**Control Layer – High-Level Overview**

**🎯 Purpose:**

To **take action** based on insights from the Monitoring and Optimization layers — turning PowerMind AI from a passive monitor into an **active manager** of electrical systems.

**🧠 Control Layer Goals:**

| **Function** | **Description** |
| --- | --- |
| 🔌 Load Switching | Turn ON/OFF non-critical devices or circuits |
| 🌞 Source Switching | Shift power input (e.g., from grid to solar) when conditions allow |
| 🔋 Battery Charging Control | Trigger charging/discharging during optimized timeframes |
| 🚨 Safety Shutoff | Immediately cut power on overcurrent, overvoltage, or fire conditions |
| 🕰️ Scheduled Automation | Turn systems on/off at specific times, or based on sensor logic |
| 📡 Remote Control & Overrides | Let user trigger actions from app or dashboard |

**🧩 Core Hardware for Control Layer:**

| **Component** | **Purpose** |
| --- | --- |
| 🔁 **Relay Modules** | Act like switches to control AC/DC |
| 🔧 **MOSFETs** | For DC load switching |
| ⚡ **SSR (Solid State Relays)** | Safer switching of AC |
| 📡 **ESP32/WiFi or Pi GPIOs** | Trigger control signals |

**🔐 Safety Considerations:**

* Isolate low-voltage control signals from high-voltage circuits
* Use **optocoupled relays**
* Implement **manual override switches**
* Add **surge protectors** and **fuses**

**🧠 Control Logic (Simple to Smart)**

**Phase 1: Rules-based**

* If power > XW → turn off X
* If solar available → switch source
* If idle for Y mins → shut down

**Phase 2: AI-Driven**

* Predict best time to run appliances
* Learn habits and automate usage patterns
* Smart prioritization during outages

**📶 Control Execution Path:**

css

Copy

Data from Monitoring Layer

↓

Optimization Layer triggers

↓

Control Layer decides actions

↓

Sends GPIO signals to Relays / Switches

**See a relay wiring diagram for controlling a device (like a light or socket)?**

* Raspberry Pi GPIO (e.g. GPIO17)
* Optocoupled 1-channel or 4-channel relay module
* 220V AC or 12V DC load (like a light, fan, or plug)
* Power source isolation (safe switching)

📌 label:

* GND & VCC
* IN1/IN2 logic pin control
* Load connection side (NO/NC/COM)
* Uses RPi.GPIO or gpiozero library
* Basic ON/OFF toggle logic from GPIO pin
* Can be extended with:
  + Scheduled actions
  + Input from Optimization Layer
  + Manual override button or web trigger

| **Condition** | **Action** |
| --- | --- |
| Power > 500W for > 10 sec | Turn OFF relay #2 (AC) |
| Voltage < 10V | Cut relay #1 (DC low safe) |
| 6:00 AM – 8:00 AM | Turn ON relay #3 (lights) |
| Optimization layer recommends shutdown | Turn OFF all loads |

We’ll write this into a control\_rules.json file and link it to the Python script.

**What We're Building Now:**

An intelligent **Relay Control System** that:

* Reads from your control\_rules\_template.json
* Listens to live data from sensors (voltage, current, power)
* Automatically triggers relay GPIO outputs (on/off)
* Logs control actions and allows manual override (future step)

**✅ 1. Script: smart\_control.py**

This will:

* Load rules from control\_rules\_template.json
* Read latest data from power\_log.csv
* Evaluate each rule
* Trigger correct relay (GPIO pin)

**✅ 2. Relays Controlled:**

| **Relay #** | **GPIO Pin** | **Device Example** |
| --- | --- | --- |
| 1 | GPIO17 | Fan or DC Load |
| 2 | GPIO27 | AC Outlet |
| 3 | GPIO22 | Light or Charger |

generate the Python script smart\_control.py that ties together:

* Relay control logic
* Rule-based automation
* Live sensor input

If you need to add more relays or update the logic later, you just update the smart\_control.py script and the control\_rules\_template.json.

**🔧 Here’s how it works:**

**🗂️ 1. control\_rules\_template.json**

You add a new rule like:

json

Copy

{

"id": "rule5",

"condition": {

"time\_range": ["20:00", "22:00"]

},

"action": "turn\_on",

"target": "relay\_4",

"description": "Turn on cooling fan at night"

}

**🧠 2. smart\_control.py**

* The script loads all rules
* Checks the latest sensor values
* If a rule matches → it controls the right **GPIO pin**

For example, if:

json

Copy

"target": "relay\_4"

Then the script knows:

python

Copy

relay\_pin\_map = {

"relay\_1": 17,

"relay\_2": 27,

"relay\_3": 22,

"relay\_4": 5, # You just add this

}

That’s it — scalable and flexible.

📂 Download smart\_control.py

**🧠 What this script does:**

* Reads rules from control\_rules\_template.json
* Monitors latest values from power\_log.csv
* Applies conditions:
  + Power threshold
  + Voltage drop
  + Time ranges
* Activates/deactivates relays via GPIO 17, 27, 22
* Logs actions (print statements for now)