EAST WEST UNIVERSITY
Department of Computer Science and Engineering
B.Sc. in Computer Science and Engineering Program
Midterm Assessment Project, Summer 2025 Semester

Course:	**CSE407 Green Computing, Section: 02**
Student ID:	2022-1-60-267
Student Name:	Abdul Mumeet Pathan

Project Title:
IoT based real-time energy monitoring and dashboard development.

Checklist:

I used IoT technology to monitor the energy consumption of the appliance for a week.
I have prepared an energy monitoring dashboard/webpage with detailed and summarized information.
I have considered the financial and business aspects.
I did PESTLE analysis on the project.

Declaration

I declare that the project was done by myself, and the submitted information is free from any unfair means. I complied with all the relevant legal and organizational requirements.

Abdul Mumeet Pathan
Signature

Name: Abdul Mumeet

Date: 03-08-2025

1. Executive Summary

This project presents the design and development of a real-time smart energy monitoring system using a Tuya-compatible smart plug and a custom-built Python Flask dashboard. The primary goal was to track the power consumption of a household refrigerator, log its energy usage continuously, and present meaningful visual and financial insights through a browser-based interface.

The project began with a detailed planning and research phase, where different devices were considered for monitoring. A refrigerator was selected due to its 24/7 operation, consistent load behavior, and practical relevance. A Tuya smart plug was chosen for its affordability, safety, and compatibility with Python-based local control using the tinytuya library.

The dashboard was developed to offer real-time device monitoring, manual ON/OFF control, and a smart timer feature. Additional functionalities include a bill calculator using local DESCO billing rates, a history viewer for daily energy trends, and a bilingual user manual for accessibility. Energy data was collected every few seconds and stored in structured JSON format, then preprocessed and visualized using Chart.js.

Several technical challenges were encountered during development. These included difficulties in extracting the Local Key, unstable hosting with LocalTunnel and ngrok, and stale data returned from the plug when the Tuya app was inactive. These were resolved by assigning a static IP, switching to Cloudflare Tunnel hosting, and finally implementing a Node.js socket-based polling mechanism to ensure fresh data retrieval.

The system was tested with a 24-hour data collection on August 1, 2025, and successfully identified power cycles and idle periods. The stored data helped estimate daily consumption and projected monthly electricity bills. A thorough review of safety, data quality, scalability, environmental impact, and business potential was conducted. The solution was found to be cost-effective, reliable, and scalable for broader use in homes and small businesses.

Overall, the project not only demonstrates the technical feasibility of submetering through smart plugs but also highlights how such solutions can contribute to energy awareness, cost savings, and sustainable practices under the scope of Green Computing.

2. Brief description of the work

The project followed a step-by-step approach, starting from planning and research to final data presentation. The goal was to measure real-time energy usage of a refrigerator using a smart plug. The steps are :

1. **Planning & Researching**
   Explored feasible measurement options and selected a refrigerator for its consistent energy use. Choose a Tuya-compatible smart plug based on availability and safe current handling. Verified power compatibility using calculated current ratings.

2. **Purchasing & Setup**
   Bought a Tuya smart plug, installed the Tuya Smart app, and connected the plug to Wi-Fi. Retrieved the device's local key and IP for communication.

3. **Configuration**
   Created a Tuya developer project, extracted credentials, and configured the device for local control using the tinytuya library.

4. **Development**
   Built a Flask-based dashboard that collects and displays power, voltage, and current data. Enabled manual ON/OFF control and timer-based switching.

5. **Testing**
   Verified connectivity, accurate data logging, and smooth interaction with the device. Ensured reliability of real-time updates and local communication.

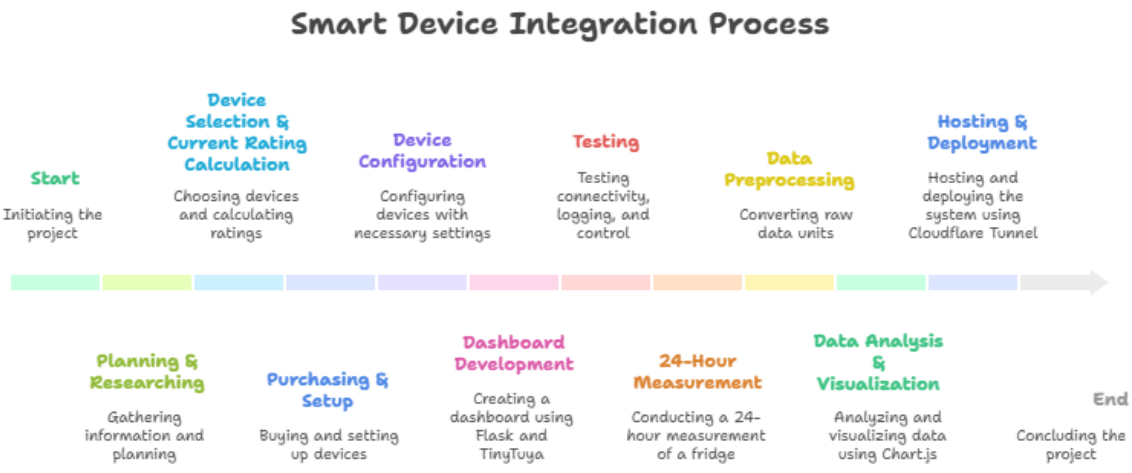
6. **Measurement (August 1, 2025)**
   Monitored the refrigerator for a full 24 hours. Logged electrical parameters continuously to capture complete usage behavior.

7. **Data Preprocessing**
   Converted raw values into standard units (W, V, A) using scaling operations, and stored the cleaned data in JSON format for easy access.

8. **Data Analysis & Visualization**
   Visualized the logged data on the dashboard using Chart.js. Identified patterns like compressor cycles and peak usage hours.

9. **Hosting**
   Hosted the dashboard online using Cloudflare Tunnel and mapped it to a custom domain. Enabled secure, browser-based access.

3. Flowchart of the work



4. Detailed description of each of the steps

0.Planning

The first step of this project involved identifying a suitable electrical device for continuous energy monitoring, as well as exploring viable tools to log its energy consumption effectively. The key planning considerations were centered around device selection, measurement method, and data access strategy.

Device Selection

To measure real-world energy usage, I considered several appliances including:

- **Air Conditioner (AC)** – High power-consuming but only active intermittently.
- **Computer Lab Devices** – Shared infrastructure with inconsistent availability.
- **Refrigerator** – A personal appliance that runs continuously throughout the day.

After thorough evaluation, a refrigerator was selected as the target device for energy monitoring due to its continuous 24/7 operation with periodic cycles, enabling uninterrupted data collection that captures natural fluctuations in power usage, compressor behavior, and temperature regulation. Its practical relevance as a common household appliance with a notable impact on electricity bills makes it an ideal candidate for generating meaningful insights into daily consumption and potential efficiency improvements. Being a personal device, it offered safe and exclusive access for setup without interfering with shared infrastructure. Moreover, its moderate and variable power draw allowed for observing useful consumption trends without the extremes of high-power appliances like air conditioners or the minimal load of devices like LED bulbs. These factors made the refrigerator the most practical and data-rich choice for this study.

Measurement Method: Buy or Build?

For energy measurement, I explored two paths:

- Buy a smart plug with monitoring capability
- Build a custom energy meter

While building offered flexibility, it required calibration, circuit design, and potentially unsafe handling of high-voltage connections. In contrast, commercially available smart plugs offered:

- Plug-and-play usage
- Built-in power, voltage, and current measurement
- Remote control support via Wi-Fi or cloud

Therefore, I chose to **buy a Tuya-compatible smart plug** that supports energy monitoring and integrates easily with Python via the tinytuya library.

Data Access: API vs Local Key

Tuya smart devices provide two major ways of accessing data:

1. **Tuya Cloud API**
    - Requires a registered developer account
    - Limited free tier (often with quotas)
    - Introduces latency due to cloud calls

2. **Local Key Access**
    - Communicates directly with the device on the local network
    - Requires extracting the device ID, IP address, and local key
    - No internet dependency or rate-limits

I opted for **local key access** for the following reasons:

- **Low Latency & Fast Updates**: Direct communication allows quicker status checks and real-time logging.
- **Free & Unlimited Use**: Avoids limitations or fees imposed by Tuya's API.
- **Privacy & Independence**: No data leaves the local network, offering higher control and privacy.
- **Offline Testing Capability**: The system works even without an internet connection, as long as the device and server are on the same Local Area Network .

1. Researching

Before implementation, it was important to confirm the feasibility of the project. The chosen smart plug needed to be readily available, compatible with the target appliance (fridge), and capable of handling its electrical load safely.

Feasibility and Availability

Tuya-compatible smart plugs with energy monitoring were found to be widely available and supported by the tinytuya Python library. I selected a plug that allows local network access using a local key, avoiding API rate limits and enabling faster, real-time communication.

Current Rating Check

To ensure safety, the plug's current rating was verified based on the refrigerator’s power draw. Using:

- **Power** = 250 W
- **Voltage** = 220 V
- **Power Factor** = 0.8
- **Safety Margin** = 20%

Step-by-step calculation:

1. Base current = 250 / 220 = **1.136 A**
2. Adjusted for power factor: 1.136 / 0.8 = **1.42 A**
3. With 20% safety margin: 1.42 × 1.2 = **1.70 A**

Thus, the plug must handle at least **1.7 A**, which is well below 20A rated plug, making it a safe and suitable choice.

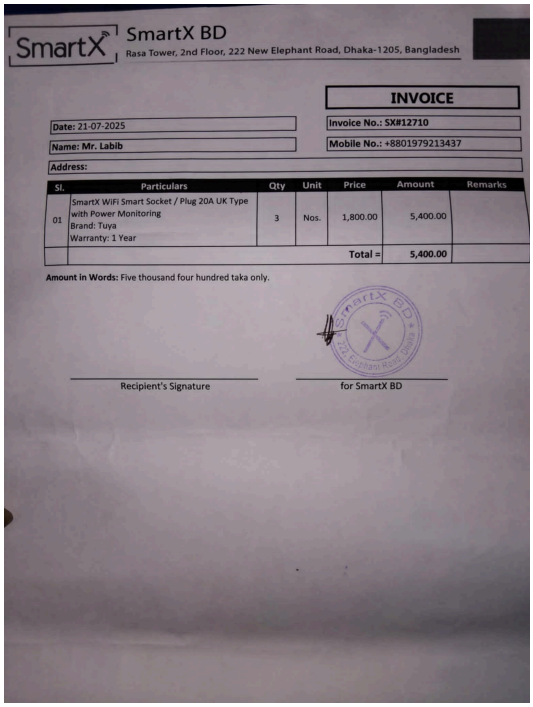
Plan B

If the chosen device had failed or was incompatible, the fallback plan was to use Tuya’s Cloud API or switch to another smart plug .This ensured that the project could continue with minimal disruption.

2.Purchasing

I bought it from New Elephant Road (SmartX BD shop).

SN	Description	Quantity	Unit Cost	Actual Expense Taka	Remarks
1.	Smart Plug	1	1800	1800	Invoice Attached
2.	Transportation	1	30	30	Public Bus
	Total (Taka)			**1830**	



SmartX BD
Rasa Tower, 2nd Floor, 222 New Elephant Road, Dhaka-1205, Bangladesh

INVOICE

Date: 21-07-2025
Invoice No.: SX#12710
Name: Mr. Labib
Mobile No.: +8801979213437
Address:

Sl.	Particulars	Qty	Unit	Price	Amount	Remarks
01	SmartX WiFi Smart Socket / Plug 20A UK Type with Power Monitoring Brand: Tuya Warranty: 1 Year	3	Nos.	1,800.00	5,400.00	
				Total =	5,400.00	

Amount in Words: Five thousand four hundred taka only.

Recipient's Signature

for SmartX BD

3.Configuration

To begin, I installed the Tuya Smart app from the Play Store and registered an account. The smart plug was then added to the app and connected to my local Wi-Fi network, allowing it to be recognized by the Tuya ecosystem.

Next, I visited the TinyTuya website and followed the steps to create a developer project in the Tuya IoT Platform. This provided me with the Access ID, Access Secret, and enabled device linking. Although this is required for initial setup, I ultimately chose to communicate with the device locally.

Local Key Configuration :

Instead of using the cloud API, I opted for local control using the device's Local Key and IP address. This approach provided faster communication, worked offline within the local network, and avoided API limitations.

4. Development

Once the device was configured and recognized on the local network, the next phase involved developing the backend logic and the dashboard interface to handle real-time interaction with the smart plug.

- I used Python + Flask to build a lightweight web application.
- The backend integrated the tinytuya library to fetch device status including power (W), voltage (V), and current (A).
- A logging system was implemented to store timestamped energy data in a JSON file.
- The Flask dashboard provided real-time visual feedback through Chart.js graphs and allowed:
    - **Manual ON/OFF control** of the smart plug
    - **Smart Timer** to auto turn the plug ON/OFF after a specified duration
    - **Energy History** to select and review past daily logs

    - **Bill Calculator** to estimate monthly electricity cost using DESCO's billing model
    - **User Manual** with bilingual accordion-style guidance on dashboard features

5. Testing

Before deploying for full-day data collection, I tested the dashboard and smart plug under controlled conditions:

- Verified device connectivity and response times.
- Confirmed accurate logging of energy data with proper time stamps.
- Simulated ON/OFF switching via both dashboard buttons and timers.
- Ensured local communication was stable and no API rate-limits were encountered.

6. Measurement & Logging (1st August 2025)

On **August 1, 2025**, I connected the smart plug to a refrigerator and ran the system for a full 24 hours.

- The dashboard logged **voltage, power, and current** every two minutes.
- Logging conditions ensured data was saved when values changed significantly or at regular intervals.
- A complete dataset was generated, capturing the fridge's energy profile across day and night.

7. Analysis and Visualization

After logging was complete, I used the dashboard's History section to load and visualize the collected data for August 1.

- Separate line charts were plotted for Power (W), Voltage (V), and Current (A) over time.
- I observed the fridge’s compressor cycling behavior, idle power draw, and voltage fluctuations.
- This helped quantify energy consumption patterns and identify peak usage periods.

8. Data Preprocessing

The raw data from the Tuya device was reported in scaled integers:

- Power and Voltage in tenths
- Current in thousandths

To convert them into standard SI units, I applied:

- Power = raw ÷ 10 (W)
- Voltage = raw ÷ 10 (V)
- Current = raw ÷ 1000 (A)

The converted values were stored in a JSON file and used directly in dashboard visualizations.

9. Hosting the Dashboard

The Flask-based dashboard was hosted online using **Cloudflare Tunnel** and mapped to a custom domain:

https://mumkaftin.dpdns.org/

The backend handled real-time device status, logging, control, and timers via dedicated API routes. The frontend was styled with custom CSS and used **Chart.js** for visualizing power, voltage, and current trends — making the system accessible from any browser securely and reliably.

Challenges and hiccups

Local Key Not Showing

One of the early challenges was extracting the Local Key from the Tuya device. At first, it didn't appear even after linking the device. The problem was resolved by:

- Re-checking that the device was in the correct pairing mode
- Using tinytuya wizard with the correct region, country code, and device type

Device Not Responding via Local IP

At times, the device stopped responding to requests. This was traced to:

- The local IP address changing due to dynamic DHCP allocation
- Solved by reserving a static IP for the device via router settings
- Ensured stable connectivity on the same Wi-Fi network as the Flask server

Hosting Challenges

Hosting the dashboard online was one of the biggest challenges:

- First, I tried li -–port (LocalTunnel), which was unstable, randomly disconnected, and often failed to assign a URL

- Next, I tried ngrok, but it ran into PowerShell execution policy errors and also limited session durations under the free plan

- Finally, I solved the issue by using Cloudflare Tunnel, which was free, reliable, and persistent, and mapped it to a custom domain:
  https://mumkaftin.dpdns.org

Tinytuya Returning Same or Stale Data

A critical issue was that, in Local Key mode, if the Tuya app wasn't open in the background, the smart plug often returned the same stale values.

- Initially, I used an Android emulator (BlueStacks) to keep the Tuya app open 24/7

- Later, I implemented a low-level socket polling system in Node.js, bypassing this limitation and retrieving fresh data without needing the app

Frequent Logging Causing Redundant Entries

At first, the system logged every value change, resulting in excessive data points.

- Optimized by adding conditional logging: only log if power changes by more than 1W or if 2 minutes have passed since the last entry

Timer Cancel Function Not Working

The smart timer could turn the plug on/off after a delay, but canceling it didn't work initially.

- Solved by creating a cancel_timer route and adding logic to terminate active threads in Python

Connectivity Issues (Wi-Fi or Power Drop)

If Wi-Fi disconnected or electricity went out, no data could be collected.

- Although this is a hardware limitation, the system resumes smoothly once the network or power is restored

- Future versions could include buffered logging or local caching to handle temporary disconnections

Finding a Compatible Device

It was important to find a device (like the fridge) that:

- Operates continuously
- Has safe current levels
- Can be left plugged in long-term

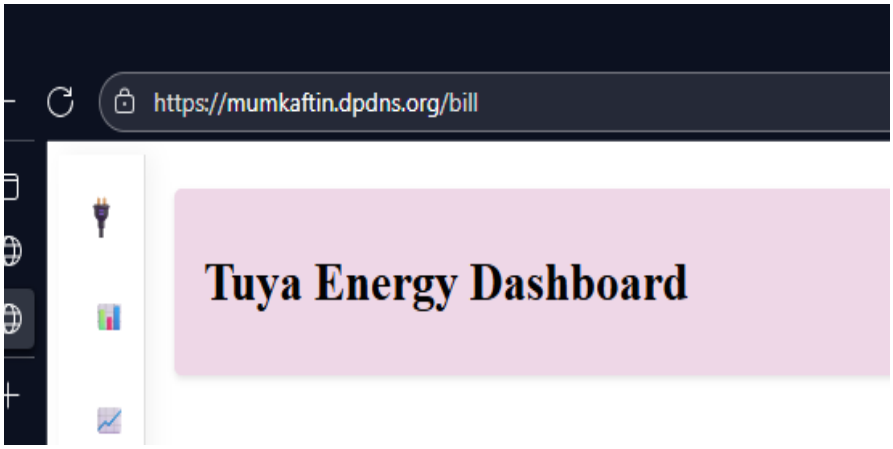
Cloud API Limitations

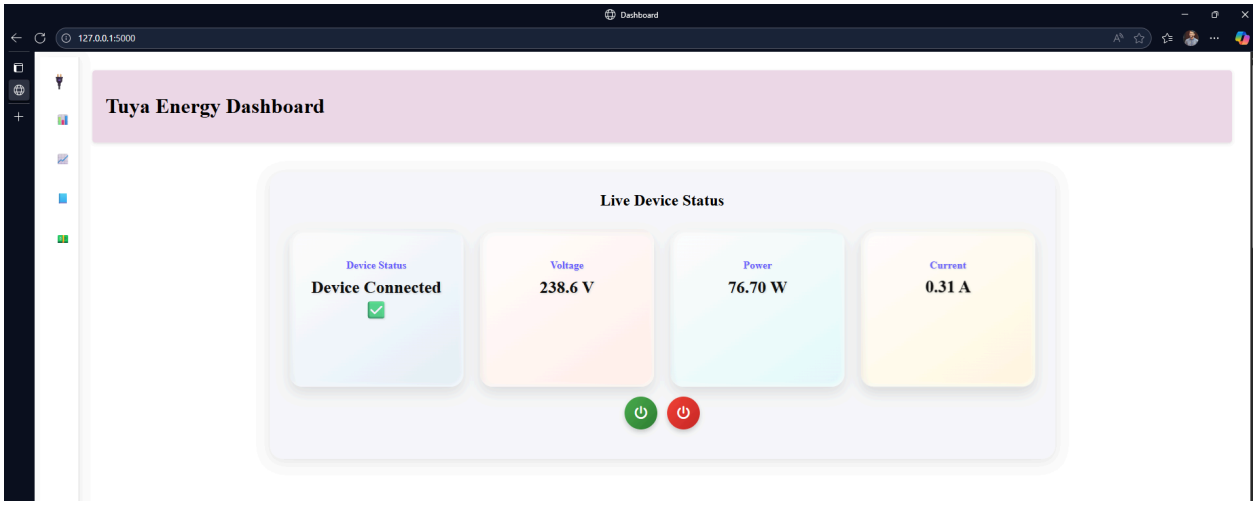
Initially considered using the Tuya Cloud API, but:

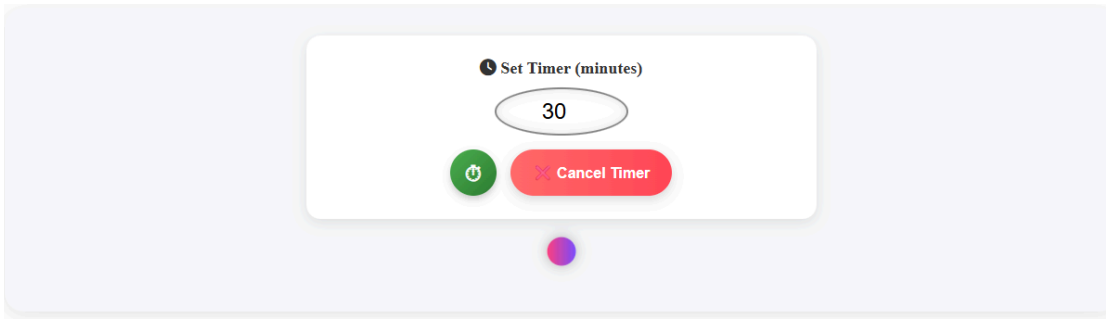
- Required developer approval
- Had token expiration and rate limits
- Instead, I used Local Key mode, which was faster and free, though it introduced some of the above challenges

Demonstration

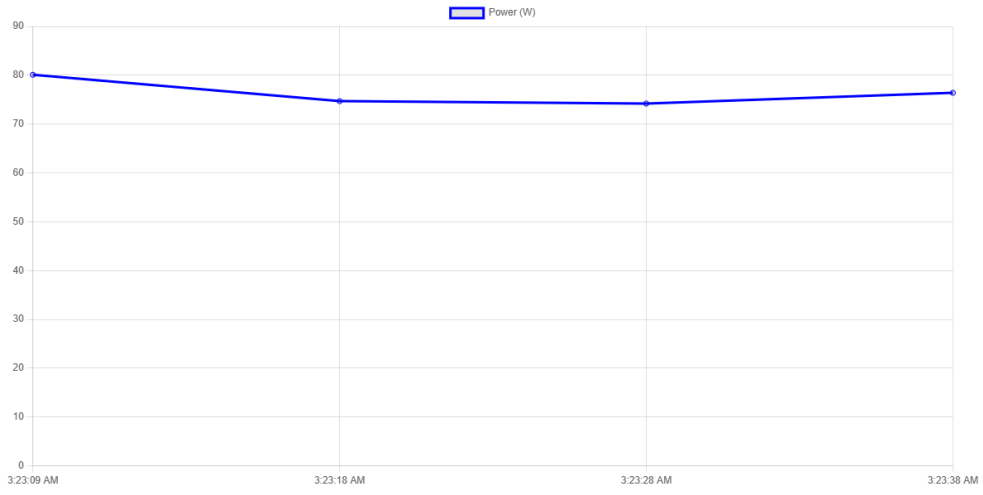
Link to energy dashboard: https://mumkaftin.dpdns.org

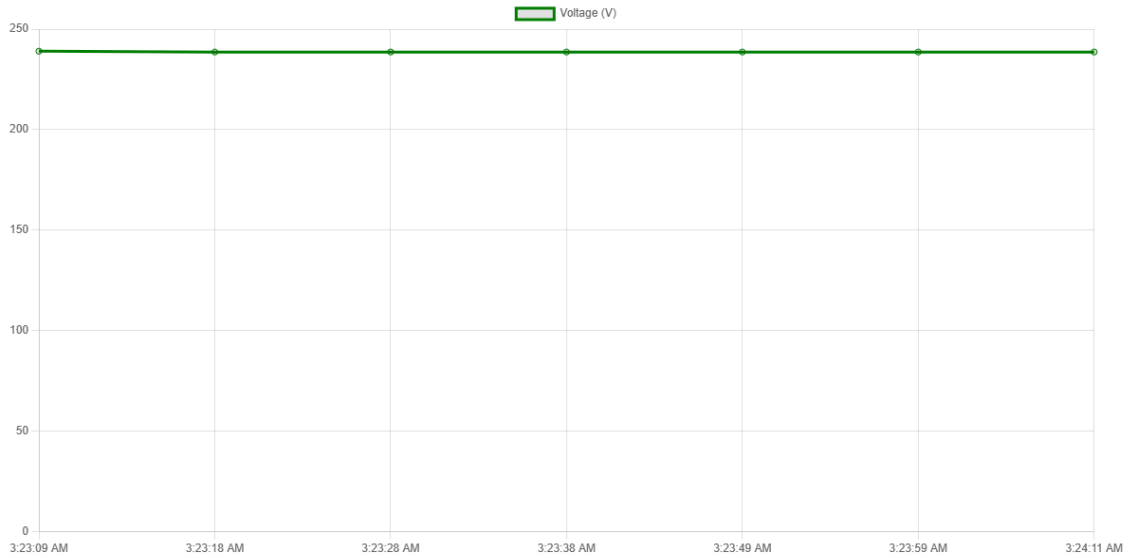


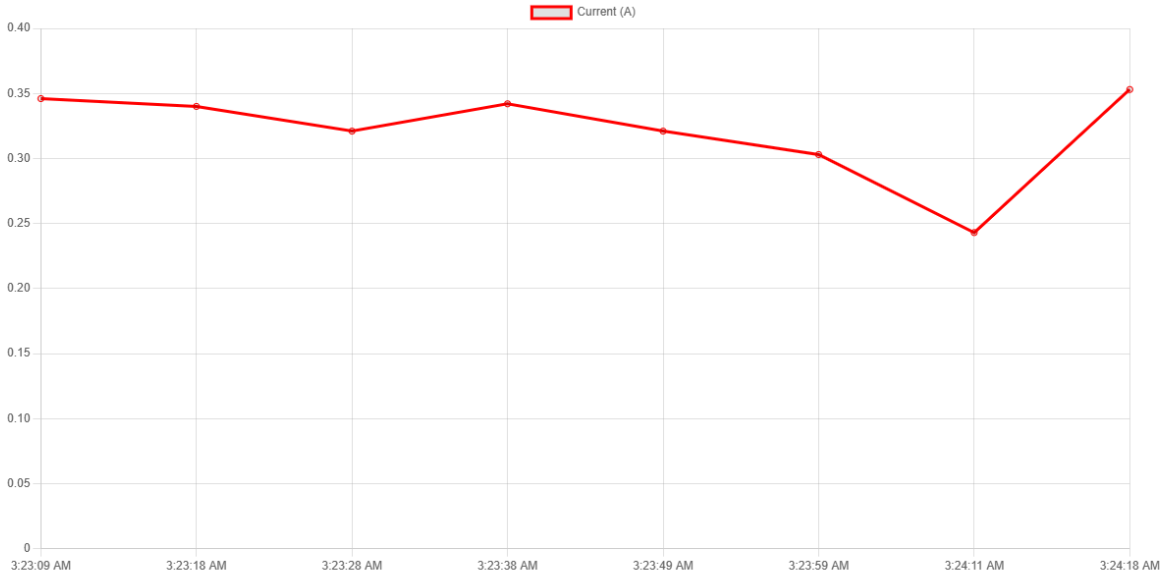




Real-Time Energy Usage

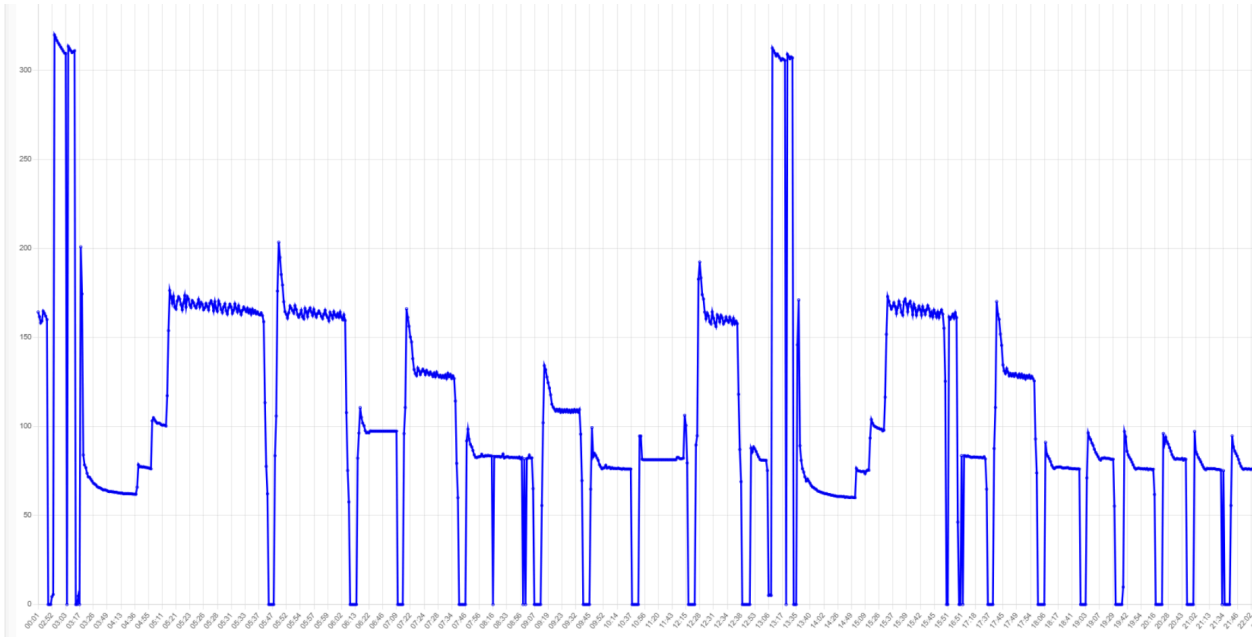



Voltage (V)
250
200
150
100
50
0
3:23:09 AM
3:23:18 AM
3:23:28 AM
3:23:38 AM
3:23:49 AM
3:23:59 AM
3:24:11 AM


Current (A)
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0
3:23:09 AM
3:23:18 AM
3:23:28 AM
3:23:38 AM
3:23:49 AM
3:23:59 AM
3:24:11 AM
3:24:18 AM

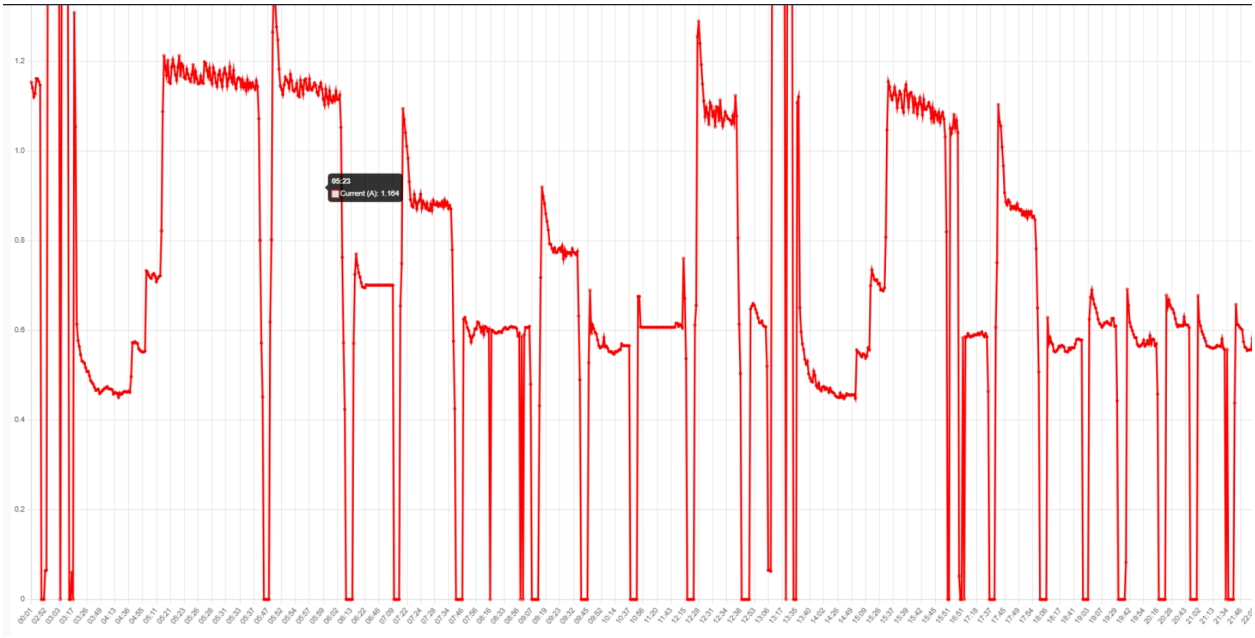
History Page :

Power Chart :



Voltage Chart :

Current :



User manual :

📘 User Manual / ব্যবহার নির্দেশিকা

Print / Download as PDF

Switch to বাংলা

Device Control

Use the ON/OFF buttons to manually control your smart plug. The live status updates show voltage, power, and current.

Timer

Set a timer to auto turn ON/OFF the device after a number of minutes.

Real-Time Monitoring

Power, voltage, and current graphs update every 10 seconds live from the smart device.

Energy History

View logs of previous dates. Select a date to see that day's data.

Bill Calculator

The Bill Calculator lets you:

- View total energy usage (kWh)
- Estimate monthly DESCO bill in Taka
- View per-month usage
- Drill down into per-day usage

Sidebar Navigation

Hover on the sidebar to see menu items like Dashboard, History, Manual, and Bill Calculator.

(In Bangla) :

📘 User Manual / ব্যবহার নির্দেশিকা

🖨 Print / 📄 Download as PDF

🌐 Switch to English

🔌 Device Control

স্মার্ট প্লাগ অন অফ করার জন্য ON ও OFF বাটন ব্যবহার করুন। লাইভ স্ট্যাটাসে ভোল্টেজ, পাওয়ার ও কারেন্ট দেখায়।

⏱ Timer

নির্দিষ্ট মিনিট পর ডিভাইস চালু বা বন্ধ করতে টাইমার ব্যবহার করুন।

📊 Real-Time Monitoring

পাওয়ার, ভোল্টেজ ও কারেন্ট এর চার্ট প্রতি ১০ সেকেন্ডে হালনাগাদ হয়।

📅 Energy History

আগের দিনের তথ্য দেখতে তারিখ সিলেক্ট করুন। চার্ট আকারে দেখাবে।

🧮 Bill Calculator

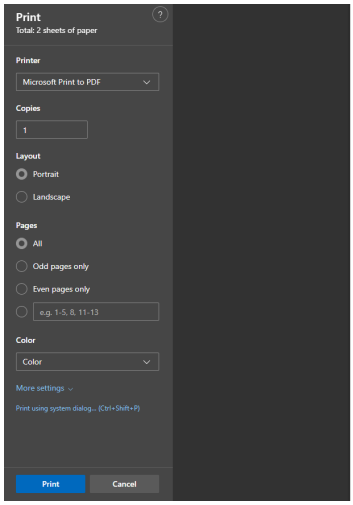
বিল ক্যালকুলেটর দ্বারা আপনি দেখতে পারবেন:

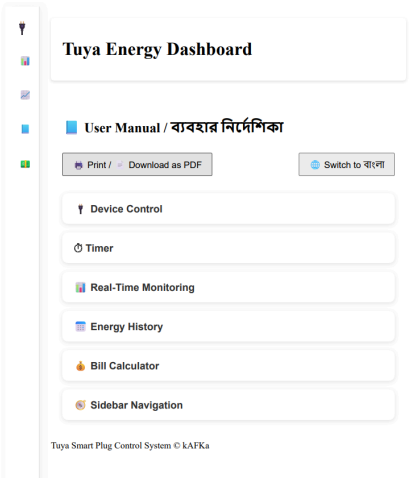
- 📅 মোট বিদ্যুৎ ব্যবহার (kWh)
- 💵 মাসিক DESCO বিলের আনুমানিক পরিমাণ (৳)
- 1️⃣ প্রতি মাসের বিদ্যুৎ ব্যবহার
- 📅 নির্দিষ্ট দিনের হিসাব

🧭 Sidebar Navigation

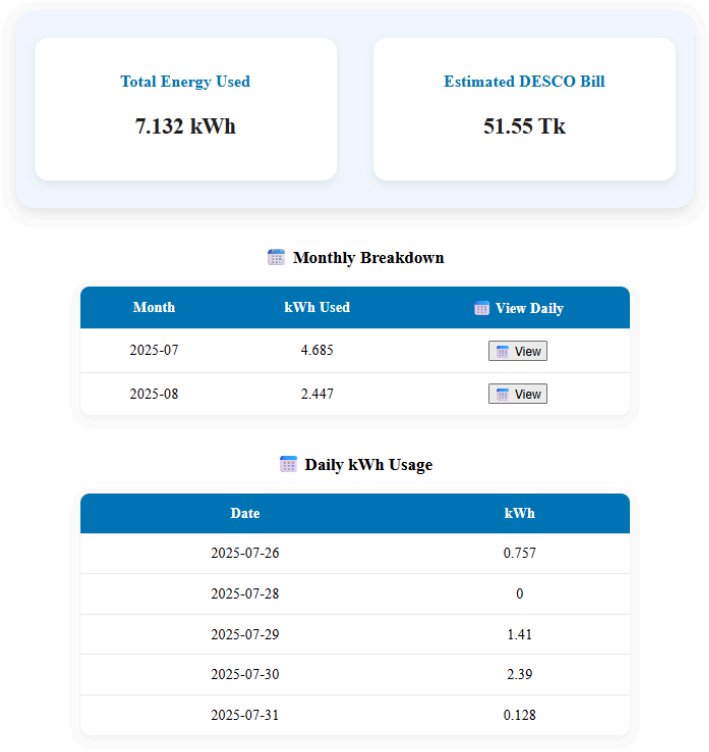
সাইডবারে মাউস রাখলে মেনু আইটেমগুলো দেখা যাবে যেমন: ড্যাশবোর্ড, হিস্টোরি, ম্যানুয়াল, এবং বিল ক্যালকুলেটর।

Print / Download PDF :





Bill and usage Show :



Total Energy Used

7.132 kWh

Estimated DESCO Bill

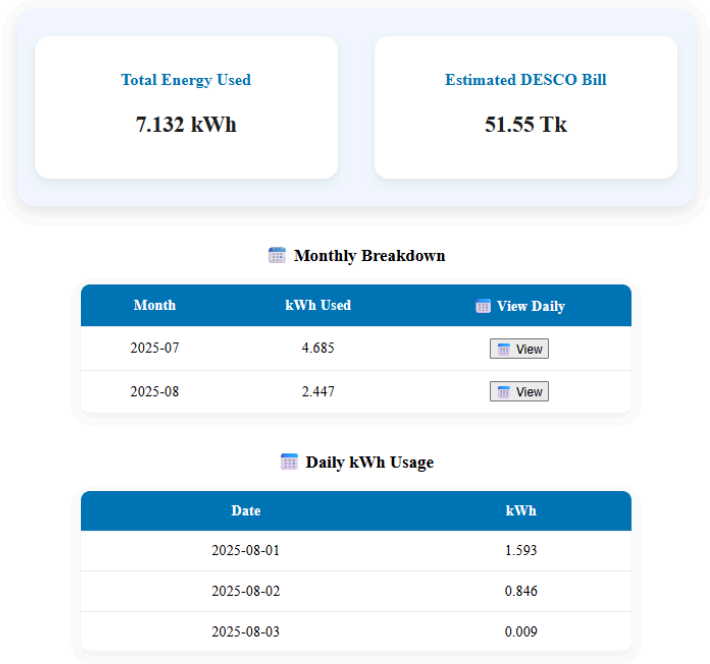
51.55 Tk

Monthly Breakdown

Month	kWh Used	View Daily
2025-07	4.685	View
2025-08	2.447	View

Daily kWh Usage

Date	kWh
2025-07-26	0.757
2025-07-28	0
2025-07-29	1.41
2025-07-30	2.39
2025-07-31	0.128



Total Energy Used

7.132 kWh

Estimated DESCO Bill

51.55 Tk

Monthly Breakdown

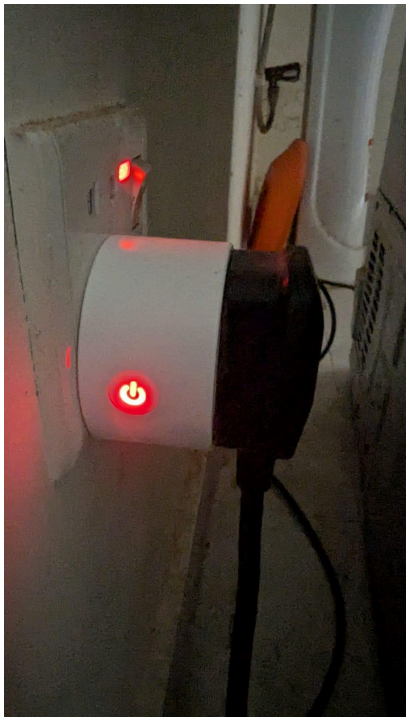
Month	kWh Used	View Daily
2025-07	4.685	View
2025-08	2.447	View

Daily kWh Usage

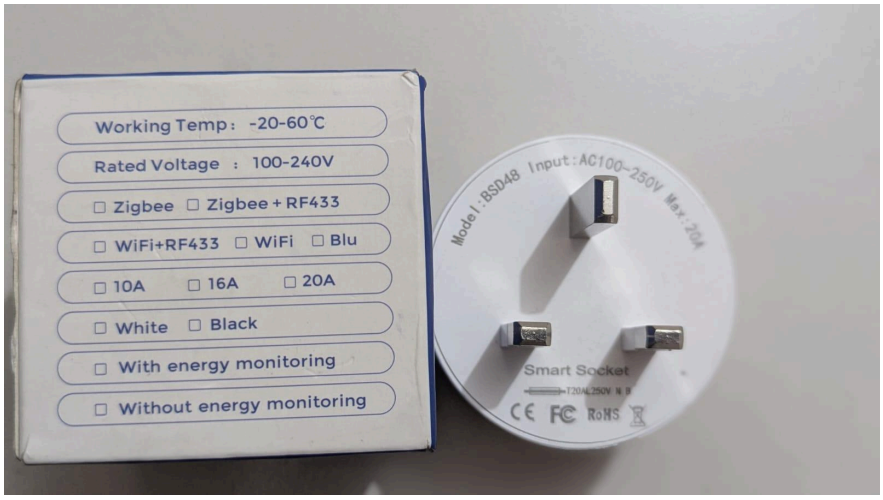
Date	kWh
2025-08-01	1.593
2025-08-02	0.846
2025-08-03	0.009

Measured Device :

Measurement Setup :




Smart socket


Working Temp: -20-60℃
Rated Voltage : 100-240V
☐ Zigbee ☐ Zigbee + RF433
☐ WiFi+RF433 ☐ WiFi ☐ Blu
☐ 10A ☐ 16A ☐ 20A
☐ White ☐ Black
☐ With energy monitoring
☐ Without energy monitoring
Model:BSD48 Input:AC100-250V Max:20A
Smart Socket
T20A 250V
RoHS

Data :


```
  "timestamp": "2025-08-02 16:48:50",
  "power": 60.3,
  "voltage": 231.3,
  "current": 0.199
},
{
  "timestamp": "2025-08-02 16:49:54",
  "power": 39.2,
  "voltage": 231.3,
  "current": 0.19
},
{
  "timestamp": "2025-08-02 16:50:50",
  "power": 36.8,
  "voltage": 231.3,
  "current": 0.192
},
{
  "timestamp": "2025-08-02 16:51:50",
  "power": 51.4,
  "voltage": 231.7,
  "current": 0.235
},
{
  "timestamp": "2025-08-02 16:52:00",
  "power": 43.8,
  "voltage": 230.8,
  "current": 0.223
},
{
  "timestamp": "2025-08-02 16:52:10",
  "power": 39.7,
  "voltage": 231.3,
  "current": 0.201
},
```


Discussion on issues:

Checklist of Issues	**Remarks (if any)**
*Planning and researching?	Completed thoroughly. Device selection, feasibility, and measurement methods were carefully evaluated before purchase.

Data Collected?	Power (W), Voltage (V), and Current (A) data were collected at 2-minute intervals over a 24-hour period.
Data Stored?	Data was stored in a structured JSON file locally, with timestamps and preprocessed units.
Data Displayed in the dashboard?	Real-time and historical data visualized using Chart.js on the dashboard with line charts.
*Realtime or stored data?	Both. Live data was shown in real-time, and logs were stored for history and billing.
*Wattage of the chosen device?	197 W. For more safety consider 250 W.
*Wattage of the measuring equipment?	20 A * 220 V = 4400 W
*AC Power: Why/not apparent power? Why/not Instantaneous Power? AC power vs. DC power?	Instantaneous active power was logged (Watt). Apparent power (VA) wasn't used, as power factor was assumed standard (0.8). AC power chosen since the device runs on alternating current; DC power irrelevant here.
*Documentation	A complete bilingual user manual was provided with instructions for control, timers, and billing.
*Safety: Electrical Insulation and Isolation?	Commercial smart plug used, rated for insulation, surge protection, and proper isolation—avoided DIY circuits for safety.
*Caution with overclocking/flushing	Not applicable
*API Issues: Did you get it? How? If not, how did you solve this problem?	Tuya Cloud API was avoided. Used Local Key for device communication. Solved access via tinytuya local key and network configuration.
*UI/UX issues? Standard components of an energy dashboard?	Custom UI built with responsive design, hover sidebar, and icon-based buttons. Live readings, control buttons, timer, historical trends, billing estimator, and a user manual were all included.

*User Manual?	Yes. Provided in both English and Bangla, accessible from the sidebar too.
*Future Extensions and Limitations?	Could integrate multiple devices, cloud sync, and mobile apps and use API and buy a domain. Currently limited to one device and local control.
*Installation, Operation and Maintenance?	Simple local setup using Flask, no complicated hardware maintenance. Needs stable power and network.
*Recurring costs	None. One-time plug purchase and free Cloudflare Tunnel setup.
*Cost Accounting?	BDT 1800
*Business Aspects? Cost savings and ROI? Value of this product/service? Justification?	High value for households and small businesses. Helps reduce energy wastage and monitor costs. Potential for commercial dashboard offering.
*Reliability? Never failed? Any fail-safe mechanisms?	The system was tested for continuous 24-hour operation without failure. Conditional logging, stable local communication, and the use of reliable commercial smart plugs ensured consistent performance and uninterrupted data collection.
*Accuracy? Calibration?	No external calibration used. Relied on built-in calibration of Tuya plug. Sufficient for non-industrial use.
*Data quality? Sampling rate? Crosstalk and interference? Accuracy and calibration?	Sampled every 120 seconds. No interference or signal loss.
*Scalability?	Scalable by adding more smart plugs and routes. Backend modular for multi-device extension.
*Interoperability?	Works with any Tuya-based plug supporting local key access.

*Data Security? Important or not in this case?	Medium importance here. Local-only access with no cloud dependency ensures privacy and LAN-only exposure
*Compliant with regulations?	For personal use, no regulatory compliance is needed. Uses CE-certified smart plug.
*Environmental Impacts? PESTLE analysis?	Encourages energy conservation and reduces unnecessary power consumption. Positive sustainability use-case.

Issues :

Issue: Planning and researching

Why did I choose this problem?

The core idea behind this project was to explore energy monitoring in a way that is both practical and scalable. Refrigerators are common household appliances that run continuously and consume significant electricity. Monitoring its energy usage in real time allowed me to:

- Understand baseline energy consumption
- Visualize how usage fluctuates throughout the day
- Explore potential areas for energy savings

It also aligns with Green Computing principles by promoting awareness about energy efficiency in everyday electronics.

Why not other devices like PCs, printers, or routers?

I initially considered monitoring devices like:

- Desktops or Laptops – Usage varies widely depending on user behavior and power settings.
- Printers or Routers – These typically have low or idle energy usage most of the time.
- Computer Lab Devices – Difficult to access consistently and ethically since shared by multiple users.

In contrast, a refrigerator:

- Runs 24/7, ensuring a continuous data stream
- Shows real-world periodic cycling due to compressor behavior

- Offers safe, personal access without disturbing shared resources

Tuya-compatible smart plugs with energy monitoring were found to be widely available and supported by the tinytuya Python library. I selected a plug that allows local network access using a local key, avoiding API rate limits and enabling faster, real-time communication.

Issue: Wattage for the chosen device

Calculate the AC current rating that the equipment can handle. Use Power Factor 0.8 and 20% safety factor.

- **Power** = 250 W
- **Voltage** = 220 V
- **Power Factor** = 0.8
- **Safety Margin** = 20%

Step-by-step calculation:

4. Base current = 250 / 220 = **1.136 A**
5. Adjusted for power factor: 1.136 / 0.8 = **1.42 A**
6. With 20% safety margin: 1.42 × 1.2 = **1.70 A**

Thus, the plug must handle at least **1.7 A**, which is well below 20A rated plug, making it a safe and suitable choice

Issue: AC Power Considerations

Understand why apparent power (kVA) is used instead of instantaneous power (Watt). Also,consider whether you're dealing with AC or DC power.

In this project, I measured real power (Watts) instead of apparent power (VA) because the refrigerator runs on AC power, and real power represents the actual energy consumed. Apparent power includes both useful and reactive components, but household energy billing and efficiency analysis are based on real power. Since the smart plug internally adjusts for the power factor (typically around 0.8), logging wattage provided an accurate and practical measure of energy usage. DC power was not relevant here, as the refrigerator is powered by the AC grid.

Issue: Documentation of procedure and Troubleshooting

Document the entire process, including any problems encountered and how they were Resolved.

The entire project was developed through a structured, step-by-step workflow:

1. Device Selection & Planning:
   Choose a refrigerator for its continuous energy usage. Verified current rating and confirmed plug compatibility.

2. Purchasing & Setup:
   Bought a Tuya-compatible smart plug, installed the Tuya Smart app, and linked the device to my Wi-Fi network.

3. Configuration:
   Created a Tuya IoT project, extracted the Local Key, and avoided cloud API due to quota and latency. Used tinytuya for local communication.

4. Dashboard Development:
   Built a Flask-based web app for real-time device control, timer scheduling, and live data visualization. Integrated Chart.js for plotting.

5. Data Logging & Preprocessing:
   Designed a smart logger that saved data only when values changed significantly. Raw values (e.g., milliamps) were converted into standard units (W, V, A) before storing.

6. Hosting:
   Hosted the dashboard using Cloudflare Tunnel with a custom domain:
   https://mumkaftin.dpdns.org

7. Visualization & Extensions:
   Added features like energy history by date, a bill calculator, and a bilingual user manual for usability.

Troubleshooting Faced	Solutions
Local Key not showing	Re-checked pairing mode and used tinytuya wizard with correct region/device type

Device not responding	Verified IP address was static, ensured it stayed connected to Wi-Fi
Logging too frequently	Optimized with conditional logging: only log if power changed >1W or after 2 minutes
Cloud API quota limitations	Avoided entirely by using local key method
Timer cancel not working initially	Fixed by implementing a cancel_timer route
Hosting issues	Solved via Cloudflare tunnel and port forwarding. Before that I tried Hosting via li –port (LocalTunnel) was unstable and ngrok also failed.
tinytuya returns same/stale data when Tuya app is not open	Initially used an Android emulator (BlueStacks) to keep the Tuya app running in background. Later implemented low-level socket polling using Node.js to establish a more reliable Tuya communication without app dependency.

Issue: Wattage of the measuring equipment

Calculate the Wattage of the measuring equipment.

Given ,

Current = 20 A

Voltage = 220 V

Power = 20*220 W

= 4400 W

Issue: Safety Issues

Ensure that all equipment is properly insulated and electrically isolated to prevent any accidents.

To ensure electrical safety, I used a commercially certified Tuya smart plug designed for household AC appliances. This device includes built-in insulation, overcurrent protection, and

surge handling, eliminating the risks associated with handling live wires manually. I avoided DIY circuits (like Arduino + sensors) to prevent exposure to high-voltage components. The setup required no direct wiring—just plug-and-play—ensuring full electrical isolation and making the project safe for continuous use in a home environment.

Issue: Caution with overclocking and flushing new firmware
Remember, safety is paramount when dealing with electrical equipment. Overclocking or flashing a different firmware may cause the device to overheat and potentially cause a fire.

In this project, I did not overclock or flash any custom firmware on the smart plug. I used the device strictly with its original Tuya firmware to maintain manufacturer safety standards. Flashing third-party firmware can void safety certifications, introduce instability, and in worst cases, cause overheating or electrical hazards. Since safety was a priority, I used only official protocols and libraries that communicate externally without modifying the device's internal software.

Issue: API Issues
If an API is used to collect data, ensure that it's working correctly. If not, troubleshoot and resolve any issues. Was the API free? How did you manage to get it? What was the hack?

Initially, I explored using the Tuya Cloud API, but had usage limitations under its free tier (rate limits, token expiry). To avoid these issues, I used a more efficient approach by leveraging Local Key access through the open-source tinytuya library. This allowed me to bypass the cloud API completely and communicate directly with the smart plug over the local network (LAN), ensuring faster, offline, and unlimited access. The key "hack" was extracting the Local Key using tinytuya wizard and implementing socket connection using node and tuya library, which made the setup lightweight and free from cloud dependency.

Issue: User Interface and Experience (UI/UX)
The user interface of the dashboard should be intuitive and easy to use. Consider conducting
user testing to identify any potential issues. Explain how the interface would be easy for non-
technical users to understand.
● Components of the dashboard should display detailed real-time and summarized

historical information on energy consumption. This could include wattage at any minute, energy usage in kWh after each day, daily operating costs, energy usage trend charts, IT load vs. non-IT loads, etc.
● Control Functionality (e.g., simple On/Off, speed control, temperature control) needs to be implemented.

The dashboard was designed with a clean, intuitive interface to ensure accessibility for both technical and non-technical users. The layout uses a hover-based sidebar, large status cards, and clearly labeled controls. Visual feedback (colors, icons, animations) enhances usability without overwhelming the user.

Key features of the UI include:

- Live status cards showing current power (W), voltage (V), and current (A) in real time
- Line charts displaying historical energy trends using Chart.js (updated per date selection)
- A Bill Calculator to estimate daily/monthly energy costs based on real usage
- Control buttons for manual ON/OFF, along with a Smart Timer that allows scheduled automation
- A bilingual User Manual (Bangla and English) using an accordion format to assist all user levels

I also considered non-technical usability by keeping inputs minimal, using visual cues (e.g., timers, status messages), and ensuring that no coding knowledge is needed to operate the system. All features were tested manually, and usability was validated by allowing others to try using it without guidance.

Issue: User Manual
Prepare a user manual for the dashboard and demonstrate how to use it.

To ensure ease of use, I developed a detailed user manual accessible directly from the dashboard sidebar. The manual is written in both English and Bangla, using an accordion-style layout for quick navigation and readability.

The manual includes:

- Device Control: Instructions on turning the plug ON/OFF with real-time status feedback
- Timer Setup: How to set and cancel automated timers for device control
- Real-Time Monitoring: Explanation of live power, voltage, and current graphs that update every 10 seconds
- Energy History: Steps to view daily energy usage trends using the date picker
- Bill Calculator: Guide to estimating DESCO-style monthly bills and viewing breakdowns per day/month
- Sidebar Navigation: How to access each section and switch between functionalities easily

The manual uses simple language and icons, making it suitable for both technical and non-technical users. It was designed to be printable, and includes a button to download as PDF. This ensures the system is user-friendly and self-explanatory without requiring external help.

Issue: Future work and Limitations:

Check if any Energy pattern is detected. Can an Energy profile be developed from this?Could inefficient appliances be identified? Could energy management decisions be taken from this data?

From the collected 24-hour dataset, clear energy patterns were observed, particularly the compressor cycling behavior of the refrigerator, which turned on and off at regular intervals. This indicates the potential to develop an energy profile of the appliance over time.

In the future, extended monitoring over weeks or months could help:

- Identify inefficient appliances (e.g., if a fridge runs constantly or draws unusually high current, it may indicate poor insulation or aging components)
- Set usage thresholds or detect anomalies automatically
- Generate daily energy budgets or compare devices

- Provide data-driven insights to support energy-saving decisions, such as replacing outdated appliances or optimizing usage schedules

However, current limitations include:

- Monitoring is limited to a single device
- No built-in appliance classification or fault detection
- Data is stored locally; no centralized energy platform integration yet
- No machine learning yet for automated profiling or prediction

Issue: Installation, Operation and Maintenance

Consider the maintenance requirements of the system. This could involve regular calibration of sensors, software updates, etc. How often should maintenance be required?

The system was designed to be low-maintenance and easy to operate. Installation involved simply plugging in the Tuya smart plug, connecting it to Wi-Fi via the Tuya Smart app, and linking it to the Flask dashboard through the local network using its Device ID, IP address, and Local Key.

Operation is handled entirely through the browser-based dashboard, requiring no technical expertise. Users can control the device, view live stats, schedule timers, and check historical data effortlessly.

In terms of maintenance:

- Hardware calibration is not needed regularly, as the Tuya smart plug comes factory-calibrated.
- Software maintenance includes:
    - Occasional checks for updates in the tinytuya library
    - Optional UI/UX improvements or bug fixes in the Flask app
    - Ensuring the device's IP remains consistent

Issue: Recurring costs

Consider the cost of installation, maintenance, and operation over the life of the system.

- Installation Cost was a one-time expense for the Tuya smart plug (BDT 1800).
- No Operation Cost .
- No Maintenance Cost
- Hosting Cost is zero because I used Cloudflare Tunnel, which is free for personal projects.

Over the life of the system, recurring costs are effectively zero to minimal, making it highly sustainable for long-term energy monitoring.

Issue: Cost Accounting

SN	Description	Quantity	Unit Cost	Actual Expense Taka	Remarks
3.	Smart Plug	1	1800	1800	Invoice Attached
4.	Transportation	1	30	30	Public Bus
	Total (Taka)			**1830**	

Issue: Business Aspects

Submetering through this project allows users to monitor individual appliance energy usage, identify inefficient devices, and make data-driven decisions to reduce electricity bills. In the context of my refrigerator:

- Average energy usage ≈ 1.5 kWh/day → 45 kWh/month
- Estimated DESCO bill ≈ BDT 500/month for the fridge alone
- If energy-saving actions (e.g., improved usage patterns or replacing inefficient models) reduce usage by even 15–20%, that results in savings of BDT 75–100/month

Cost & Return on Investment (ROI)

Smart Plug	1800
Transportation	30

Assuming BDT 100/month saved, ROI is:

Payback Period = 1,830 ÷ 100 = 18.3 months

After this, the device continues to provide long-term savings and energy insight with no further cost.

Valuation and Market Pricing

If this system were sold as a product or service, I would value it at:

- BDT 2,500–3,000 for a full plug-and-play version with a hosted dashboard and manual
- Justification: It includes not just hardware, but insights, bill estimates, smart control, and historical data—all without monthly fees

Issue: Real-time data or Stored data

The system provides both real-time and stored data. Live readings are updated every 10 seconds with minimal latency using local communication, while all data is also saved in a JSON file for viewing historical trends. This ensures accurate real-time monitoring along with long-term analysis.

Issue: Data Quality

The data collected is of high quality, as it comes directly from a certified Tuya smart plug using local communication, minimizing latency and packet loss. The sampling rate was set to every 120 seconds, which balances accuracy and performance. To improve reliability, logging was triggered (every 25 seconds) only when power changed significantly or after fixed intervals. Overall, the data is clean, consistent, and suitable for analysis.

Issue: Data Security

The system stores data locally in a JSON file on the same machine running the dashboard. No cloud services or third-party servers were used, ensuring privacy. Since the system operates on a local network , there is minimal risk of external interception.

Issue: Scalability

The system can be scaled by adding more Tuya smart plugs and assigning unique IDs and endpoints for each. The Flask backend is modular and can be extended to support multi-device dashboards and higher-frequency data collection if needed.

Issue: Reliability

The system was tested to run continuously over 24 hours without failure. Conditional logging and stable local communication ensured uninterrupted data capture. High-quality, commercially tested smart plugs were used to avoid hardware issues.

Issue: Accuracy

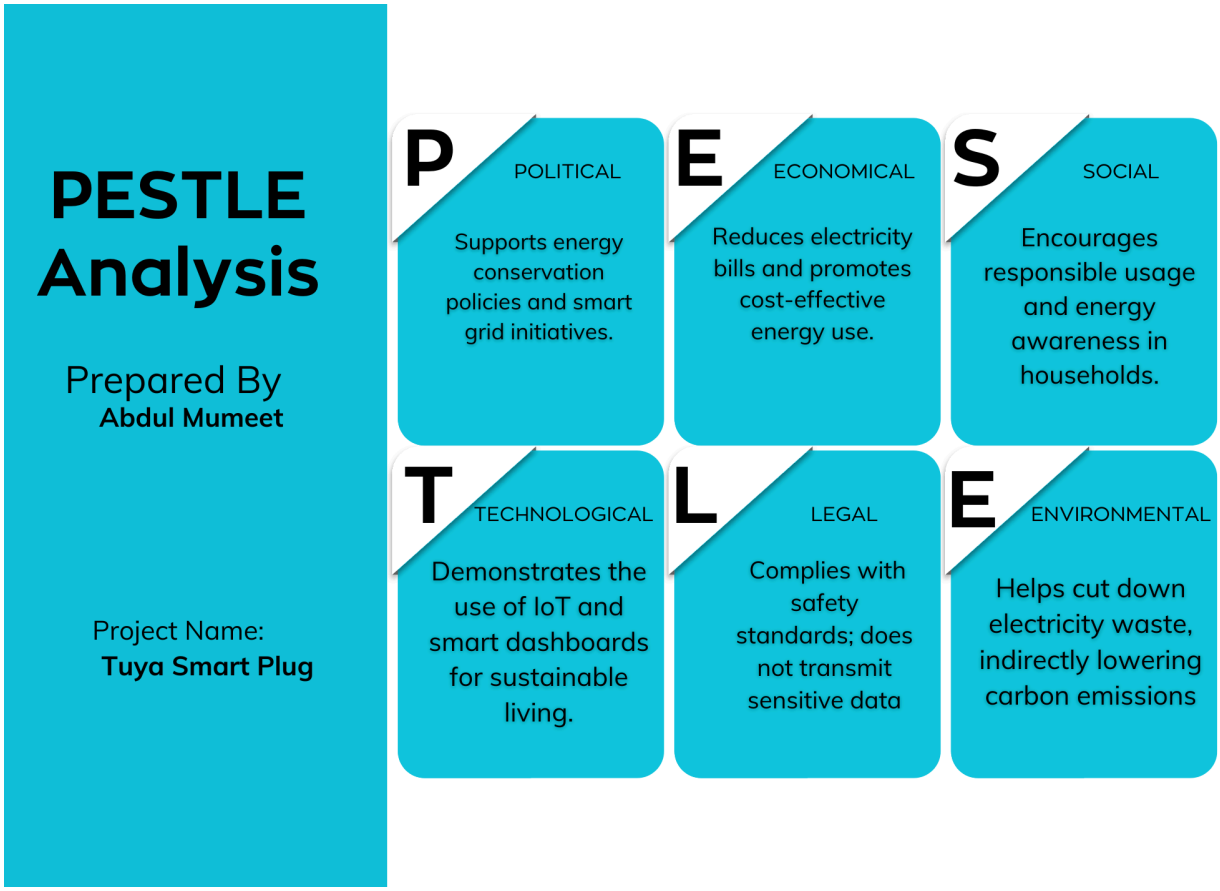
The Tuya smart plug used is pre-calibrated by the manufacturer and provides accurate readings for household-level use. No manual calibration was needed.

Issue: Interoperability

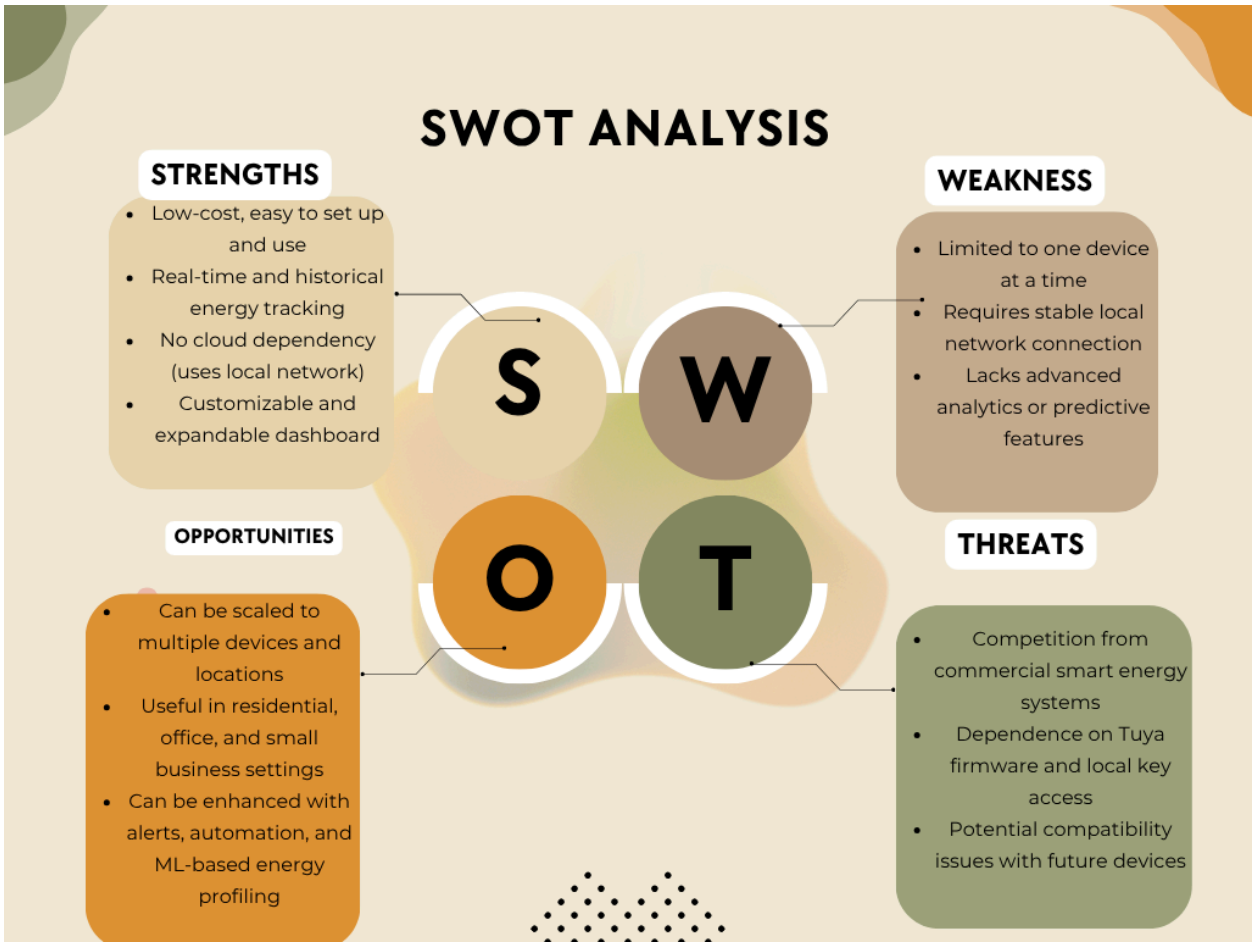
The system is built on standard web technologies (HTTP, JSON) and uses device APIs that can be integrated with other platforms. In future, it can be extended to connect with home automation systems or cloud-based dashboards.

Issue: Environmental Impact

The monitoring system itself has very low energy consumption, making its environmental impact minimal compared to the energy savings it enables through improved awareness and appliance-level monitoring.



Issue: SWOT Analysis



Complex Engineering Problems (EP):

This project qualifies as a complex engineering problem as it requires in-depth knowledge of electrical systems (AC power, power factor), IoT communication (Local Key extraction, socket-based polling), and software development (Flask backend, real-time APIs). EP1 was addressed by applying first-principles understanding to replace limited cloud APIs with efficient local communication. EP2 involved resolving conflicting requirements such as accuracy vs. performance (smart logging), cloud ease vs. local control, and usability vs. technical depth.

Trade-offs were carefully balanced to ensure reliability, real-time monitoring, and user-friendliness within a secure, cost-effective system.

Appendices

GITHUB REPO LINK : https://github.com/Abdul-Mumeet-Pathan/407_Tuya_Smart_Device_Web

Style.css :

```
/* Layout wrapper */
.layout {
    display: flex;
    min-height: 100vh;
}

/* Sidebar Container */
.sidebar {
    width: 60px;
    background: #ffffff;
    color: #333;
    padding: 20px 10px;
    box-sizing: border-box;
    display: flex;
    flex-direction: column;
    align-items: center;
    font-family: 'Segoe UI', sans-serif;
    box-shadow: 4px 0 15px rgba(0, 0, 0, 0.05);
    transition: width 0.3s ease;
    position: relative;
    overflow: hidden;
    border-right: 1px solid #e0e0e0;
```

```css
}


.sidebar:hover {
    width: 220px;
    background: #ffffff;
}
/* Optional: logo/icon */
.sidebar-logo {
    font-size: 24px;
    margin-bottom: 20px;
}

/* Nav links */
.sidebar nav {
    display: flex;
    flex-direction: column;
    gap: 15px;
    width: 100%;
}

.sidebar nav a {
    display: flex;
    align-items: center;
    gap: 10px;
    color: #000000;
    text-decoration: none;
    font-size: 16px;
    font-weight: 500;
    padding: 12px 14px;
    border-radius: 30px;
    transition: all 0.3s ease-in-out;
    background-color: transparent;
    white-space: nowrap;
    position: relative;
}

.sidebar nav a:hover::before {
    opacity: 1;
}
```

```css
.sidebar nav a:hover {
    background-color: #f1f1f1;
    color: #000;
    transform: translateX(5px);
}

.sidebar nav a.active {
    background-color: #ffe3f0;
    color: #d81b60;
    font-weight: 600;
    box-shadow: 0 2px 8px rgba(255, 0, 128, 0.2);
}

.sidebar nav a::before {
    content: '';
    position: absolute;
    inset: 0;
    background: rgba(255, 255, 255, 0.08);
    border-radius: 30px;
    opacity: 0;
    transition: opacity 0.3s ease;
    z-index: 0;
}

/* Text inside nav link */
.link-text {
    opacity: 0;
    transition: opacity 0.3s ease;
    color: #333;
}
/* Show text only on hover */
.link-text {
    opacity: 0;
    transition: opacity 0.3s ease;
    color: #000; /* black text when visible */
}

.sidebar:hover .link-text {
    opacity: 1;
```

```
}
/* Main content area */
.main-content {
    flex: 1;
    padding: 20px;
    box-sizing: border-box;
    background-color: #ffffff;
}

/* Header */
.main-header {
    display: flex;
    justify-content: space-between;
    align-items: center;
    margin-bottom: 25px;
    background-color: #eed7e7;
    color: #030303;
    padding: 15px 20px;
    border-radius: 5px;
    font-weight: bold;
    box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);
}

/* Main dashboard container */
.dashboard {
    display: flex;
    flex-direction: column;
    gap: 40px;
    padding: 20px;
    max-width: 1200px;
    margin: 0 auto;
}

/* Status section */
.status-section {
    background-color: #f8f9fa;
    padding: 30px;
    border-radius: 20px;
    box-shadow: 0 4px 20px rgba(0,0,0,0.1);
    margin-bottom: 30px;
```

```css
}

/* Timer section */
.timer-section {
    background-color: #f8f9fa;
    padding: 30px;
    border-radius: 20px;
    box-shadow: 0 4px 20px rgba(0,0,0,0.1);
    margin-top: 20px;
}

/* Legacy status bar (optional if unused now) */
.status-bar {
    display: flex;
    justify-content: center;
    gap: 2rem;
    padding: 1rem;
    background-color: #ffffff;
    border-radius: 5px;
    margin: 1rem 0;
    flex-wrap: wrap;
}

.status-item {
    display: flex;
    flex-direction: column;
    align-items: center;
    text-align: center;
}

.status-label {
    font-weight: bold;
    margin-bottom: 0.5rem;
    color: #030303;
}

.status-bar span {
    font-weight: bold;
    font-size: 16px;
}
```

```css
/* Modern Status Card Design */
.status-cards {
    display: flex;
    justify-content: space-around;
    flex-wrap: wrap;
    gap: 25px;
    margin-top: 30px;
}

.status-card {
    flex: 1;
    min-width: 220px;
    max-width: 300px;
    height: 180px;
    background: #ffffff;
    border-radius: 20px;
    box-shadow:
        0 10px 20px rgba(0, 0, 0, 0.1),
        inset -3px -3px 6px rgba(255, 255, 255, 0.6),
        inset 3px 3px 8px rgba(0, 0, 0, 0.05);
    padding: 30px 20px;
    text-align: center;
    transition: all 0.3s ease-in-out;
    color: #2c3e50;
    position: relative;
}

.status-card:hover {
    transform: translateY(-5px) scale(1.02);
    box-shadow:
        0 15px 30px rgba(0, 0, 0, 0.25),
        inset -4px -4px 6px rgba(255, 255, 255, 0.5),
        inset 4px 4px 6px rgba(0, 0, 0, 0.1);
}

.status-card h3 {
    font-size: 16px;
    font-weight: bold;
    color: #6c63ff;
```

```
    margin-bottom: 10px;
    text-shadow: 1px 1px 2px rgba(0, 0, 0, 0.1);
}

.status-card p {
    font-size: 26px;
    font-weight: 800;
    color: #111;
    text-shadow: 1px 1px 3px rgba(0,0,0,0.15);
    margin-top: 5px;
}


/* Update the color gradients */
.status-card:nth-child(1) {
    background: linear-gradient(145deg, #ffffff, #e6f0f8);
}

.status-card:nth-child(2) {
    background: linear-gradient(145deg, #ffffff, #fff0ea);
}

.status-card:nth-child(3) {
    background: linear-gradient(145deg, #ffffff, #e4f9fb);
}

.status-card:nth-child(4) {
    background: linear-gradient(145deg, #ffffff, #fff8e1);
}
/* Gradient animation */
@keyframes gradientShift {
    0% { background-position: 0% 50%; }
    50% { background-position: 100% 50%; }
    100% { background-position: 0% 50%; }
}

/* Charts container */
.charts-container {
    display: flex;
    flex-direction: column;
```

```
    gap: 30px;
    padding: 10px;
}

/* Control Panel Buttons */
.control-panel {
    display: flex;
    justify-content: center;
    gap: 20px;
    margin: 20px 0;
}

.btn-on, .btn-off {
    font-size: 16px;
    font-weight: 600;
    padding: 12px 24px;
    border: none;
    border-radius: 30px;
    cursor: pointer;
    transition: all 0.3s ease-in-out;
    box-shadow: 0 4px 12px rgba(0,0,0,0.15);
    color: white;
}


/* Circular Icon Buttons */
.btn-on, .btn-off {
    width: 50px;
    height: 50px;
    border-radius: 50%;
    border: none;
    cursor: pointer;
    display: flex;
    align-items: center;
    justify-content: center;
    box-shadow: 0 4px 8px rgba(0,0,0,0.2);
    transition: all 0.3s ease;
    font-size: 20px;
    color: white;
}
```

```css
.btn-on {
    background: linear-gradient(135deg, #4CAF50, #2E7D32);
}

.btn-off {
    background: linear-gradient(135deg, #F44336, #C62828);
}

.btn-on:hover {
    background: linear-gradient(135deg, #66BB6A, #388E3C);
    transform: translateY(-3px) scale(1.05);
    box-shadow: 0 6px 12px rgba(76, 175, 80, 0.4);
}

.btn-off:hover {
    background: linear-gradient(135deg, #EF5350, #B71C1C);
    transform: translateY(-3px) scale(1.05);
    box-shadow: 0 6px 12px rgba(244, 67, 54, 0.4);
}

/* Power controls container adjustment */
.power-controls {
    display: flex;
    justify-content: center;
    gap: 20px;
    margin: 20px 0;
}

/* Optional: Add pulse animation when active */
@keyframes pulse {
    0% { transform: scale(1); }
    50% { transform: scale(1.1); }
    100% { transform: scale(1); }
}

.btn-on.active {
    animation: pulse 1.5s infinite;
}
```

```css
/* History Page Form Styling */
.history form {
    display: flex;
    justify-content: center;
    align-items: center;
    gap: 15px;
    margin: 30px 0;
    flex-wrap: wrap;
}

.history select {
    padding: 10px 16px;
    font-size: 16px;
    border: none;
    border-radius: 25px;
    background-color: #f1f1f1;
    box-shadow: 0 2px 8px rgba(0, 0, 0, 0.1);
    appearance: none;
    outline: none;
    transition: all 0.3s ease-in-out;
    cursor: pointer;
}

.history select:hover {
    background-color: #e0e0e0;
}

.history button[type="submit"] {
    padding: 10px 22px;
    font-size: 16px;
    font-weight: 600;
    border: none;
    border-radius: 25px;
    background: linear-gradient(to right, #6a11cb, #2575fc);
    color: white;
    cursor: pointer;
    box-shadow: 0 4px 10px rgba(0, 0, 0, 0.2);
    transition: background 0.3s, transform 0.2s;
}
```

```css
.history button[type="submit"]:hover {
    background: linear-gradient(to right, #7b1fa2, #1e88e5);
    transform: translateY(-2px);
}

/* Timer Inputs */
.timer-label {
    font-size: 18px;
    font-weight: bold;
    display: block;
    margin-bottom: 8px;
    color: #333;
    text-align: center;
}

.timer-input {
    font-size: 24px;
    text-align: center;
    padding: 10px 20px;
    width: 100px;
    border-radius: 50%;
    border: 2px solid #888;
    margin: 10px auto;
    display: block;
    box-shadow: inset 0 0 10px rgba(0,0,0,0.1);
    transition: 0.3s ease-in-out;
}

.timer-input:focus {
    outline: none;
    border-color: #4CAF50;
    box-shadow: 0 0 12px rgba(76, 175, 80, 0.4);
}

.countdown-style {
    font-size: 36px;
    font-weight: bold;
    text-align: center;
    padding: 15px;
    color: #ffffff;
```

```
    background: linear-gradient(to right, #ff4081, #7c4dff);
    border-radius: 15px;
    box-shadow: 0 0 15px rgba(0,0,0,0.3);
    margin: 20px auto;
    width: fit-content;
    transition: transform 0.2s;
    animation: pulse 1s infinite ease-in-out;
}

@keyframes pulse {
    0% { transform: scale(1); }
    50% { transform: scale(1.03); }
    100% { transform: scale(1); }
}

/* Smart control center modifications */
.smart-control-center {
    display: flex;
    flex-direction: column;
    gap: 40px;
    padding: 0;
    background-color: transparent;
    box-shadow: none;
    max-width: 100%;
}

/* Power controls */
.power-controls {
    display: flex;
    gap: 15px;
    margin: 15px 0;
    justify-content: center;
}

/* Timer controls */
.timer-control {
    text-align: center;
    background-color: white;
    padding: 25px;
    border-radius: 15px;
```

```
    box-shadow: 0 4px 15px rgba(0,0,0,0.1);
    max-width: 500px;
    margin: 0 auto;
}

/* Countdown display */
.countdown-style {
    margin-top: 20px;
    max-width: 100%;
}

.timer-buttons {
    margin-top: 15px;
    display: flex;
    gap: 15px;
    flex-wrap: wrap;
    justify-content: center;
}

.btn-cancel {
    font-size: 16px;
    font-weight: 600;
    padding: 12px 24px;
    border: none;
    border-radius: 30px;
    cursor: pointer;
    background: linear-gradient(to right, #ff6b6b, #ff4757);
    color: white;
    transition: all 0.3s ease-in-out;
    box-shadow: 0 4px 12px rgba(0,0,0,0.15);
}

.btn-cancel:hover {
    background: linear-gradient(to right, #ff8787, #ff6b6b);
    transform: translateY(-2px);
    box-shadow: 0 6px 16px rgba(255, 107, 107, 0.4);
}

/* Responsive adjustments */
@media (max-width: 768px) {
```

```
    .status-cards {
        gap: 10px;
    }

    .status-card {
        min-width: 100px;
        height: 100px;
        padding: 10px;
    }

    .status-card h3 {
        font-size: 12px;
    }

    .status-card p {
        font-size: 16px;
    }

    .power-controls {
        gap: 10px;
    }

    .btn-on, .btn-off {
        font-size: 13px;
        padding: 6px 12px;
        min-width: 80px;
    }
}
#deviceStatus.loading::after {
    content: '...';
    animation: dots 1.5s steps(3, end) infinite;
}

@keyframes dots {
    0%, 20% { content: ''; }
    40% { content: '.'; }
    60% { content: '..'; }
    80%, 100% { content: '...'; }
}
```

```css
.accordion-container {
    margin-top: 20px;
    display: flex;
    flex-direction: column;
    gap: 15px;
}

.accordion-item {
    background-color: #f4f4f4;
    border-radius: 10px;
    box-shadow: 0 2px 8px rgba(0,0,0,0.1);
    overflow: hidden;
}

.accordion-header {
    padding: 15px 20px;
    font-size: 18px;
    font-weight: bold;
    background-color: #dceefb;
    color: #333;
    cursor: pointer;
    border: none;
    text-align: left;
    width: 100%;
    transition: background 0.3s ease;
}

.accordion-header:hover {
    background-color: #b4dbf4;
}

.accordion-content {
    max-height: 0;
    overflow: hidden;
    padding: 0 20px;
    transition: max-height 0.3s ease;
    background: #fff;
}
```

```css
.accordion-content p,
.accordion-content ul {
    margin: 15px 0;
}


.monthly-table {
    width: 100%;
    max-width: 600px;
    margin: 20px auto;
    border-collapse: collapse;
    background: #fff;
    box-shadow: 0 4px 12px rgba(0,0,0,0.05);
    border-radius: 10px;
    overflow: hidden;
}

.monthly-table th, .monthly-table td {
    padding: 12px 20px;
    text-align: center;
    border-bottom: 1px solid #eee;
}

.monthly-table th {
    background: #0077b6;
    color: #fff;
    font-size: 16px;
}

.monthly-table tr:last-child td {
    border-bottom: none;
}
```

Templates

Base.html :

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>{% block title %}Smart Plug Control{% endblock %}</title>
    <link rel="stylesheet" href="{{ url_for('static',
filename='style.css') }}">
    <script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
</head>
<body>
    <div class="layout">
       <aside class="sidebar">
    <div class="sidebar-logo">🔌</div>
    <nav>
        <a href="{{ url_for('dashboard') }}" class="{% if request.path ==
'/dashboard' %}active{% endif %}" title="Dashboard">
            📊 <span class="link-text">Dashboard</span>
        </a>
        <a href="{{ url_for('history') }}" class="{% if request.path ==
'/history' %}active{% endif %}" title="History">
            📈 <span class="link-text">History</span>
        </a>
        <a href="{{ url_for('user_manual') }}">
            📘</i> <span class="link-text">User Manual</span>
        </a>
        <a href="{{ url_for('bill_calc') }}">
            💵</i> <span class="link-text">Bill Calc</span>
        </a>
    </nav>
</aside>

        <div class="main-content">
            <header class="main-header">
                <h1 style="text-align: center;">Tuya Energy
Dashboard</h1>

            </header>

            <main>
```

```
                {% block content %}{% endblock %}
            </main>

            <footer>
                <p>Tuya Smart Plug Control System &copy; kAFKa</p>
            </footer>
        </div>
    </div>

  {% block scripts %}{% endblock %}
</body>
</html>
```

Bill_calc.html :

```
{% extends "base.html" %}
{% block title %}Bill Calculator{% endblock %}
{% block content %}
<div class="bill-summary-block" style="display: flex; justify-content:
center; gap: 40px; padding: 30px 20px; background: #f1f9ff; border-radius:
20px; box-shadow: 0 6px 15px rgba(0,0,0,0.1); flex-wrap: wrap; max-width:
700px; margin: 20px auto;">

    <div class="bill-block-item" style="flex: 1; min-width: 240px;
padding: 20px; background: white; border-radius: 15px; box-shadow: 0 4px
10px rgba(0,0,0,0.07); text-align: center;">
        <h3 style="margin-bottom: 10px; font-size: 18px; color:
#0077b6;">Total Energy Used</h3>
        <p id="kwhTotal" style="font-size: 24px; font-weight: bold; color:
#222;">-- kWh</p>
    </div>

    <div class="bill-block-item" style="flex: 1; min-width: 240px;
padding: 20px; background: white; border-radius: 15px; box-shadow: 0 4px
10px rgba(0,0,0,0.07); text-align: center;">
        <h3 style="margin-bottom: 10px; font-size: 18px; color:
#0077b6;">Estimated DESCO Bill</h3>
        <p id="billTotal" style="font-size: 24px; font-weight: bold;
color: #222;">-- ৳</p>
```

```
    </div>

</div>

    <h3 style="margin-top: 40px; text-align: center;">📅 Monthly
Breakdown</h3>
    <table class="monthly-table">
        <thead><tr><th>Month</th><th>kWh Used</th><th>📅 View
Daily</th></tr></thead>
        <tbody id="monthlyBreakdown"></tbody>
    </table>

    <div id="dailyBreakdownContainer" style="display:none;">
        <h3 style="text-align:center; margin-top: 40px;">📅 Daily kWh
Usage</h3>
        <table class="monthly-table">
            <thead><tr><th>Date</th><th>kWh</th></tr></thead>
            <tbody id="dailyBreakdown"></tbody>
        </table>
    </div>

{% endblock %}

{% block scripts %}
<script>
fetch('/api/total_kwh')
    .then(res => res.json())
    .then(data => {
        document.getElementById('kwhTotal').textContent = data.total_kwh +
' kWh';
        document.getElementById('billTotal').textContent =
data.estimated_bill_bdt + ' Tk';
    });

fetch('/api/monthly_kwh')
    .then(res => res.json())
    .then(months => {
        const tbody = document.getElementById('monthlyBreakdown');
        Object.entries(months).forEach(([month, kwh]) => {
            const row = `
```

```
            <tr>
              <td>${month}</td>
              <td>${kwh}</td>
              <td><button onclick="loadDaily('${month}')">📅
View</button></td>
            </tr>`;
            tbody.insertAdjacentHTML('beforeend', row);
        });
    });

function loadDaily(month) {
    fetch(`/api/daily_kwh/${month}`)
        .then(res => res.json())
        .then(data => {
            const tbody = document.getElementById('dailyBreakdown');
            tbody.innerHTML = '';
            Object.entries(data).forEach(([date, kwh]) => {
                const row = `<tr><td>${date}</td><td>${kwh}</td></tr>`;
                tbody.insertAdjacentHTML('beforeend', row);
            });

document.getElementById('dailyBreakdownContainer').style.display =
'block';
        });
}
</script>
{% endblock %}
```

Dashboard.html :

```
{% extends "base.html" %}

{% block title %}Dashboard{% endblock %}
{% set show_status_bar = true %}
{% block content %}
<link rel="stylesheet"
href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.4.0/css/all.mi
n.css"/>

<div class="dashboard">
```

```
    <div class="status-section">
        <h2 style="text-align:center; margin-top: 0;">Live Device
Status</h2>
        <div class="status-cards">
            <div class="status-card">
                <h3>Device Status</h3>
                <p id="deviceStatus">Checking device...</p>
            </div>
            <div class="status-card">
                <h3>Voltage</h3>
                <p id="currentVoltage">-- V</p>
            </div>
            <div class="status-card">
                <h3>Power</h3>
                <p id="currentPower">-- W</p>
            </div>
            <div class="status-card">
                <h3>Current</h3>
                <p id="current">-- A</p>
            </div>
        </div>

        <div class="power-controls">
    <button id="btnTurnOn" class="btn-on">
        <i class="fas fa-power-off"></i>
    </button>
    <button id="btnTurnOff" class="btn-off">
        <i class="fas fa-power-off"></i>
    </button>
</div>
    </div>

    <div class="timer-section">
        <div class="timer-control">
            <label for="timerMinutes" class="timer-label">
                <i class="fas fa-clock"></i> Set Timer (minutes)
            </label>
            <input type="number" id="timerMinutes" min="1" value="30"
class="timer-input" />
```

```
            <div class="timer-buttons">
                <button id="toggleTimer" class="btn-on">
                        ⏱
                </button>

                <button id="cancelTimer" class="btn-cancel">
                    ❌ Cancel Timer
                </button>
            </div>
        </div>

        <div id="countdownDisplay" class="countdown-style"></div>
    </div>

    <h2 style="text-align:center;">Real-Time Energy Usage</h2>
    <div class="charts-container">
        <canvas id="powerChart"></canvas>
        <canvas id="voltageChart"></canvas>
        <canvas id="currentChart"></canvas>
    </div>
</div>
{% endblock %}

{% block scripts %}
<script>
const powerChart = new
Chart(document.getElementById('powerChart').getContext('2d'), {
    type: 'line',
    data: {
        labels: [],
        datasets: [{
            label: 'Power (W)',
            data: [],
            borderColor: 'blue',
            fill: false
        }]
    },
    options: {
        responsive: true,
        scales: { y: { beginAtZero: true } }
```

```
    }
});

const voltageChart = new
Chart(document.getElementById('voltageChart').getContext('2d'), {
    type: 'line',
    data: {
        labels: [],
        datasets: [{
            label: 'Voltage (V)',
            data: [],
            borderColor: 'green',
            fill: false
        }]
    },
    options: {
        responsive: true,
        scales: { y: { beginAtZero: true } }
    }
});

const currentChart = new
Chart(document.getElementById('currentChart').getContext('2d'), {
    type: 'line',
    data: {
        labels: [],
        datasets: [{
            label: 'Current (A)',
            data: [],
            borderColor: 'red',
            fill: false
        }]
    },
    options: {
        responsive: true,
        scales: { y: { beginAtZero: true } }
    }
});

async function updateData() {
```

```
    const res = await fetch('/api/status');
    const data = await res.json();
    const now = new Date().toLocaleTimeString();

    const statusDiv = document.getElementById('deviceStatus');
    const voltageSpan = document.getElementById('currentVoltage');
    const powerSpan = document.getElementById('currentPower');
    const currentSpam = document.getElementById('current');

    if (data.connected) {
        if (statusDiv) statusDiv.textContent = "Device Connected ✅";
        if (voltageSpan) voltageSpan.textContent = data.voltage ?
data.voltage.toFixed(1) + ' V' : '-- V';
        if (powerSpan) powerSpan.textContent = data.power ?
data.power.toFixed(2) + ' W' : '-- W';
        if (currentSpam) currentSpam.textContent = data.current ?
data.current.toFixed(2) + ' A' : '-- A';

        powerChart.data.labels.push(now);
        powerChart.data.datasets[0].data.push(data.power);
        voltageChart.data.labels.push(now);
        voltageChart.data.datasets[0].data.push(data.voltage);
        currentChart.data.labels.push(now);
        currentChart.data.datasets[0].data.push(data.current);

        if (powerChart.data.labels.length > 50) {
            powerChart.data.labels.shift();
            powerChart.data.datasets[0].data.shift();
            voltageChart.data.labels.shift();
            voltageChart.data.datasets[0].data.shift();
            currentChart.data.labels.shift();
            currentChart.data.datasets[0].data.shift();
        }

        powerChart.update();
        voltageChart.update();
        currentChart.update();
    } else {
        if (statusDiv) statusDiv.textContent = "Device Not Connected ❌";
        if (voltageSpan) voltageSpan.textContent = '-- V';
```

```
        if (powerSpan) powerSpan.textContent = '-- W';
        if (currentSpam) currentSpam.textContent = '-- A';
    }
}

document.getElementById('btnTurnOn').addEventListener('click', async () =>
{
    const statusDiv = document.getElementById('deviceStatus');
    statusDiv.textContent = "Turning ON...";
    try {
        const res = await fetch('/api/turn_on', { method: 'POST' });
        const data = await res.json();
        statusDiv.textContent = data.message;
    } catch (error) {
        statusDiv.textContent = "Error turning ON device.";
    }
});

document.getElementById('btnTurnOff').addEventListener('click', async ()
=> {
    const statusDiv = document.getElementById('deviceStatus');
    statusDiv.textContent = "Turning OFF...";
    try {
        const res = await fetch('/api/turn_off', { method: 'POST' });
        const data = await res.json();
        statusDiv.textContent = data.message;
    } catch (error) {
        statusDiv.textContent = "Error turning OFF device.";
    }
});

setInterval(updateData, 10000);
updateData();

let countdownInterval = null;
function startCountdown(minutes, actionType) {
    clearInterval(countdownInterval); // clear previous countdown
    const countdownDisplay = document.getElementById('countdownDisplay');
    let totalSeconds = minutes * 60;
```

```
    countdownInterval = setInterval(() => {
        const mins = Math.floor(totalSeconds / 60);
        const secs = totalSeconds % 60;

    countdownDisplay.textContent = `${String(mins).padStart(2,
'0')}:${String(secs).padStart(2, '0')}`;

        if (totalSeconds <= 0) {
            clearInterval(countdownInterval);
            //countdownDisplay.textContent = `✅ Device should be
${actionType === 'on' ? 'ON' : 'OFF'} now.`;

    // Hide after 5 seconds
    setTimeout(() => {
        countdownDisplay.textContent = '';
    }, 5000);
        }

        totalSeconds--;
    }, 1000);
}



document.getElementById('toggleTimer').addEventListener('click', async ()
=> {
    const minutes =
parseInt(document.getElementById('timerMinutes').value);
    const statusDiv = document.getElementById('deviceStatus');
    if (!minutes || minutes < 0) return alert("Please enter valid
minutes");

    try {
        console.log("📡 Fetching status...");
        const res = await fetch('/api/status');
        const statusData = await res.json();
        console.log("📊 Status data:", statusData);

        const isDeviceRunning = statusData.is_on;
```

```
        const endpoint = isDeviceRunning ? '/api/timer_off' :
'/api/timer_on';

        console.log("🧭 Target timer endpoint:", endpoint);

        const timerRes = await fetch(endpoint, {
            method: 'POST',
            headers: { 'Content-Type': 'application/json' },
            body: JSON.stringify({ minutes })
        });

        const timerData = await timerRes.json();
        console.log("✅ Timer API response:", timerData);

        statusDiv.textContent = timerData.message || "⚠️ Timer set but no
message returned";
        startCountdown(minutes, isDeviceRunning ? 'off' : 'on');

    } catch (err) {
        console.error("❌ Timer setup error:", err);
        statusDiv.textContent = "❌ Failed to set smart timer.";
    }
});



document.getElementById('cancelTimer').addEventListener('click', async ()
=> {
    clearInterval(countdownInterval);  // stop local timer

    try {
        const res = await fetch('/api/cancel_timer', { method: 'POST' });
        const data = await res.json();
        const countdownDisplay =
document.getElementById('countdownDisplay');
countdownDisplay.textContent = '⛔ ' + data.message;

// Auto-hide after 4 seconds
setTimeout(() => {
    countdownDisplay.textContent = '';
```

```
}, 4000);
    } catch (error) {
        document.getElementById('countdownDisplay').textContent = '❌
Failed to cancel timer.';
    }
});

</script>
{% endblock %}
```

History.html :

```
{% extends "base.html" %}

{% block title %} History{% endblock %}

{% block content %}
<div class="history">
    <h2> History</h2>

    <form method="GET" action="{{ url_for('history') }}">
        <select name="date" required>
            <option value="" disabled {% if not selected_date %}selected{%
endif %}>-- Select a date --</option>
            {% for date in dates %}
                <option value="{{ date }}" {% if selected_date == date
%}selected{% endif %}>{{ date }}</option>
            {% endfor %}
        </select>
        <button type="submit">Show</button>
    </form>

    {% if selected_date %}
        <h3>Data for {{ selected_date }}</h3>

        {% if times %}
            <canvas id="powerChart"></canvas>
            <canvas id="voltageChart"></canvas>
            <canvas id="currentChart"></canvas>
        {% else %}
```

```
            <div class="no-data">No data found for this date.</div>
        {% endif %}
    {% endif %}
</div>
{% endblock %}

{% block scripts %}
{% if selected_date and times %}
<script>
const labels = {{ times | tojson }};
const powers = {{ powers | tojson }};
const volts = {{ volts | tojson }};
const currents = {{ currents | tojson }};

const chartOptions = {
    responsive: true,
    scales: { y: { beginAtZero: true } },
    plugins: {
        legend: { display: true, position: 'top' },
        tooltip: { mode: 'index', intersect: false }
    },
    interaction: { mode: 'nearest', intersect: false }
};

new Chart(document.getElementById('powerChart').getContext('2d'), {
    type: 'line',
    data: {
        labels: labels,
        datasets: [{
            label: 'Power (W)',
            data: powers,
            borderColor: 'blue',
            fill: false,
            pointRadius: 2,
            tension: 0
        }]
    },
    options: chartOptions
});
```

```
new Chart(document.getElementById('voltageChart').getContext('2d'), {
    type: 'line',
    data: {
        labels: labels,
        datasets: [{
            label: 'Voltage (V)',
            data: volts,
            borderColor: 'green',
            fill: false,
            pointRadius: 2,
            tension: 0
        }]
    },
    options: chartOptions
});

new Chart(document.getElementById('currentChart').getContext('2d'), {
    type: 'line',
    data: {
        labels: labels,
        datasets: [{
            label: 'Current (A)',
            data: currents,
            borderColor: 'red',
            fill: false,
            pointRadius: 2,
            tension: 0
        }]
    },
    options: chartOptions
});
</script>
{% endif %}
{% endblock %}
```

User_manual.html :

```
{% extends "base.html" %}

{% block title %}User Manual{% endblock %}
```

```
{% block content %}
<div class="main-content">
    <h2>📘 User Manual / ব্যবহার নির্দেশিকা</h2>

    <div style="display: flex; justify-content: space-between;
align-items: center; flex-wrap: wrap;">
        <button onclick="window.print()" style="margin: 10px 0; padding:
8px 16px; font-size: 16px;">
            🖨️ Print / 📄 Download as PDF
        </button>

        <button id="toggleLang" style="margin: 10px 0; padding: 8px 16px;
font-size: 16px;">
            🌐 Switch to বাংলা
        </button>
    </div>

    <div class="accordion-container">

        <div class="accordion-item">
            <button class="accordion-header">🔌 Device Control</button>
            <div class="accordion-content">
                <p class="en">Use the ON/OFF buttons to manually control
your smart plug. The live status updates show voltage, power, and
current.</p>
                <p class="bn" style="display: none;">স্মার্ট প্লাগ অন/অফ করার জন্য
ON ও OFF বাটন ব্যবহার করুন। লাইভ স্ট্যাটাসে ভোল্টেজ, পাওয়ার ও কারেন্ট দেখায়।</p>
            </div>
        </div>

        <div class="accordion-item">
            <button class="accordion-header">⏱️ Timer</button>
            <div class="accordion-content">
                <p class="en">Set a timer to auto turn ON/OFF the device
after a number of minutes.</p>
                <p class="bn" style="display: none;">নির্দিষ্ট মিনিট পর ডিভাইস চালু
বা বন্ধ করতে টাইমার ব্যবহার করুন।</p>
            </div>
        </div>
```

```
        <div class="accordion-item">
            <button class="accordion-header">📊 Real-Time
Monitoring</button>
            <div class="accordion-content">
                <p class="en">Power, voltage, and current graphs update
every 10 seconds live from the smart device.</p>
                <p class="bn" style="display: none;">পাওয়ার, ভোল্টেজ ও কারেন্ট
এর চার্ট প্রতি ১০ সেকেন্ডে হালনাগাদ হয়।</p>
            </div>
        </div>

        <div class="accordion-item">
            <button class="accordion-header">📅 Energy History</button>
            <div class="accordion-content">
                <p class="en">View logs of previous dates. Select a date
to see that day's data.</p>
                <p class="bn" style="display: none;">আগের দিনের তথ্য দেখতে
তারিখ সিলেক্ট করুন। চার্ট আকারে দেখাবে।</p>
            </div>
        </div>

        <div class="accordion-item">
            <button class="accordion-header">💰 Bill Calculator</button>
            <div class="accordion-content">
                <p class="en">
                    The Bill Calculator lets you:
                    <ul class="en">
                        <li>📅 View total energy usage (kWh)</li>
                        <li>💵 Estimate monthly DESCO bill in Taka</li>
                        <li>🗓️ View per-month usage</li>
                        <li>📅 Drill down into per-day usage</li>
                    </ul>
                </p>
                <p class="bn" style="display: none;">
                    বিল ক্যালকুলেটর দ্বারা আপনি দেখতে পারবেন:
                    <ul class="bn" style="display: none;">
                        <li>📅 মোট বিদ্যুৎ ব্যবহার (kWh)</li>
                        <li>💵 মাসিক DESCO বিলের আনুমানিক পরিমাণ (৳)</li>
                        <li>🗓️ প্রতি মাসের বিদ্যুৎ ব্যবহার</li>
                        <li>📅 নির্দিষ্ট দিনের হিসাব</li>
```

```
                    </ul>
                </p>
            </div>
        </div>

        <div class="accordion-item">
            <button class="accordion-header">🧭 Sidebar
Navigation</button>
            <div class="accordion-content">
                <p class="en">Hover on the sidebar to see menu items like
Dashboard, History, Manual, and Bill Calculator.</p>
                <p class="bn" style="display: none;">সাইডবারে মাউস রাখলে মেনু
আইটেমগুলো দেখা যাবে যেমন: ড্যাশবোর্ড, হিস্টোরি, ম্যানুয়াল, এবং বিল ক্যালকুলেটর।</p>
            </div>
        </div>

    </div>
</div>
{% endblock %}

{% block scripts %}
<script>
    // Accordion behavior
    const accordions = document.querySelectorAll(".accordion-header");
    accordions.forEach(btn => {
        btn.addEventListener("click", () => {
            btn.classList.toggle("active");
            const content = btn.nextElementSibling;
            if (content.style.maxHeight) {
                content.style.maxHeight = null;
            } else {
                content.style.maxHeight = content.scrollHeight + "px";
            }
        });
    });

    // Language toggle
    const toggleBtn = document.getElementById("toggleLang");
    let currentLang = 'en';
```

```
    toggleBtn.addEventListener("click", () => {
        const enText = document.querySelectorAll(".en");
        const bnText = document.querySelectorAll(".bn");

        if (currentLang === 'en') {
            enText.forEach(el => el.style.display = 'none');
            bnText.forEach(el => el.style.display = 'block');
            toggleBtn.innerText = "🌐 Switch to English";
            currentLang = 'bn';
        } else {
            enText.forEach(el => el.style.display = 'block');
            bnText.forEach(el => el.style.display = 'none');
            toggleBtn.innerText = "🌐 Switch to বাংলা";
            currentLang = 'en';
        }
    });
</script>
{% endblock %}
```

Tuyapi-module :

Package-lock.json :

```
{
  "name": "tuyapi-node",
  "version": "1.0.0",
  "lockfileVersion": 3,
  "requires": true,
  "packages": {
    "": {
      "name": "tuyapi-node",
      "version": "1.0.0",
      "license": "ISC",
      "dependencies": {
        "chalk": "^5.4.1",
        "tuyapi": "^7.7.1"
      }
    },
    "node_modules/@types/retry": {
```

```
      "version": "0.12.0",
      "resolved":
"https://registry.npmjs.org/@types/retry/-/retry-0.12.0.tgz",
      "integrity":
"sha512-wWKOClTTiizcZhXnPY4wikVAwmdYHp8q6DmC+EJUzAMsycb7HB32Kh9RN4+0gExjmP
mZSAQjgURXIGATPegAvA==",
      "license": "MIT"
    },
    "node_modules/chalk": {
      "version": "5.4.1",
      "resolved": "https://registry.npmjs.org/chalk/-/chalk-5.4.1.tgz",
      "integrity":
"sha512-zgVZuo2WcZgfUEmsn6eO3kINexW8RAE4maiQ8QNs8CtpPCSyMiYsULR3HQYkm3w8FI
A3SberyMJMSldGsW+U3w==",
      "license": "MIT",
      "engines": {
        "node": "^12.17.0 || ^14.13 || >=16.0.0"
      },
      "funding": {
        "url": "https://github.com/chalk/chalk?sponsor=1"
      }
    },
    "node_modules/debug": {
      "version": "4.4.1",
      "resolved": "https://registry.npmjs.org/debug/-/debug-4.4.1.tgz",
      "integrity":
"sha512-KcKCqiftBJcZr++7ykoDIEwSa3XWowTfNPo92BYxjXiyYEVrUQh2aLyhxBCwww+heo
rtUFxEJYcRzosstTEBYQ==",
      "license": "MIT",
      "dependencies": {
        "ms": "^2.1.3"
      },
      "engines": {
        "node": ">=6.0"
      },
      "peerDependenciesMeta": {
        "supports-color": {
          "optional": true
        }
      }
```

```
    },
    "node_modules/eventemitter3": {
      "version": "4.0.7",
      "resolved":
"https://registry.npmjs.org/eventemitter3/-/eventemitter3-4.0.7.tgz",
      "integrity":
"sha512-8guHBZCwKnFhYdHr2ysuRWErTwhoN2X8XELRlrRwpmfeY2jjuUN4taQMsULKUVo1K4
DvZl+0pgfyoysHxvmvEw==",
      "license": "MIT"
    },
    "node_modules/ms": {
      "version": "2.1.3",
      "resolved": "https://registry.npmjs.org/ms/-/ms-2.1.3.tgz",
      "integrity":
"sha512-6FlzubTLZG3J2a/NVCAleEhjzq5oxgHyaCU9yYXvcLsvoVaHJq/s5xXI6/XXP6tz7R
9xAOtHnSO/tXtF3WRTlA==",
      "license": "MIT"
    },
    "node_modules/p-finally": {
      "version": "1.0.0",
      "resolved":
"https://registry.npmjs.org/p-finally/-/p-finally-1.0.0.tgz",
      "integrity":
"sha512-LICb2p9CB7FS+0eR1oqWnHhp0FljGLZCWBE9aix0Uye9W8LTQPwMTYVGWQWIw9RdQi
Dg4+epXQODwIYJtSJaow==",
      "license": "MIT",
      "engines": {
        "node": ">=4"
      }
    },
    "node_modules/p-queue": {
      "version": "6.6.2",
      "resolved":
"https://registry.npmjs.org/p-queue/-/p-queue-6.6.2.tgz",
      "integrity":
"sha512-RwFpb72c/BhQLEXIZ5K2e+AhgNVmIejGlTgiB9MzZ0e93GRvqZ7uSi0dvRF7/XIXDe
Nkra2fNHBxTyPDGySpjQ==",
      "license": "MIT",
      "dependencies": {
        "eventemitter3": "^4.0.4",
```

```
        "p-timeout": "^3.2.0"
      },
      "engines": {
        "node": ">=8"
      },
      "funding": {
        "url": "https://github.com/sponsors/sindresorhus"
      }
    },
    "node_modules/p-retry": {
      "version": "4.6.2",
      "resolved":
"https://registry.npmjs.org/p-retry/-/p-retry-4.6.2.tgz",
      "integrity":
"sha512-312Id396EbJdvRONlngUx0NydfrIQ5lsYu0znKVUzVvArzEIt08V1qhtyESbGVd1FG
X7UKtiFp5uwKZdM8wIuQ==",
      "license": "MIT",
      "dependencies": {
        "@types/retry": "0.12.0",
        "retry": "^0.13.1"
      },
      "engines": {
        "node": ">=8"
      }
    },
    "node_modules/p-timeout": {
      "version": "3.2.0",
      "resolved":
"https://registry.npmjs.org/p-timeout/-/p-timeout-3.2.0.tgz",
      "integrity":
"sha512-rhIwUycgwwKcP9yTOOFK/AKsAopjjCakVqLHePO3CC6Mir1Z99xT+R63jZxAT5lFZL
a2inS5h+ZS2GvR99/FBg==",
      "license": "MIT",
      "dependencies": {
        "p-finally": "^1.0.0"
      },
      "engines": {
        "node": ">=8"
      }
    },
```

```
    "node_modules/retry": {
      "version": "0.13.1",
      "resolved": "https://registry.npmjs.org/retry/-/retry-0.13.1.tgz",
      "integrity":
"sha512-XQBQ3I8W1Cge0Seh+6gjj03LbmRFWuoszgK9ooCpwYIrhhoO80pfq4cUkU5DkknwfO
fFteRwlZ56PYOGYyFWdg==",
      "license": "MIT",
      "engines": {
        "node": ">= 4"
      }
    },
    "node_modules/tuyapi": {
      "version": "7.7.1",
      "resolved": "https://registry.npmjs.org/tuyapi/-/tuyapi-7.7.1.tgz",
      "integrity":
"sha512-aJHaW0WOhW5y1XLvOOy2G6/SGMUbCTkKgfF8ub8YyhrGkTR6abSB4YNJBXaILr+4UF
CBv3cF2cfVjXMeUQ0vYg==",
      "license": "MIT",
      "dependencies": {
        "debug": "^4.4.0",
        "p-queue": "6.6.2",
        "p-retry": "4.6.2",
        "p-timeout": "3.2.0"
      }
    }
  }
}
```

Package.json :

```
{
  "name": "tuyapi-node",
  "version": "1.0.0",
  "description": "",
  "main": "tuya-monitor.js",
  "type": "module",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
```

```
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {
    "chalk": "^5.4.1",
    "tuyapi": "^7.7.1"
  }
}
```

tuya-monitor.js :

```
// const TuyAPI = require('tuyapi');
// const chalk = require('chalk').default;

import TuyAPI from 'tuyapi';
import chalk from 'chalk';
// Configuration - REPLACE WITH YOUR VALUES
const deviceConfig = {
  id: "bf493814f4d1067dcbqvx5",       // Tuya device ID
  key: "2V2W+q}UA;pF~'Cb",            // Tuya local key
  ip: "192.168.0.111",                // Local IP (optional, but
recommended)
  version: '3.5'                      // Device protocol version: 3.1, 3.3,
or 3.5
};

// Initialize device
const device = new TuyAPI({
  ...deviceConfig,
  issueRefreshOnConnect: true,
  issueGetOnConnect: true,
  connectionTimeout: 2000,
  persistentConnection: false
});

// Global error listener (prevents crash on unplug)
device.on('error', (err) => {
  console.log(chalk.red('Device emitted an error:'), err.message);
  handleConnectionError(err);
```

```
  lastValues.status = 'disconnected';
  lastValues.timestamp = new Date();
  console.log(formatStatus(lastValues));
});

let monitoringActive = true;
let connectionAttempts = 0;

let lastValues = {
  power: 0,
  voltage: 0,
  current: 0,
  timestamp: null,
  status: 'disconnected'
};

function formatValue(value, unit) {
  const formatted = value.toFixed(unit === 'A' ? 3 : unit === 'V' ? 1 :
2);
  if (unit === 'W') return chalk.green(`${formatted}${unit}`);
  if (unit === 'V') return chalk.blue(`${formatted}${unit}`);
  if (unit === 'A') return chalk.red(`${formatted}${unit}`);
  return formatted;
}

function formatStatus(data) {
  const time = data.timestamp?.toLocaleTimeString() || '--:--:--';
  const statusIcon = data.status === 'connected' ? chalk.green('✓') :
                     data.status === 'error' ? chalk.red('✖') :
chalk.yellow('⌛');
  return [
    chalk.gray(`[${time}]`),
    'Power:', formatValue(data.power, 'W'),
    '| Voltage:', formatValue(data.voltage, 'V'),
    '| Current:', formatValue(data.current, 'A'),
    statusIcon,
    data.status === 'error' ? chalk.red('(retrying)') : ''
  ].join(' ');
}
```

```
// Handle connection errors - unlimited retries, no maxAttempts limit
function handleConnectionError(error) {
  connectionAttempts++;
  lastValues.status = 'error';

  const shortMsg = error?.message || error.toString();
  console.log(chalk.yellow(`Connection attempt ${connectionAttempts}:
${shortMsg}`));
}

async function forceRefresh() {
  try {
    if (device.isConnected) {
      try {
        await device.disconnect();
      } catch (disconnectError) {
        console.log(chalk.yellow(`Disconnect warning:
${disconnectError.message}`));
      }
    }

    if (!device.ip) {
      try {
        await device.find();
      } catch (findError) {
        handleConnectionError(findError);
        lastValues.status = 'disconnected';
        lastValues.timestamp = new Date();
        console.log(formatStatus(lastValues));
        return;
      }
    }

    try {
      await device.connect();
    } catch (connectError) {
      handleConnectionError(connectError);
      lastValues.status = 'disconnected';
      lastValues.timestamp = new Date();
      console.log(formatStatus(lastValues));
```

```
      return;
    }

    connectionAttempts = 0; // Reset after successful connection
    lastValues.status = 'connected';

    let data;
    try {
      data = await device.get({ schema: true, force: true });
    } catch (getError) {
      handleConnectionError(getError);
      lastValues.status = 'error';
      lastValues.timestamp = new Date();
      console.log(formatStatus(lastValues));
      return;
    }

    const dps = data.dps || {};
    const newValues = {
      power: dps['19'] !== undefined ? dps['19'] / 10 : lastValues.power,
      voltage: dps['20'] !== undefined ? dps['20'] / 10 :
lastValues.voltage,
      current: dps['18'] !== undefined ? dps['18'] / 1000 :
lastValues.current,
      timestamp: new Date(),
      status: 'connected'
    };

    if (JSON.stringify(newValues) !== JSON.stringify(lastValues)) {
      lastValues = newValues;
      console.log(formatStatus(lastValues));
    }

  } catch (unexpectedError) {
    handleConnectionError(unexpectedError);
    lastValues.status = 'error';
    lastValues.timestamp = new Date();
    console.log(formatStatus(lastValues));
  } finally {
    try {
```

```
      if (device.isConnected) {
        await device.disconnect();
      }
    } catch (cleanupError) {
      console.log(chalk.yellow(`Cleanup warning:
${cleanupError.message}`));
    }
  }
}

async function monitor() {
  console.log(chalk.bold.cyan('Tuya Smart Plug Monitor - Resilient
Version'));
  console.log(chalk.gray('Press Ctrl+C to exit\n'));

  await forceRefresh();

  const interval = setInterval(async () => {
    if (monitoringActive) {
      await forceRefresh();
    }
  }, 2000); // every 5 seconds

  process.on('SIGINT', async () => {
    monitoringActive = false;
    clearInterval(interval);
    try {
      if (device.isConnected) {
        await device.disconnect();
      }
    } catch (error) {
      console.log(chalk.yellow(`Exit cleanup error: ${error.message}`));
    }
    console.log(chalk.cyan('\nMonitoring stopped. Goodbye!'));
    process.exit();
  });
}

(async () => {
  try {
```

```
    await monitor();
  } catch (error) {
    console.error(chalk.red('Fatal monitoring error:'), error);
    process.exit(1);
  }
})();
```

app.py :

```
from flask import Flask, render_template, jsonify, request, redirect,
url_for
from datetime import datetime, timedelta
import json
import os
import atexit
from tuya_control import TuyaDevice
from config import DATA_FILES

from flask import Flask, jsonify, request
import threading
import time


app = Flask(__name__)
device = TuyaDevice()

active_timer = {
    "on": None,
    "off": None
}
# Initialize data files if they don't exist
if not os.path.exists('data'):
    os.makedirs('data')

if not os.path.exists(DATA_FILES['energy_logs']):
    with open(DATA_FILES['energy_logs'], 'w') as f:
        json.dump([], f)

if not os.path.exists(DATA_FILES['device_status']):
    with open(DATA_FILES['device_status'], 'w') as f:
```

```
        json.dump({'status': 'unknown'}, f)

# Replace log_energy_data() with:
def log_energy_data():
    """Log only if values changed significantly"""
    data = device.get_status()
    if not data.get('connected'):
        return

    with open(DATA_FILES['energy_logs'], 'r+') as f:
        logs = json.load(f)
        last_log = logs[-1] if logs else None

        # Only log if power changed by >1W or 5 minutes passed
        should_log = (
            not last_log or
            abs(data['power'] - last_log['power']) > 1 or
            (datetime.now() - datetime.strptime(last_log['timestamp'],
'%Y-%m-%d %H:%M:%S')).seconds > 120
        )

        if should_log:
            logs.append({
                'timestamp': datetime.now().strftime('%Y-%m-%d %H:%M:%S'),
                'power': round(data['power'], 2),
                'voltage': round(data['voltage'], 1),
                'current': round(data['current'], 3)
            })
            f.seek(0)
            json.dump(logs, f, indent=2)

def get_energy_logs(date=None):
    """Get energy logs, optionally filtered by date"""
    with open(DATA_FILES['energy_logs'], 'r') as f:
        logs = json.load(f)

    if date:
        return [log for log in logs if log['timestamp'].startswith(date)]
    return logs
```

```
def get_available_dates():
    """Get unique dates with logged data"""
    with open(DATA_FILES['energy_logs'], 'r') as f:
        logs = json.load(f)

    dates = set()
    for log in logs:
        date = log['timestamp'].split()[0]
        dates.add(date)

    return sorted(dates, reverse=True)[:30]  # Last 30 days

@app.route('/')
def dashboard():
    return render_template('dashboard.html')

@app.route('/history')
def history():
    date = request.args.get('date')
    dates = get_available_dates()

    if date:
        logs = get_energy_logs(date)
        times = [log['timestamp'].split()[1][:5] for log in logs]
        powers = [log['power'] for log in logs]
        volts = [log['voltage'] for log in logs]
        currents = [log['current'] for log in logs]
    else:
        logs = []
        times = powers = volts = currents = []

    return render_template('history.html',
                           dates=dates,
                           selected_date=date,
                           times=times,
                           powers=powers,
                           volts=volts,
                           currents=currents)

@app.route('/api/status')
```

```python
def get_status():
    try:
        data = device.get_status()
        if data and data.get('connected'):
            log_energy_data()  # Log data each time we check status
            return jsonify({
                'connected': True,
                'status': 'connected',
                'voltage': data.get('voltage', 0),
                'power': data.get('power', 0),
                'current': data.get('current', 0),
                'is_on': data.get('power', 0) > 1  # <-- Add this
            })
        else:
            return jsonify({
                'connected': False,
                'status': 'disconnected',
                'voltage': None,
                'power': None,
                'current': None
            })
    except Exception as e:
        print(f"Error getting device status: {e}")
        return jsonify({
            'connected': False,
            'status': 'error',
            'voltage': None,
            'power': None,
            'current': None
        }), 500

@app.route('/api/turn_on', methods=['POST'])
def turn_on():
    result = device.turn_on()
    return jsonify(result)

@app.route('/api/turn_off', methods=['POST'])
def turn_off():
    result = device.turn_off()
    return jsonify(result)
```

```python
@app.teardown_appcontext
def shutdown(exception=None):
    device.stop_keep_alive()

@app.route('/api/timer_on', methods=['POST'])
def timer_on():
    data = request.get_json()
    minutes = data.get('minutes', 0)
    delay = minutes * 60

    def turn_on():
        print("⏰ Timer reached: Turning device ON")
        device.turn_on()
        active_timer["on"] = None  # clear ref after execution

    if active_timer["on"]:
        active_timer["on"].cancel()

    t = threading.Timer(delay, turn_on)
    t.start()
    active_timer["on"] = t

    return jsonify({"message": f"Timer set: Device will turn ON in
{minutes} minute(s) ✅"})

@app.route('/api/timer_off', methods=['POST'])
def timer_off():
    data = request.get_json()
    minutes = data.get('minutes', 0)
    delay = minutes * 60

    def turn_off():
        print("⏰ Timer reached: Turning device OFF")
        device.turn_off()
        active_timer["off"] = None  # clear ref after execution

    if active_timer["off"]:
        active_timer["off"].cancel()
```

```
    t = threading.Timer(delay, turn_off)
    t.start()
    active_timer["off"] = t

    return jsonify({"message": f"Timer set: Device will turn OFF in
{minutes} minute(s) ✅"})

@app.route('/api/cancel_timer', methods=['POST'])
def cancel_timer():
    # Cancel the scheduled job/timer from your system
    cancel_scheduled_task()  # <-- implement this logic!
    return jsonify({"message": "Timer cancelled on server."})

def cancel_scheduled_task():
    for key in ["on", "off"]:
        if active_timer[key]:
            active_timer[key].cancel()
            print(f"🛑 Cancelled {key.upper()} timer")
            active_timer[key] = None


#user manual
@app.route('/manual')
def user_manual():
    return render_template('user_manual.html')




#BILL ____________________________________

# ---------- app.py (add at bottom) ----------
@app.route('/bill')
def bill_calc():
    return render_template('bill_calc.html')


@app.route('/api/total_kwh')
def total_kwh():
```

```
    logs = get_energy_logs()
    total_kwh = 0

    for i in range(1, len(logs)):
        try:
            p1 = logs[i - 1]['power']
            p2 = logs[i]['power']
            t1 = datetime.strptime(logs[i - 1]['timestamp'], '%Y-%m-%d
%H:%M:%S')
            t2 = datetime.strptime(logs[i]['timestamp'], '%Y-%m-%d
%H:%M:%S')
            delta_hours = (t2 - t1).total_seconds() / 3600

            energy = ((p1 + p2) / 2) * delta_hours / 1000
            total_kwh += energy
        except Exception as e:
            print(f"[SKIP BAD DATA] index {i}: {e}")
            continue

    bill = calculate_desco_bill(total_kwh)
    return jsonify({
        "total_kwh": round(total_kwh, 3),
        "estimated_bill_bdt": bill
    })


def calculate_desco_bill(kwh):
    slabs = [
        (75, 4.08), (125, 5.99), (100, 6.34),
        (100, 9.94), (200, 10.97), (float('inf'), 11.45)
    ]
    total = 0
    remaining = kwh
    for limit, rate in slabs:
        units = min(remaining, limit)
        total += units * rate
        remaining -= units
        if remaining <= 0:
            break
    meter_rent = 20
```

```
    subtotal = total + meter_rent
    vat = subtotal * 0.05
    return round(subtotal + vat, 2)

@app.route('/api/monthly_kwh')
def monthly_kwh():
    logs = get_energy_logs()
    monthly_data = {}

    for i in range(1, len(logs)):
        try:
            t1 = datetime.strptime(logs[i - 1]['timestamp'], '%Y-%m-%d
%H:%M:%S')
            t2 = datetime.strptime(logs[i]['timestamp'], '%Y-%m-%d
%H:%M:%S')
            p1 = logs[i - 1]['power']
            p2 = logs[i]['power']
            delta_hours = (t2 - t1).total_seconds() / 3600

            energy = ((p1 + p2) / 2) * delta_hours / 1000
            month_key = t1.strftime('%Y-%m')
            monthly_data[month_key] = monthly_data.get(month_key, 0) +
energy
        except Exception as e:
            print(f"Monthly KWh skip: {e}")
            continue

    return jsonify({k: round(v, 3) for k, v in
sorted(monthly_data.items(), reverse=True)})

@app.route('/api/daily_kwh/<month>')
def daily_kwh(month):
    logs = get_energy_logs()
    daily_data = {}

    for i in range(1, len(logs)):
        try:
            t1 = datetime.strptime(logs[i - 1]['timestamp'], '%Y-%m-%d
%H:%M:%S')
```

```
            t2 = datetime.strptime(logs[i]['timestamp'], '%Y-%m-%d
%H:%M:%S')
            if not t1.strftime('%Y-%m') == month:
                continue

            p1 = logs[i - 1]['power']
            p2 = logs[i]['power']
            delta_hours = (t2 - t1).total_seconds() / 3600

            energy = ((p1 + p2) / 2) * delta_hours / 1000
            day_key = t1.strftime('%Y-%m-%d')
            daily_data[day_key] = daily_data.get(day_key, 0) + energy
        except Exception as e:
            print(f"Daily KWh skip: {e}")
            continue

    return jsonify({k: round(v, 3) for k, v in
sorted(daily_data.items())})




# Register cleanup
atexit.register(shutdown)

if __name__ == '__main__':
    app.run(debug=True)
```

config.py :

```
# Device configuration
DEVICE_CONFIG = {
    'DEVICE_ID': "bf493814f4d1067dcbqvx5",
    'DEVICE_IP': "192.168.0.111",
    'LOCAL_KEY': "2V2W+q}UA;pF~'Cb",
    'VERSION': 3.5
}

# Data file paths
DATA_FILES = {
```

```
    'energy_logs': 'data/energy_logs.json',
    'device_status': 'data/device_status.json'
}
```

Tuya_control.py :

```
import tinytuya
import time
from threading import Thread
from config import DEVICE_CONFIG

class TuyaDevice:
    def __init__(self):
        self.device = tinytuya.OutletDevice(
            DEVICE_CONFIG['DEVICE_ID'],
            DEVICE_CONFIG['DEVICE_IP'],
            DEVICE_CONFIG['LOCAL_KEY']
        )
        self.device.set_version(DEVICE_CONFIG['VERSION'])
        self._keep_alive_active = True
        self._start_keep_alive()
        self.last_status = {
            'power': 0,
            'voltage': 0,
            'current': 0,
            'connected': False
        }

    def _start_keep_alive(self):
        """Background thread to ping device every 25 seconds"""
        def keep_alive():
            while self._keep_alive_active:
                try:
                    self.device.status()  # Ping device
                    time.sleep(25)  # Slightly less than 30s sleep
threshold
                except Exception as e:
                    print(f"Keep-alive error: {e}")
                    time.sleep(10)

        Thread(target=keep_alive, daemon=True).start()
```

```
    def get_status(self):
        try:
            status = self.device.status()
            dps = status.get('dps', {})
            self.last_status = {
                'power': dps.get('19', 0) / 10,
                'voltage': dps.get('20', 0) / 10,
                'current': dps.get('18', 0) / 1000,
                'connected': True
            }
            return self.last_status
        except Exception as e:
            print(f"Error in get_status: {e}")
            return {'connected': False}

    def turn_on(self):
        result = self._send_command(True)
        return {'status': 'success' if result else 'error',
                'message': 'Turned ON' if result else 'Failed to turn ON'}

    def turn_off(self):
        result = self._send_command(False)
        return {'status': 'success' if result else 'error',
                'message': 'Turned OFF' if result else 'Failed to turn
OFF'}

    def _send_command(self, state):
        try:
            self.device.set_status(state)
            return True
        except Exception as e:
            print(f"Command error: {e}")
            return False

    def stop_keep_alive(self):
        self._keep_alive_active = False
```