Republic of the Philippines
UNIVERSITY OF SCIENCE AND TECHNOLOGY OF SOUTHERN PHILIPPINES
Claro M Recto Avenue, Lapasan, Cagayan de Oro City

Culminating Activity: Develop and Deploy a University-Wide Electricity Consumption Monitoring Dashboard with IoT Integration

Dionela, Joeira Lou S.

Bual, Ivy Rose C.

Ganohay, Maria Desiree A.

Soldevillo, Antonio C.

Mr. Al-Monte Vince M. Calo

AppDev Instructor

May 21, 2025

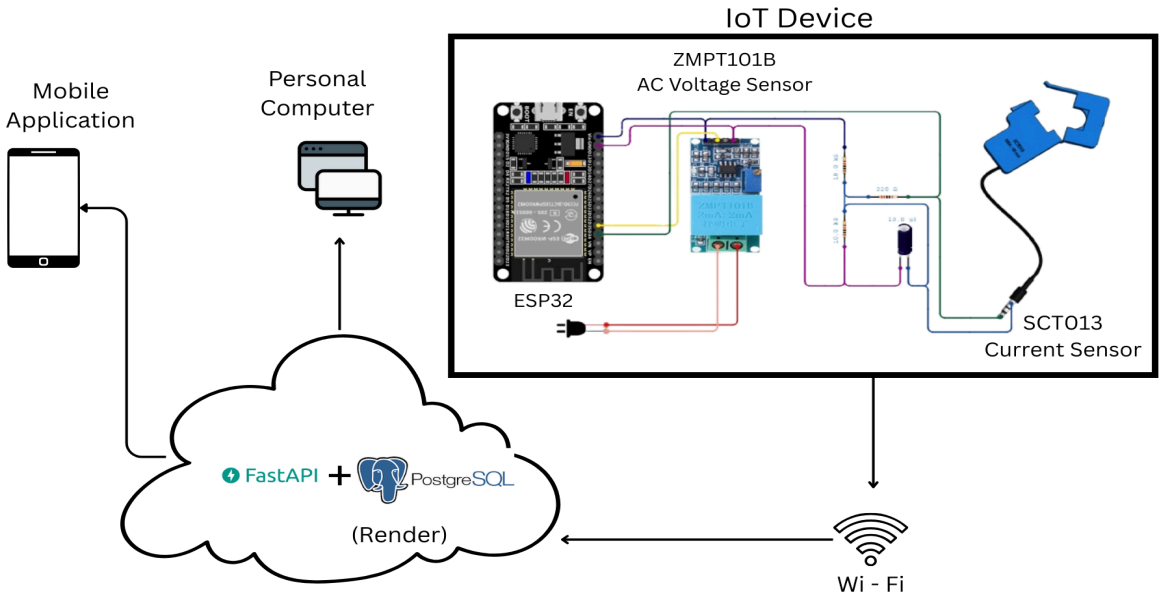
Project Overview

The University of Science and Technology of the Philippines (USTP) incurs significant monthly electricity expenses without a proper monitoring system in place. This project addresses that issue by developing a real-time electricity consumption monitoring system using IoT, FastAPI, and PostgreSQL, along with a web and mobile application dashboard for visualization.

In this prototype, the system monitors the electricity consumption of a single electrical appliance to demonstrate the core functionality. An ESP32 microcontroller reads voltage and current values through sensors such as the ZMPT101B Voltage Sensor and SCT-013 Current Transformer (CT clamp sensor). The collected data is transmitted to a FastAPI-based backend, stored in a PostgreSQL database, and visualized through a React-based web dashboard and a Flutter mobile app.

This proof of concept aims to pave the way for a scalable solution that could eventually be deployed across various buildings within the university for comprehensive energy monitoring.

System Architecture



The system is composed of four primary components:

1. IoT Device

- Includes an ESP32 microcontroller, a ZMPT101B AC Voltage Sensor, and an SCT013 Current Sensor.
- Measures voltage and current from a connected appliance or load.
- Calculates real-time power consumption in kilowatt-hours (kWh).
- Sends real-time energy consumption data every minute via HTTP POST requests over Wi-Fi to a FastAPI-based REST API.

2. REST API (FastAPI)

- Built using FastAPI, hosted on Render.
- Accepts POST requests from the ESP32 to receive energy data.
- Provides GET endpoints for retrieving data to be displayed on the dashboard.
- Handles the following fields: id: Unique entry ID, current: Current in Amperes (A), voltage: Voltage in Volts (V), power: Power in Watts (W), kwh: Energy consumed in kilowatt-hour, and timestamp: Date and time of reading.

3. Database (PostgreSQL)

- Stores energy consumption data with relevant fields such as: id, current (A), voltage (V), power (W), kwh, and timestamp.

4. Web Application Dashboard

- Built to be responsive and accessible on both mobile devices and desktop.
- Displays real-time and historical electricity data in a tabular format.
- Fetches data via GET request from the FastAPI backend.

Technologies and Tools Used

Hardware:

ESP32 (Microcontroller), ZMPT101B (Voltage Sensor), ACS712 (Current Sensor), resistor, and capacitor.

Software:

Frontend: Flutter (Mobile) and Vite React (Web)

Backend: FastAPI and PostgreSQL (Database)

Deployment Platforms: Render (API hosting)

Development Challenges and Solutions

Throughout the development of our application, we encountered several hurdles, especially when it came to using Flutter. Since most of us were new to the framework, getting a grip on how to design the user interface and manage data was a steep learning curve. To tackle this, we relied heavily on online resources—tutorials, YouTube videos, and articles—to deepen our understanding of Flutter and its best practices.

On the IoT side, one of the trickier parts was connecting the sensors and microcontroller. While we were familiar with IoT concepts, we were unsure about the proper connectivity between the components. Ensuring that the power, data, and ground wires were set up correctly was crucial for the system to work properly. We took extra care to avoid mistakes by following wiring diagrams and ensuring the connections were solid. Additionally, we made sure to handle the electrical components carefully, using towels to grip the wires and ensuring our hands were dry. We also double-checked everything before powering up the system.

We also faced issues with sensor accuracy. During the first round of testing, the energy readings were inconsistent. To fix this, we calibrated the sensors using resistive loads, ensuring they were accurate and reliable.

Finally, we ran into deployment compatibility issues when trying to host both the API and the frontend on the same platform. This caused some technical challenges, so we decided to host the API separately on Render while using Vite react for the web frontend and Flutter for the mobile frontend. Both the web and mobile apps connected to the same backend API, which was hosted on Render. By configuring the connections with environment variables, we were able to ensure everything ran smoothly.

Despite these obstacles, we persevered and made significant progress by constantly learning, adapting, and working together as a team.

Future Improvements

- Integrate machine learning for anomaly detection and usage prediction.
- Add SMS/Email alerts for consumption thresholds.
- Deploy multiple ESP32 units for building-level data granularity.
- Add an admin panel to manage IoT devices and users.
- Enable offline data caching on ESP32 with later synchronization.