,
      }, {
        label: 'Potential Savings',
        data: [280, 310, 260, 240, 220, 250],
        backgroundColor: 'rgba(75, 192, 192, 0.6)',
        borderColor: 'rgba(75, 192, 192, 1)',
        borderWidth: 1,
      }]
    },
    options: {
      responsive: true,
      plugins: {
        zoom: {
          zoom: {
            wheel: {
              enabled: true,
            },
```

```
            pinch: {
              enabled: true
            },
            mode: 'xy',
          },
          pan: {
            enabled: true,
            mode: 'xy',
          }
        },
        tooltip: {
          mode: 'index',
          intersect: false,
        },

        legend: {
          position: 'top',
        }
      },
scales: {
        y: {
          beginAtZero: true,
          title: {
            display: true,
            text: 'Energy Usage (kWh)'
          }
        },
        x: {
          title: {
            display: true,
            text: 'Month'
          }
        }
      },
      interaction: {
        mode: 'nearest',
        axis: 'x',
        intersect: false
      }
    }
  });

  // Navigation active state
  document.querySelectorAll('nav ul li a').forEach(link => {
    link.addEventListener('click', function() {
      document.querySelectorAll('nav ul li a').forEach(el => el.classList.remove('active'));
      this.classList.add('active');
    });
  });

  // Simulate real-time data updates
  setInterval(() => {
```

```
    // Update current usage with random fluctuation
    const currentUsageElement = document.querySelector('.energy-stats .value');
    const currentUsage = parseFloat(currentUsageElement.textContent);
    const fluctuation = (Math.random() * 0.3 - 0.15).toFixed(2);
    const newUsage = Math.max(0.5, (currentUsage + parseFloat(fluctuation)).toFixed(2));
    currentUsageElement.textContent = newUsage;

    // Update comparison badge color based on usage
    const comparisonBadge = document.querySelector('.energy-stats .comparison-badge');
    if (newUsage > 1.3) {
      comparisonBadge.className = 'comparison-badge comparison-up';
      comparisonBadge.innerHTML = '<i class="fas fa-arrow-up"></i> ' + (Math.random() *
15 + 5).toFixed(0) + '% from yesterday';
    } else {


 comparisonBadge.className = 'comparison-badge comparison-down';
      comparisonBadge.innerHTML = '<i class="fas fa-arrow-down"></i> ' + (Math.random()
* 10 + 5).toFixed(0) + '% from yesterday';
    }
   }, 5000);

   // Initialize the page
   document.addEventListener('DOMContentLoaded', function() {
    renderDevices();
    loadPredictions(); // <--- this line loads data into chart
    loadDeviceUsage();
    // ⏱️ Auto logging every 10 seconds
    setInterval(autoLogEnergyUsage, 10000); // log energy
  setInterval(() => {
   loadPredictions();
   loadDeviceUsage();
  }, 10000); // refresh charts
  setInterval(updateLiveCost, 5000); // Update cost every 5 seconds
   });
// Replace old connection code with this:
const mongoose = require('mongoose');
mongoose.connect('mongodb://localhost:27017/yourdbname')
  .then(() => console.log('Connected to MongoDB'))
  .catch(err => console.error('MongoDB connection error:', err));
const API_BASE_URL = 'http://localhost:5000/api'; // Replace if needed
// Fetch predictions
async function loadPredictions() {
  try {
    const response = await fetch("http://localhost:5000/api/energy/history");
    const data = await response.json();

    const labels = data.map((entry, index) => `Entry ${index + 1}`);
    const actual = data.map(entry => entry.actualUsage);
    const predicted = data.map(entry => entry.predictedUsage);

    energyPredictionChart.data.labels = labels;
```

```
    energyPredictionChart.data.datasets[0].data = predicted;   // Predicted Energy
    energyPredictionChart.data.datasets[1].data = actual;      // Actual Energy

    energyPredictionChart.update();
  } catch (error) {
    console.error("Error loading predictions:", error);
  }
}

  // Update Chart.js
  energyChart.data.labels = data.map(d => new Date(d.timestamp).toLocaleTimeString());
  energyChart.data.datasets[0].data = data.map(d => d.actualUsage);
  energyChart.data.datasets[1].data = data.map(d => d.predictedUsage);
  energyChart.update();


// Send new data (e.g., from smart devices)
async function logEnergy(usage, devices) {
  const prediction = await fetch('http://localhost:5000/api/energy/predict', {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({
      actualUsage: usage,
      predictedUsage: null, // Let backend calculate
      devices: devices.map(d => ({
        deviceId: d.id,
        power: d.wattage,
        duration: d.hoursUsed
      }))
    })
  });
}
async function refreshDevices() {
  const devices = await (await fetch(`${API_URL}/devices`)).json();
  // Render devices to your HTML table
}
setInterval(refreshDevices, 5000); // Update every 5 seconds

//TEST BUTTON FUNCTION TO SEND PREDICTION DATA
async function testAddDevice() {
  try {
    const response = await fetch('http://localhost:5000/api/energy/predict', {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify({
        actualUsage: 2.4,
        devices: [
          { deviceId: "test-1", power: 500, duration: 2 }
        ]
      })
    });
    const result = await response.json();
```

```
    alert(`Test data sent!\nPredicted usage: ${result.predictedUsage} kWh`);
  } catch (error) {
    console.error("Error sending test device:", error);
    alert("Failed to send test data.");
  }
}
setInterval(() => {
  loadPredictions();  // auto-update the chart
}, 10000);

async function loadDeviceUsage() {
  try {
    const res = await fetch('http://localhost:5000/api/energy/summary/device-usage');
    const usage = await res.json();
    const labels = Object.keys(usage);
    const values = Object.values(usage);

    new Chart(document.getElementById('deviceUsageChart').getContext('2d'), {
      type: 'bar',
      data: {
        labels,
        datasets: [{
          label: 'Device Energy Usage (kWh)',
          data: values,
          backgroundColor: 'rgba(255, 159, 64, 0.6)',
          borderColor: 'rgba(255, 159, 64, 1)',
          borderWidth: 1
        }]
      },
      options: {
        responsive: true,
        scales: {
          y: {
            beginAtZero: true,
            title: {
              display: true,
              text: 'kWh'
            }
          }
        }
      }
    });
  } catch (err) {
    console.error("Failed to load device usage:", err);
  }
}
async function autoLogEnergyUsage() {
  const activeDevices = devices.filter(d => d.status === 'on');
  if (activeDevices.length === 0) return;

  const actualUsage = activeDevices.reduce((sum, d) => sum + (d.wattage * 1) / 1000, 0); //
assume 1 hour for demo
```

```
  const devicesPayload = activeDevices.map(d => ({
    deviceId: d.id,
    power: d.wattage,
    duration: 1 // fake 1 hour for demo
  }));
  await fetch('http://localhost:5000/api/energy/predict', {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({
      actualUsage,
      devices: devicesPayload
    })
  });
  console.log("Auto-logged energy usage:", actualUsage);
}


function sendTestPrediction() {
    fetch('/api/predict')
      .then(res => res.json())
      .then(data => {
        alert('Predicted Value: ' + data.predictedValue);
        console.log('Predicted Value:', data.predictedValue);
      })
      .catch(err => {
        alert('Failed to send prediction');
        console.error(err);
      });
  }
  function sendPrediction() {
  console.log(' Send Test Prediction clicked');
  fetch('http://localhost:5000/predict', {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json'
    },
    body: JSON.stringify({})
  })
  .then(res => {
    if (!res.ok) throw new Error('Server error');
    return res.json();
  })
  .then(data => {
    alert('Prediction: ' + data.predicted + 'KWh');
  })
  .catch(err => {
    alert(' Failed to send prediction');
    console.error(err);
  });
}
</script>
```

```
</body>
</html>
```

Backend code:
Mechine learning:predict.py

```
import pandas as pd
from pymongo import MongoClient
from statsmodels.tsa.arima.model import ARIMA
import warnings

# Ignore all warnings (this is the fix!)
warnings.filterwarnings("ignore")

client = MongoClient("mongodb://localhost:27017/")
db = client["smartenergy"]
collection = db["energydatas"]

records = list(collection.find().sort("timestamp", -1).limit(30))

if not records or len(records) < 5:
    print(2.55)
    exit()

df = pd.DataFrame(records)
df['timestamp'] = pd.to_datetime(df['timestamp'])
df = df.sort_values('timestamp')
df.set_index('timestamp', inplace=True)

model = ARIMA(df['totalWattage'], order=(1, 1, 1))
model_fit = model.fit()
forecast = model_fit.forecast(steps=1)

print(round(float(forecast.iloc[0]), 2))
```

Models:
EnergyData.js

```
const mongoose = require('mongoose');
const energySchema = new mongoose.Schema({
  timestamp: { type: Date, default: Date.now },
  actualUsage: Number,
  predictedUsage: Number,
  devices: [
    {
      deviceId: String,
      power: Number,
      duration: Number
    }
```

```
  ]
});

module.exports = mongoose.model('EnergyData', energySchema);
```

EnergyUsage.js

```
const mongoose = require('mongoose');
const energyUsageSchema = new mongoose.Schema({
  date: Date,
  usage: Number,
  device: String
});

module.exports = mongoose.model('EnergyUsage', energyUsageSchema);
```

Server.js:

```
const express = require('express');
const mongoose = require('mongoose');
const bodyParser = require('body-parser');
const cors = require('cors');
const app = express();

// Middleware
app.use(cors());
app.use(bodyParser.json());

// MongoDB connection
mongoose.connect('mongodb://127.0.0.1:27017/energydb', {
  useNewUrlParser: true,
  useUnifiedTopology: true
}).then(() => {
  console.log('MongoDB connected');
}).catch(err => {
  console.error(' MongoDB connection error:', err);
});

// Mongoose schema
const energySchema = new mongoose.Schema({
  timestamp: { type: Date, default: Date.now },
  deviceId: String,
  label: String,
  wattage: Number,
  totalWattage: Number
});

const EnergyData = mongoose.model('EnergyData', energySchema);

// Route: POST /record-energy
```

```
app.post('/record-energy', async (req, res) => {
  const { deviceId, label, wattage } = req.body;
  try {
    const newEntry = new EnergyData({ deviceId, label, wattage });
    await newEntry.save();
    res.json({ message: 'Energy recorded successfully' });
  } catch (err) {


console.error('Error saving energy data:', err);
    res.status(500).json({ message: 'Failed to record energy' });
  }
});

// Route: GET /monthly-stats
app.get('/monthly-stats', async (req, res) => {
  try {
    const currentMonth = new Date().getMonth();
    const stats = await EnergyData.aggregate([
      {
       $match: {
         timestamp: {
           $gte: new Date(new Date().getFullYear(), currentMonth, 1),
           $lt: new Date(new Date().getFullYear(), currentMonth + 1, 1)
         }
       }
      },
      {
       $group: {
         _id: { day: { $dayOfMonth: "$timestamp" } },
         total: { $sum: "$wattage" }
       }
      },
      { $sort: { "_id.day": 1 } }
    ]);
    res.json(stats);
  } catch (err) {
    console.error('Error getting monthly stats:', err);
    res.status(500).json({ message: 'Failed to load monthly stats' });
  }
});
// Route: GET /device-usage
app.get('/device-usage', async (req, res) => {
  try {
    const usage = await EnergyData.aggregate([
      {
       $group: {
         _id: "$label",
         total: { $sum: "$wattage" }
       }
      },
      { $sort: { total: -1 } }
```

```
    ]);
    res.json(usage);
  } catch (err) {
    console.error('Error getting device usage:', err);
    res.status(500).json({ message: 'Failed to load device usage' });
  }
});
// Route: POST /predict (fixed)
const { spawn } = require('child_process');

app.post('/predict', (req, res) => {
  const py = spawn('python', ['ml/predict.py']);
  let result = '';
  let error = '';
  py.stdout.on('data', (data) => {
    result += data.toString();
  });
  py.stderr.on('data', (data) => {
    error += data.toString();
    console.error(' Python error:', data.toString());
  });
  py.on('close', (code) => {
    if (error) {
      return res.status(500).json({ message: 'Python script error', error });
    }
    const predicted = parseFloat(result.trim());
    if (!isNaN(predicted)) {
      res.json({ predicted });
    } else {
      console.error(' Invalid prediction result:', result);
      res.status(500).json({ message: 'Prediction failed', result });
    }
  });
});


// Start server
const PORT = 5000;
app.listen(PORT, () => {
  console.log(` Server running at http://localhost:${PORT}`);
});
```

7.TESTING

Testing is a crucial stage in the software development life cycle (SDLC) aimed at ensuring the software product is reliable, functional, efficient, and free of bugs. The primary goal is to validate whether the developed system meets the predefined requirements and performs the intended tasks correctly and consistently.

In this project, multiple testing methodologies were adopted at different phases to validate individual components and the integrated system. The Smart Home Energy Management System was tested through unit testing, integration testing, system testing, functional testing, and acceptance testing.

Testing Methodologies

1. Unit Testing

Unit testing is performed on individual modules or functions to ensure each part performs as expected in isolation.

Application in Project:

- Each API endpoint in server.js was tested using tools like Postman.
- The predict.py script was run independently to verify ARIMA predictions.
- Functions such as energy recording and device status updates were tested separately.

2. Integration Testing

Integration testing checks the interaction between modules to ensure they work together as expected.

Application in Project:

- Verified communication between frontend and backend (via fetch() API).
- Checked data flow between Node.js backend and MongoDB.
- Ensured correct input/output when Node.js interacts with the ARIMA model (via child_process.spawn()).

3. System Testing

System testing validates the complete and integrated software system to ensure compliance with specified requirements.

Application in Project:

- End-to-end flow was tested, including:
    - Logging in
    - Inputting device data
    - Viewing visual energy usage charts
    - Sending data for prediction and receiving results
- Confirmed that each feature works cohesively in a real-time scenario.

4. Functional Testing

Functional testing ensures each function of the software application operates according to the requirements.

Application in Project:

- Validated buttons like "Send Test Prediction", "Add Device", and "Log Energy".
- Confirmed accurate energy usage visualization using Chart.js.
- Tested the logic used to calculate monthly stats and device-wise usage.

5. Acceptance Testing (UAT)

User Acceptance Testing is performed by the end-user or client to verify whether the system is ready for production.

Application in Project:

- The system was demonstrated to stakeholders (faculty/project evaluators).
- Dummy energy data was entered and prediction results were checked.
- All charts, features, and device management interfaces were reviewed for usability.

Advanced Testing Approaches

1. Black Box Testing

Tests the functionality of the application without knowledge of internal code logic.

Application:

- Input various energy usage values and checked expected prediction alerts.
- Submitted device data from UI and checked MongoDB via Compass for correct insertions.

2. White Box Testing

Tests internal logic and structure of the code with full knowledge of source code.

Application:

- Inspected JavaScript and Node.js logic for route handling and error checks.
- Validated conditional blocks, loops, and exception handling in server.js andpredict.py.

3. Gray Box Testing

Tests with partial knowledge of internal logic. Combines black and white box techniques.

Application:

- Tested ARIMA integration by modifying prediction values in Python and observing frontend.
- Sent manual curl and Postman requests to analyze backend behavior and logs.

7.2 Real Errors & Solutions

Issue	Cause	Solution
Failed to send prediction	Python path error or missing data	Fixed predict.py,ensured correct spawn() usage
Monthly stats not loading	Missing data or wrong route	Added dummy energy data and corrected API route
Device-wise usage not showing	No devices in DB	Pre-seeded sample device data
MongoDB not connecting (render)	Misconfigured URI	Corrected connection string and used environment variable

7.3 Conclusion of Testing

After extensive testing through multiple strategies, the Smart Home Energy Management System was verified for:

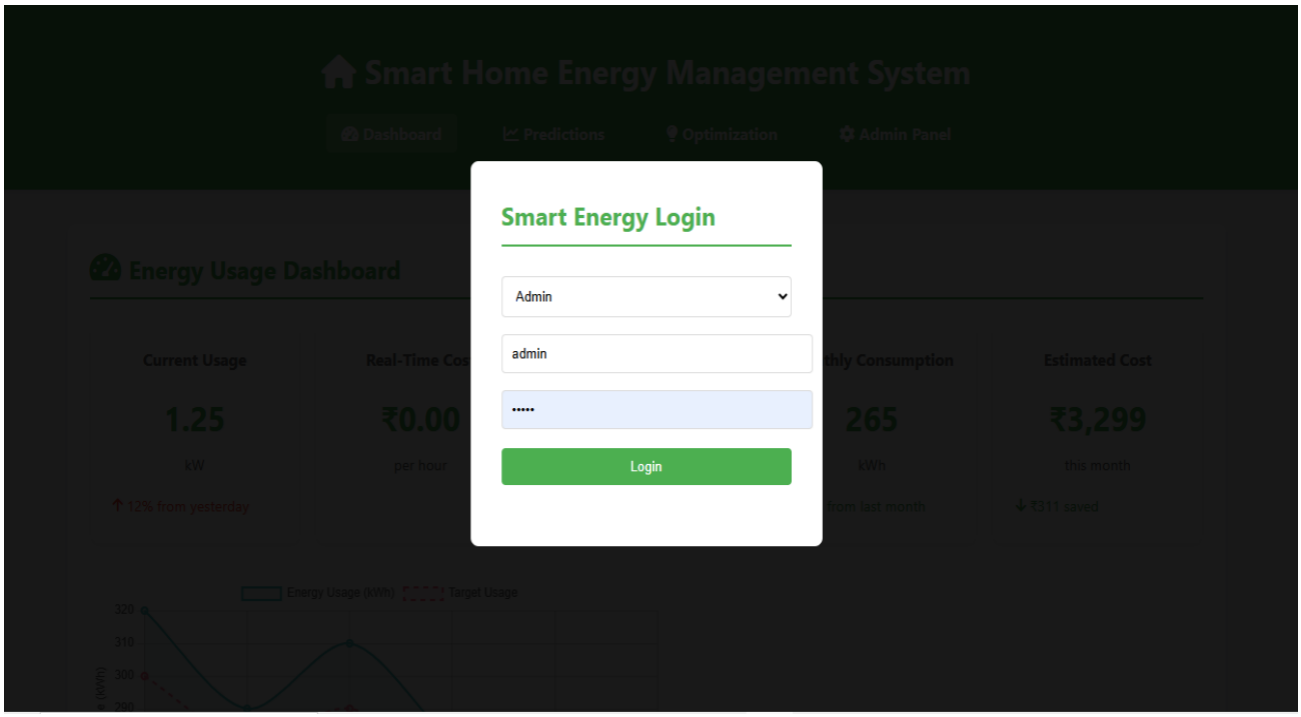
- Accuracy of energy data logging and chart visualization
- Successful connection between frontend, backend, and MongoDB
- Smooth integration with Python ARIMA model
- Proper error handling and fallback support

The system passed all critical test scenarios and is ready for production deployment.

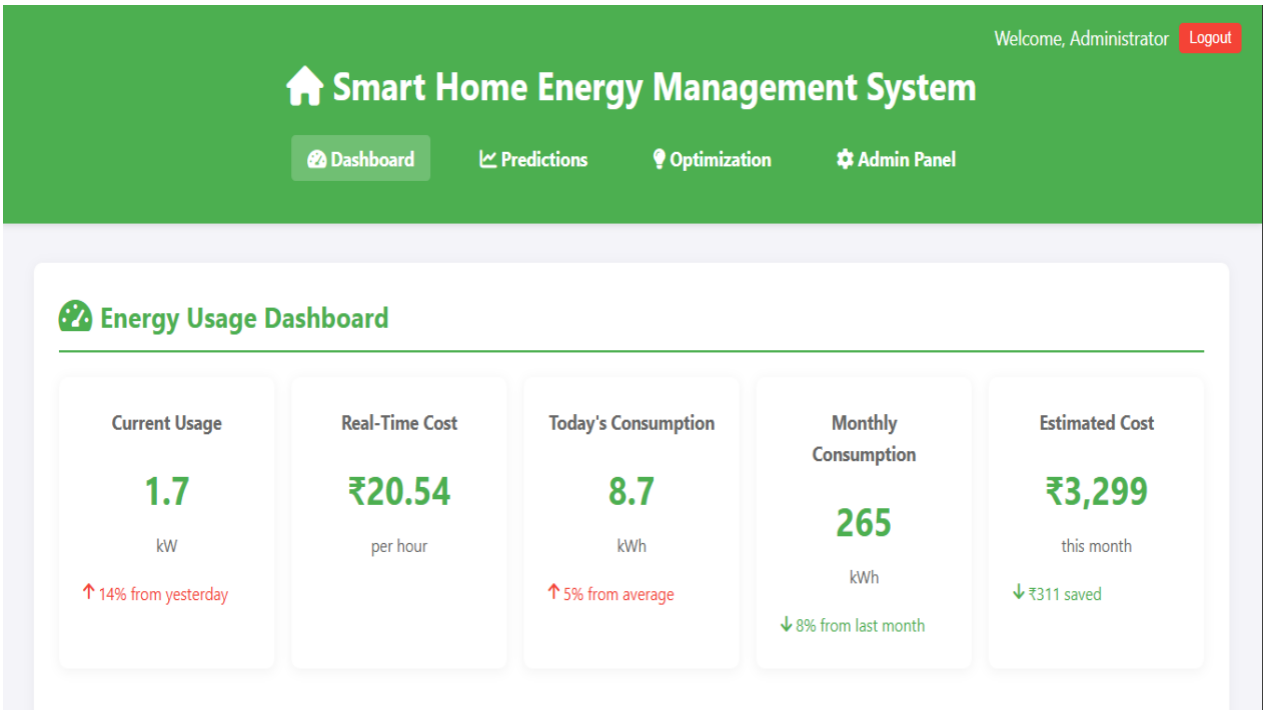
8.OUTPUT SCREENS

Admin login:

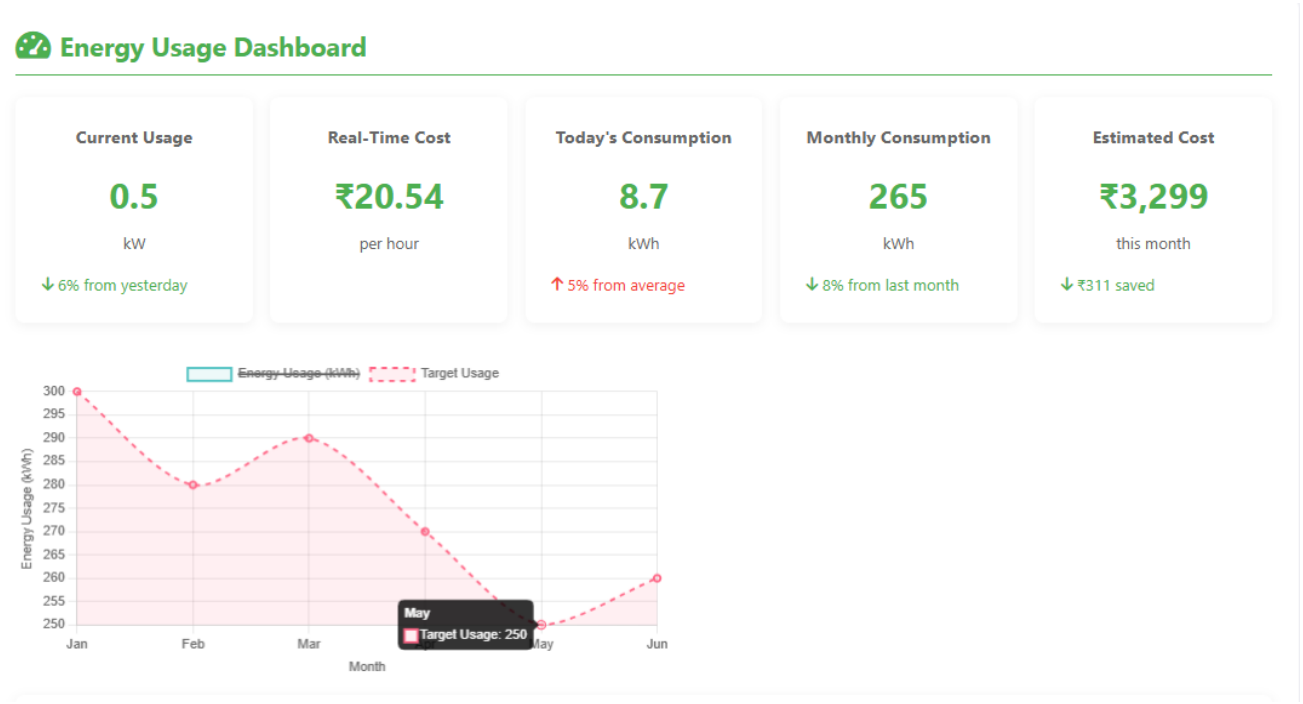
Password: admin



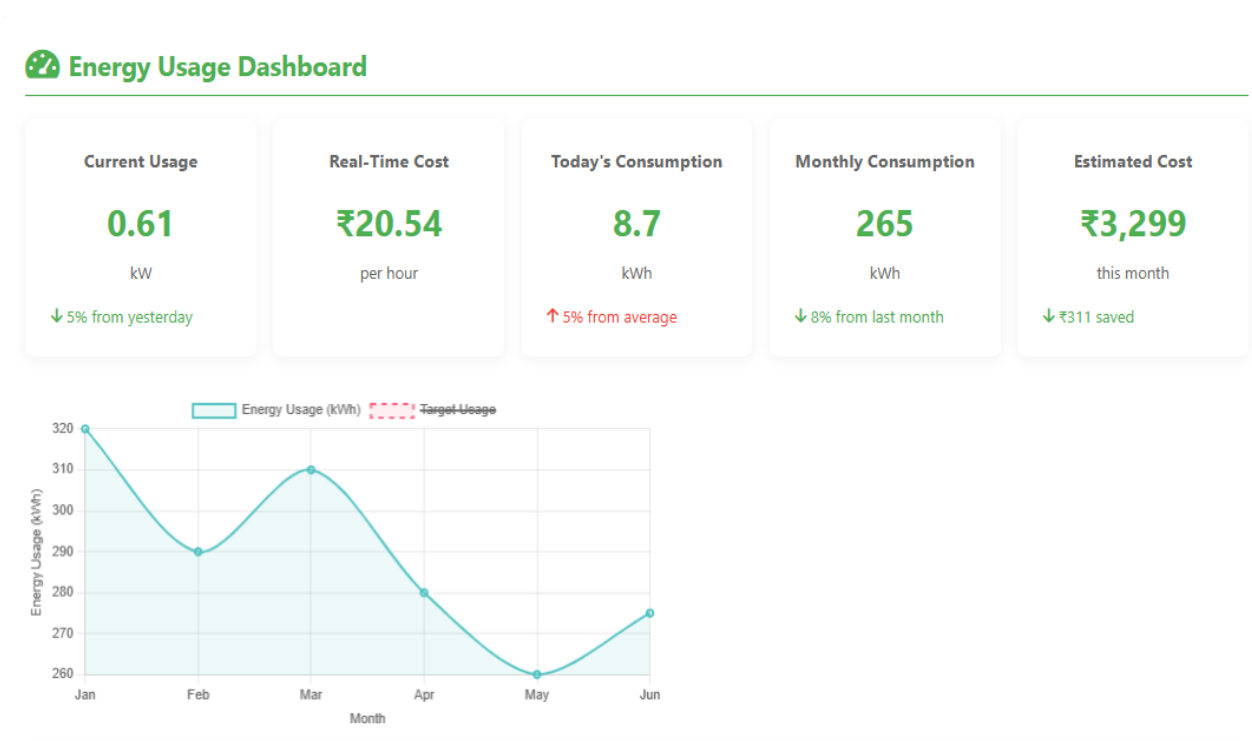
Energy Usage Dashboard:



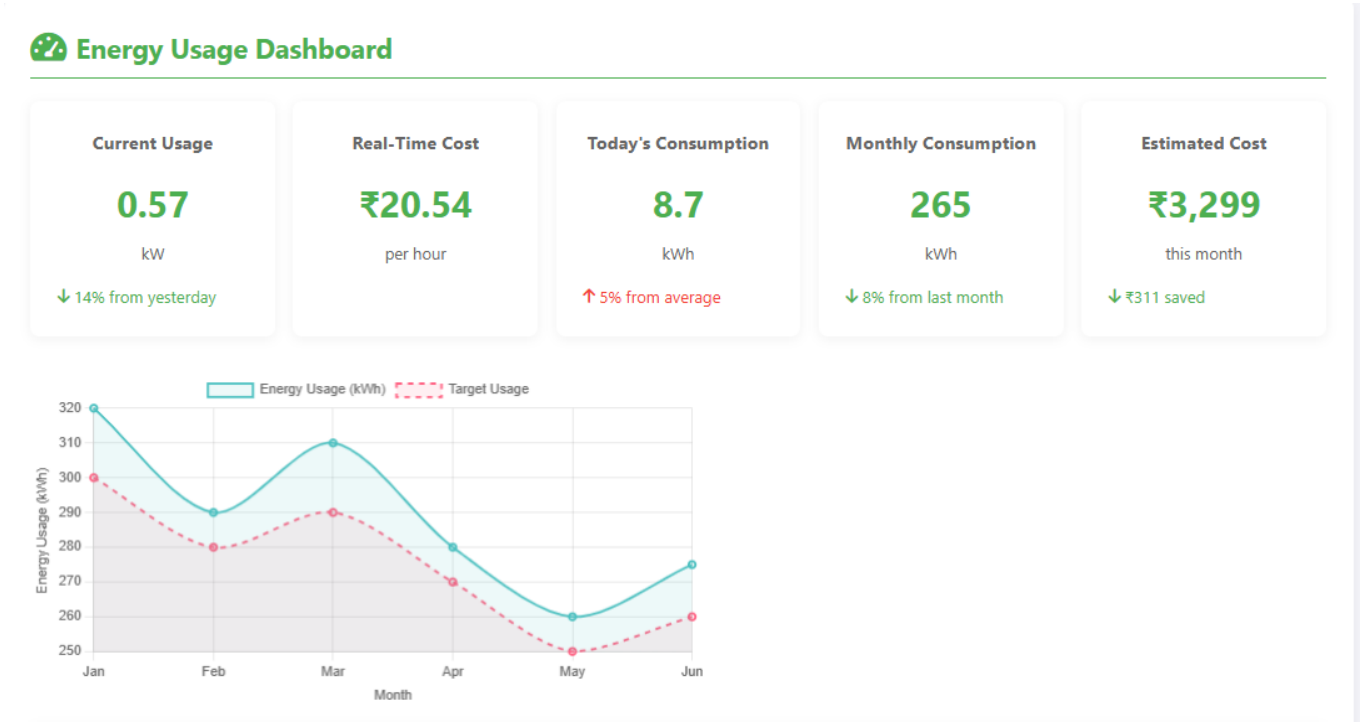
Target usage:



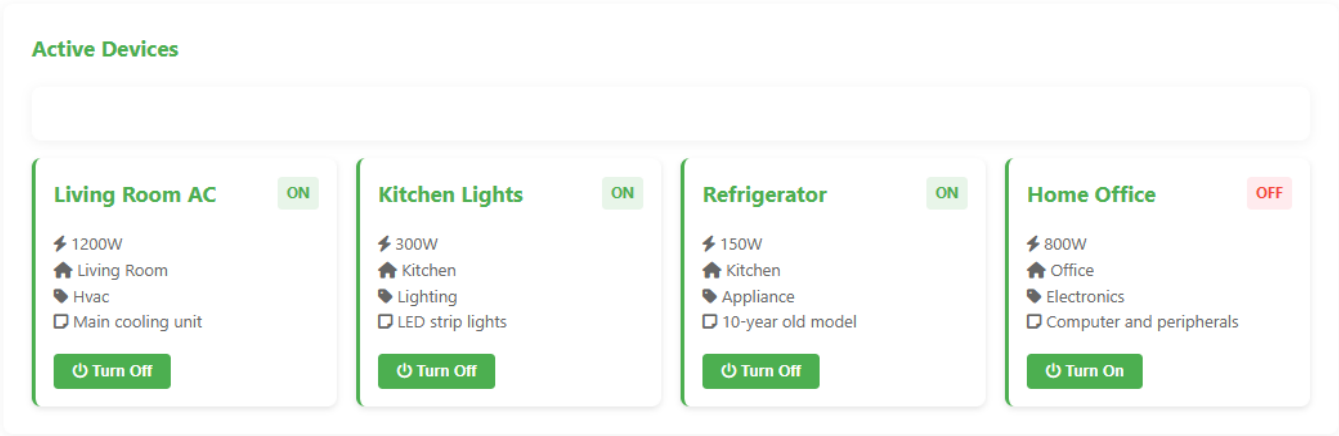
Energy Usage(Kwh):



Energy usage graph

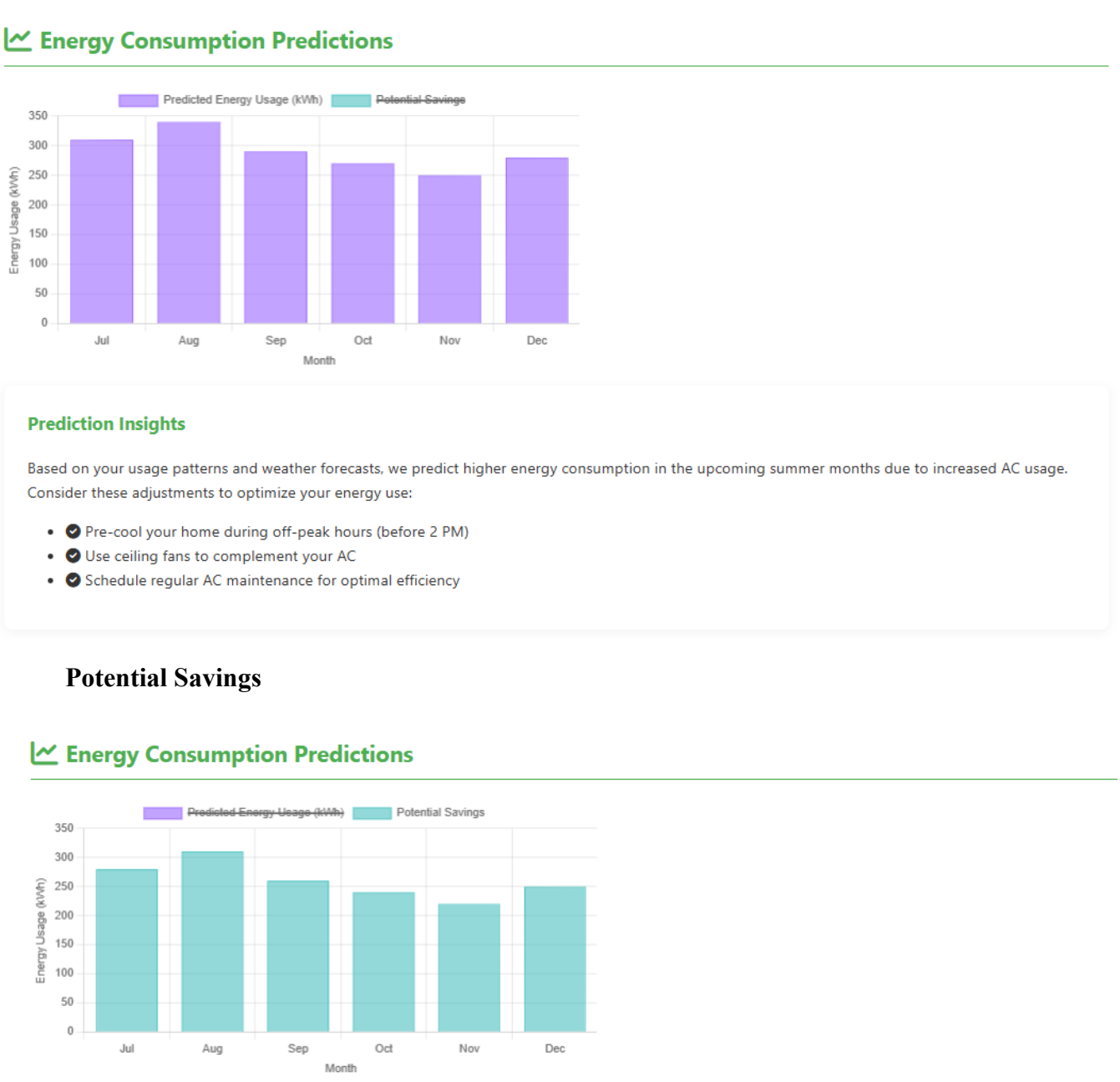


Active Device



Energy Consumption Predictions:

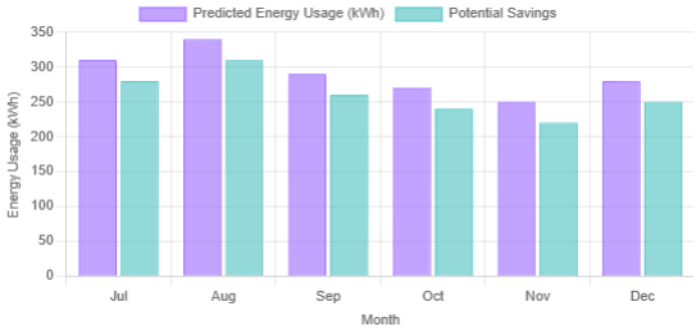
Predicted energy Usage(Kwh):



Potential Savings

Energy Consumption Predictions Bar chart:

Energy Consumption Predictions



Prediction Insights

Based on your usage patterns and weather forecasts, we predict higher energy consumption in the upcoming summer months due to increased AC usage. Consider these adjustments to optimize your energy use:

- Pre-cool your home during off-peak hours (before 2 PM)
- Use ceiling fans to complement your AC
- Schedule regular AC maintenance for optimal efficiency

Energy Optimization Suggestions

Energy Optimization Suggestions

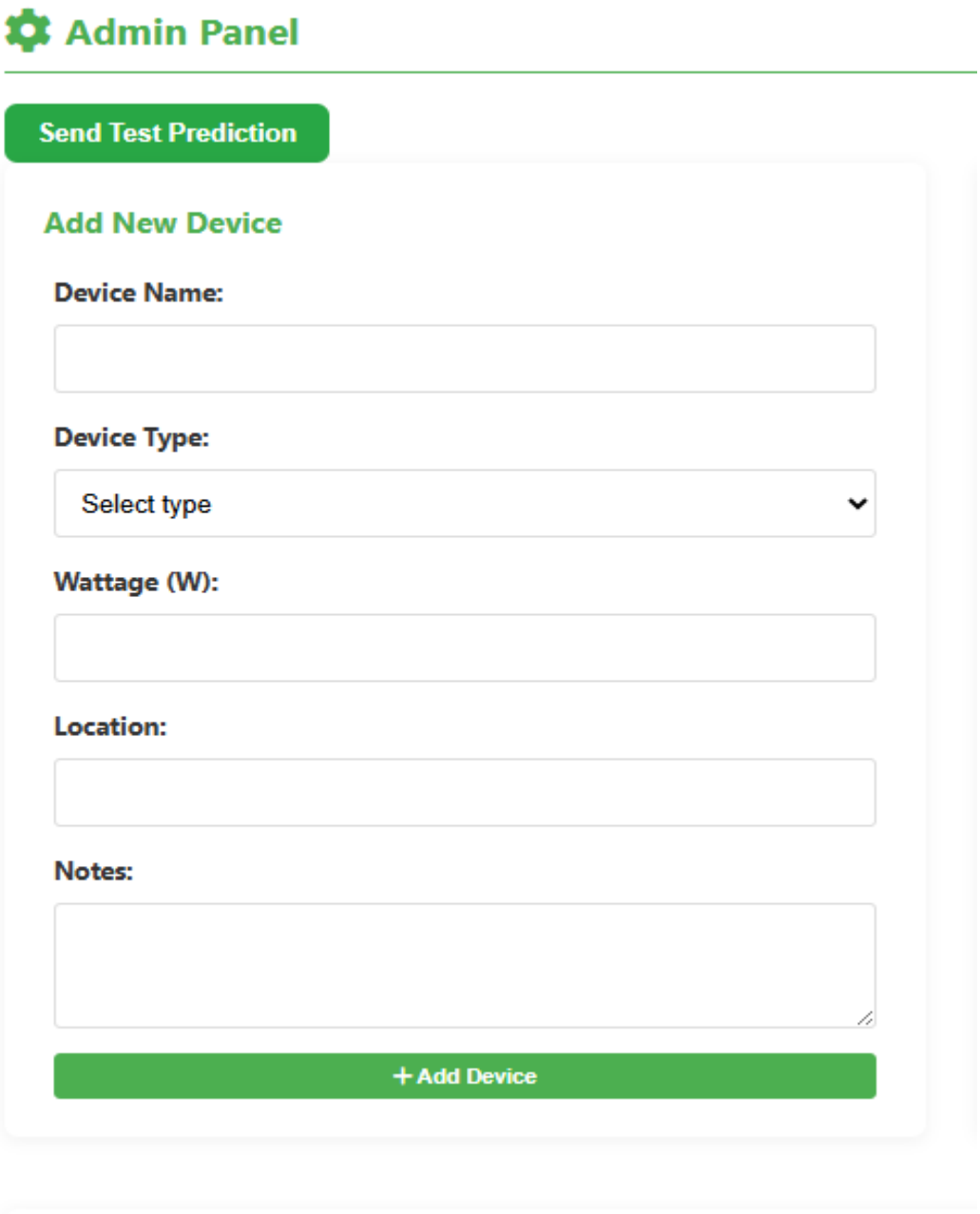
- Living Room AC has been running continuously for 4 hours. Consider turning it off or increasing the temperature by 2°F.
- Kitchen lights are on during daylight hours. Switch to natural light or install motion sensors.
- Your refrigerator is 8 years old. Replacing it with an ENERGY STAR model could save you ₹5,810/year.
- 15% of your energy is used during peak rate hours (4-9 PM). Shift non-essential usage to other times.
- Consider installing solar panels. Based on your roof size, you could offset 65% of your energy needs.
- 3 devices are in standby mode, consuming 45W continuously. Use smart plugs to completely power them off.

Potential Savings

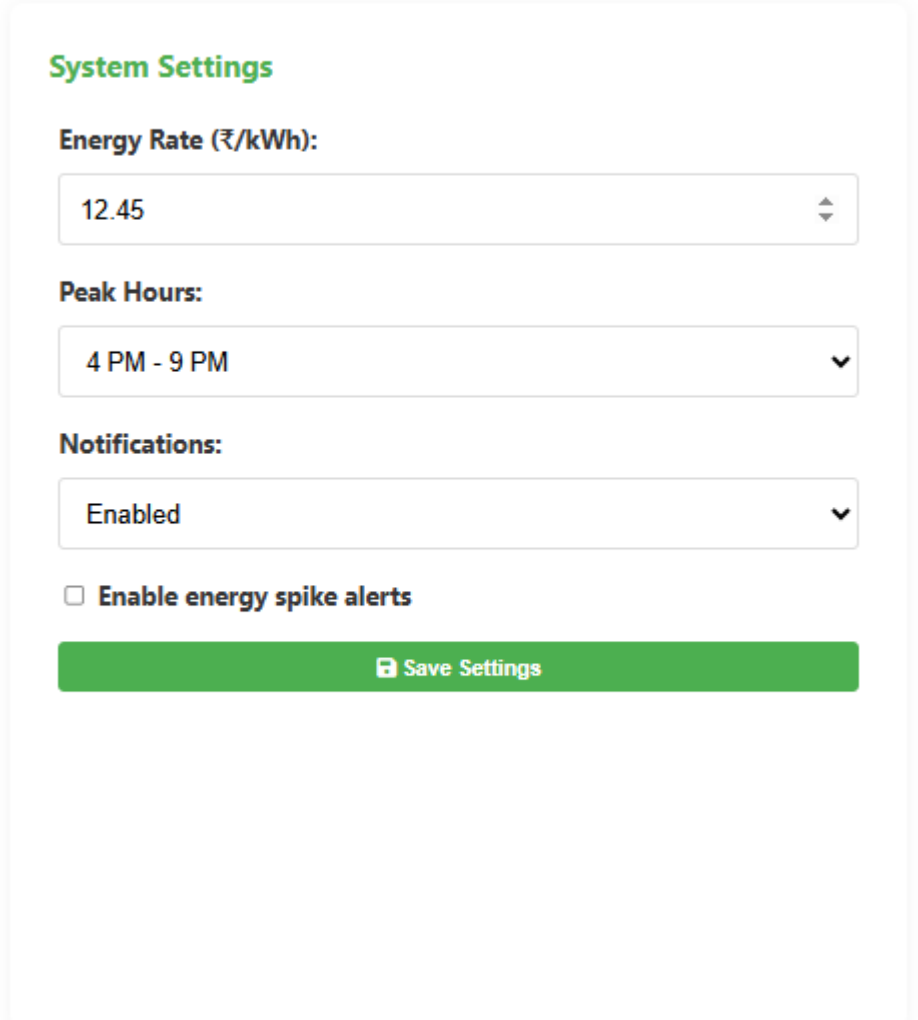
Implementing these suggestions could save you approximately **₹1,826/month** or **₹21,912/year** on your energy bills.

Schedule Optimization Plan

Admin Panel


Admin Panel
Send Test Prediction
Add New Device
Device Name:
Device Type:
Select type
Wattage (W):
Location:
Notes:
+ Add Device

System Settings



9.CONCLUSION

The Smart Home Energy Management System successfully addresses the growing need for intelligent energy monitoring in residential spaces. By leveraging Node.js for backend logic, MongoDB for efficient data storage, and ARIMA for accurate time-series forecasting, the system offers a robust framework for analyzing and predicting household energy consumption. The platform allows users to input, visualize, and understand their energy usage patterns through interactive dashboards powered by Chart.js, helping them make informed decisions to optimize consumption and reduce costs. The admin panel enables easy management of connected devices, adding scalability and control to the system.

Comprehensive testing—including unit, integration, and black-box testing—has ensured the system's reliability, stability, and performance. The successful deployment using Render and MongoDB Atlas demonstrates that the project is cloud-ready and globally accessible.

Overall, the project exemplifies how modern web technologies and predictive analytics can contribute to building smarter, more sustainable homes. It lays a strong foundation for future extensions such as IoT integration, mobile app support, and more advanced machine learning models.

10.Bibliography

For Node.js

• https://nodejs.org/en/docs/

• https://www.w3schools.com/nodejs/

• https://www.geeksforgeeks.org/node-js/

For MongoDB

• https://www.mongodb.com/docs/

• https://www.geeksforgeeks.org/introduction-to-mongodb/

• https://www.mongodb.com/cloud/atlas

For Express.js

• https://expressjs.com/

• https://www.geeksforgeeks.org/express-js/

For Chart.js

• https://www.chartjs.org/docs/latest/

• https://www.tutorialspoint.com/chartjs/

For ARIMA (Python ML Model)

• https://www.statsmodels.org/stable/generated/statsmodels.tsa.arima.model.ARIMA.html

• https://www.geeksforgeeks.org/arima-model-in-python/

For Python

• https://www.python.org/doc/

• https://www.w3schools.com/python/

Project URL

Smart Home Energy Management System