GitHub Portfolio Review and Assistance

Model: Gemini 3.0 pro
https://gemini.google.com/share/912a0c06ba70

https://github.com/JohnRobert-Olano/Olano-COSC111-Portfolio/tree/main

can you look upon this

This looks like the repository for your **COSC111 (CS Elective 3: Internet of Things)** class at Cavite State University.

While I can't browse live GitHub links directly to see the files inside, I can tell this portfolio likely collects the Arduino and Python projects we've discussed recently. Based on your coursework, this probably includes:

- **The Running Light Circuit** (from September)
- **The Fire Sensor Project** (using the thermistor and photoresistor)
- **The Light Intensity Meter** (with the manual/auto modes)
- **The Python-Arduino HMI** (controlling LEDs via serial)

How would you like me to help with this portfolio?

- **Review the README:** If you paste your `README.md` text here, I can check if it clearly explains your projects to your professor.
- **Code Cleanup:** I can help format or comment on specific sketches (like the `main.ino` or Python scripts) if you paste them.
- **Project Descriptions:** I can generate short, professional descriptions for each project folder if you need them for the documentation.

Just let me know what you need!

README_final MD | README_Act7 MD | README_Act6 MD | README_Act5 MD | README_Midterm MD | README_Act4 MD | README_Act3 MD | READM_Act2 MD | README_Act1 MD

create a overall readme for all of this

Here is a professional, comprehensive **Root README** for your GitHub portfolio. I have organized your projects to show your progression from basic electronics to full-stack IoT integration.

You can copy-paste the code block below directly into a file named `README.md` at the very top level of your repository.

Markdown

```
# ⚡ COSC 111: Internet of Things Portfolio

**Student:** John Robert Olaño
**Subject:** CS Elective 3 (Internet of Things)
**Institution:** Cavite State University

---

## 📖 About This Repository
This portfolio documents my coursework and laboratory activities for **COSC 111**. It showcases a progression of skills in embedded systems, starting from basic Arduino circuitry and C++ programming, moving through Serial communication, and culminating in full-stack IoT implementations using Python, FastAPI, and Web Interfaces.

## 🛠️ Tech Stack & Tools

| Category | Tools & Components |
| :--- | :--- |
| **Hardware** | Arduino Uno, ESP8266/32, Sensors (Thermistor, Photoresistor), Actuators (LEDs, Buzzers) |
| **Firmware** | C++ (Arduino IDE), Serial Communication (UART) |
| **Backend** | Python 3, PySerial, FastAPI, Uvicorn |
| **Frontend** | HTML5, JavaScript (Fetch API) |
| **Protocols** | HTTP, REST API, Serial (9600 Baud) |

---

## 📁 Project Index

### 🟢 Part 1: Fundamentals of Embedded Systems
*Basic circuit logic, digital I/O, and analog sensors.*

| Activity | Project Name | Description | Key Concepts |
| :--- | :--- | :--- | :--- |
| **Lab 1** | **[Running Light Sequence](./Lab_Activity_1)** | A sequential LED lighting circuit that demonstrates digital output control and timing loops. | `digitalWrite`, `delay`, Loops |
| **Lab 3** | **[Fire Sensor Simulation](./Lab_Activity_3)** | An automated alarm system that triggers a buzzer/LED when heat and light thresholds are exceeded. | `analogRead`, `if/else`, Thermistor/LDR |
| **Lab 4** | **[Latching Serial Alarm](./Lab_Activity_4)** | A security system logic where an alarm "latches" (stays on) after a trigger until manually reset via Serial command. | State Flags, Boolean Logic, Serial Monitor |

### 🟡 Part 2: Advanced Logic & State Machines
*Complex control flows and user-configurable systems.*

| Activity | Project Name | Description | Key Concepts |
| :--- | :--- | :--- | :--- |
| **Midterm** | **[Light Intensity Meter](./Midterm_Exam)** | A smart metering system with two modes: **Automatic** (fixed thresholds) and **Manual** (user-defined thresholds via CLI). | State Machines, String Parsing, EEPROM (Conceptual) |

### 🔴 Part 3: Python Integration & Full-Stack IoT
*Bridging hardware with software using Python and Web Technologies.*
```

~~~~
| Activity | Project Name | Description | Key Concepts |
| :--- | :--- | :--- | :--- |
| **Lab 5** | **[Python Serial Controller](./Lab_Activity_5)** | A Python script that acts as a remote control for the Arduino, allowing LED manipulation via keyboard commands. | `PySerial`, Python Scripting, Remote Control |
| **Lab 6** | **[Bi-Directional Relay](./Lab_Activity_6)** | A "Software-in-the-Loop" system where Arduino buttons send data to Python, which processes logic and sends commands back to LEDs. | Full-Duplex, Loopback Logic, Latency Management |
| **Lab 7** | **[HTTP Web Control](./Lab_Activity_7)** | A modern IoT architecture using a **Web Browser** to control hardware. Uses **FastAPI** to bridge HTTP requests to Serial commands. | **FastAPI**, REST, JavaScript Fetch, CORS |
| **Final** | **[Physical-to-Cloud API Bridge](./Final_Exam)** | The capstone project: A physical button press on Arduino triggers a remote network API call via a Python gateway. | LAN Networking, HTTP Requests, Middleware |

---

## 🚀 How to Run These Projects

Most projects in this portfolio require a split setup between the Hardware (Arduino) and the Software (Computer).

1.  **Upload Firmware:** Open the `.ino` file in the `Arduino_Firmware` folder of the specific activity and upload it to your board.
2.  **Close Serial Monitor:** If using a Python script, you must close the Arduino IDE Serial Monitor to free up the port.
3.  **Run Python:** Navigate to the `Python_Backend` or `Python_Controller` folder and run:
    ```bash
    pip install -r requirements.txt  # If applicable, or install pyserial/fastapi manually
    python main.py
    ```

---

## 📬 Contact
**John Robert Olaño** *Student, Cavite State University* [GitHub Profile](https://github.com/JohnRobert-Olano)
~~~~

all necessary information regarding the activities to better explain the code base to the viewers

To better explain your codebase to viewers, I have compiled a **Technical Deep Dive** for your portfolio. This document goes beyond the basic summaries and details the specific **logic, algorithms, and hardware integration** used in each activity.

You can use this content as a `TECHNICAL_GUIDE.md` file or append it to your main README to demonstrate your in-depth understanding of the code to recruiters or professors.

📘 Technical Guide & Codebase Architecture

This document provides a detailed breakdown of the logic and control flow for each project in the **COSC 111 Internet of Things** portfolio. It explains how the firmware (Arduino) interacts with the software (Python/Web) and the physical hardware.

🟢 Part 1: Embedded Logic & Sensors

1. Sequential Control (Lab 1)

Project: Running Light Circuit

- **The Problem:** Controlling multiple outputs with precise timing without blocking the processor unnecessarily.
- **Code Logic:**
  - **Pin Initialization:** Pins 8-12 are set to `OUTPUT` mode in the `setup()` block.
  - **Sequential Loop:** The `loop()` uses a standard linear execution flow. It writes a `HIGH` signal to the first LED, pauses execution with `delay(1000)`, and then proceeds to the next.
  - **State Reset:** After the "Turn ON" sequence completes, a mirror "Turn OFF" sequence executes to reset the system state for the next iteration.

2. Multi-Sensor Conditional Logic (Lab 3)

Project: Fire Sensor Simulation

- **The Problem:** Reading analog environmental data and making decisions based on multiple variables simultaneously.
- **Code Logic:**
  - **Threshold Definition:** The system uses `const` integers (`TEMP_THRESHOLD = 50`, `BRIGHT_THRESHOLD = 220`) to define trigger points, making calibration easy without rewriting logic.
  - **Compound Conditionals:** The alarm triggers only when **BOTH** conditions are met. The code uses the logical AND operator (`&&`) inside the `if` statement:

    C++

    ```
    if (temp >= TEMP_THRESHOLD && light >= BRIGHT_THRESHOLD) {
        digitalWrite(ALERT, HIGH);
    }
    ```
  - **Input Types:** Reads from `A0` (Thermistor) and `A2` (Photoresistor) using `analogRead()`, which returns a 10-bit integer (0-1023) representing voltage.

3. State-Latching Logic (Lab 4)

Project: Serial Alarm System

- **The Problem:** Creating an alarm that stays ON even if the sensor value returns to normal (a "Latch"), requiring manual human intervention to reset.
- **Code Logic:**
  - **State Variable:** A boolean flag `isBlinking` tracks the system state, separate from the sensor reading.
  - **The Latch:**
    - When `brightness >= Threshold`, `isBlinking` is set to `true`.
    - The loop checks `if (isBlinking)` to maintain the alarm, ignoring the current sensor data.
  - **String Parsing:** The system listens for serial input to reset the flag. It uses `String.trim()` to remove whitespace and handle user commands cleanly.

🟠 Part 2: Advanced Control Systems

4. Dual-Mode State Machine (Midterm Exam)

Project: Light Intensity Meter

- **The Problem:** Building a system that can switch behaviors (Auto vs. Manual) at runtime without restarting.
- **Code Logic:**
  - **Mode Flag:** A global variable tracks the current mode (`AUTO` or `MANUAL`).
  - **Dynamic Thresholds:** In Manual mode, the code compares sensor data against `lowThreshold` and `highThreshold` variables (which can be updated via Serial) rather than hardcoded constants.
  - **Command Parsing:** The firmware implements a command interpreter that looks for keywords:
    - `startsWith("MODE")` : Switches the logic branch.
    - `startsWith("SET")` : Parses the integer value (`.toInt()`) from the string to update thresholds dynamically.

🔴 Part 3: Python & Full-Stack Integration

5. Uni-Directional Serial Control (Lab 5)

Project: Python LED Controller

- **The Problem:** Controlling hardware from a desktop application.
- **Architecture:** PC (Master) → Arduino (Slave).
- **Python Logic:**
  - Uses `pyserial` to open a COM port.
  - Runs a non-terminating `while True` loop to keep the connection alive.
  - Sanitizes input by converting all keystrokes to lowercase (`.lower()`) before sending, ensuring 'R' and 'r' both trigger the Red LED.
- **Arduino Logic:** Uses `Serial.read()` to grab one character at a time and a `switch/case` block to toggle the corresponding pin.

6. "Software-in-the-Loop" Relay (Lab 6)

Project: Bi-Directional Communication

- **The Problem:** The hardware buttons should not control the LEDs directly; the decision must be made by the Python script (simulating a centralized server).
- **Data Flow:** Button Press → Arduino → Serial ('R') → Python Logic → Serial ('1') → Arduino → LED ON.
- **Key Constraint:** The Arduino is "dumb" regarding the LEDs; it only reports button states. The Python script acts as the "brain," receiving `'R'` , processing it, and sending back `'1'` to confirm the action.

7. HTTP-to-Serial Bridge (Lab 7)

Project: Web-Controlled IoT

- **The Problem:** Controlling hardware via a web browser, which speaks HTTP, not Serial.
- **Architecture:** Browser (JS) ↔ FastAPI (Python) ↔ Arduino.
- **Full Stack Logic:**
  - **Frontend:** `web.html` uses JavaScript's `fetch()` API to send asynchronous HTTP GET requests.
  - **Middleware:** The Python **FastAPI** server receives the HTTP request (e.g., `GET /led/red`). It then writes the corresponding byte (e.g., `b'1'`) to the open Serial connection.
  - **Backend:** Arduino receives the byte and toggles the LED.
  - **CORS:** The Python server implements Cross-Origin Resource Sharing (CORS) to allow the browser to talk to the local server without security blocking.

8. Physical-to-Cloud Trigger (Final Exam)

Project: Serial API Bridge

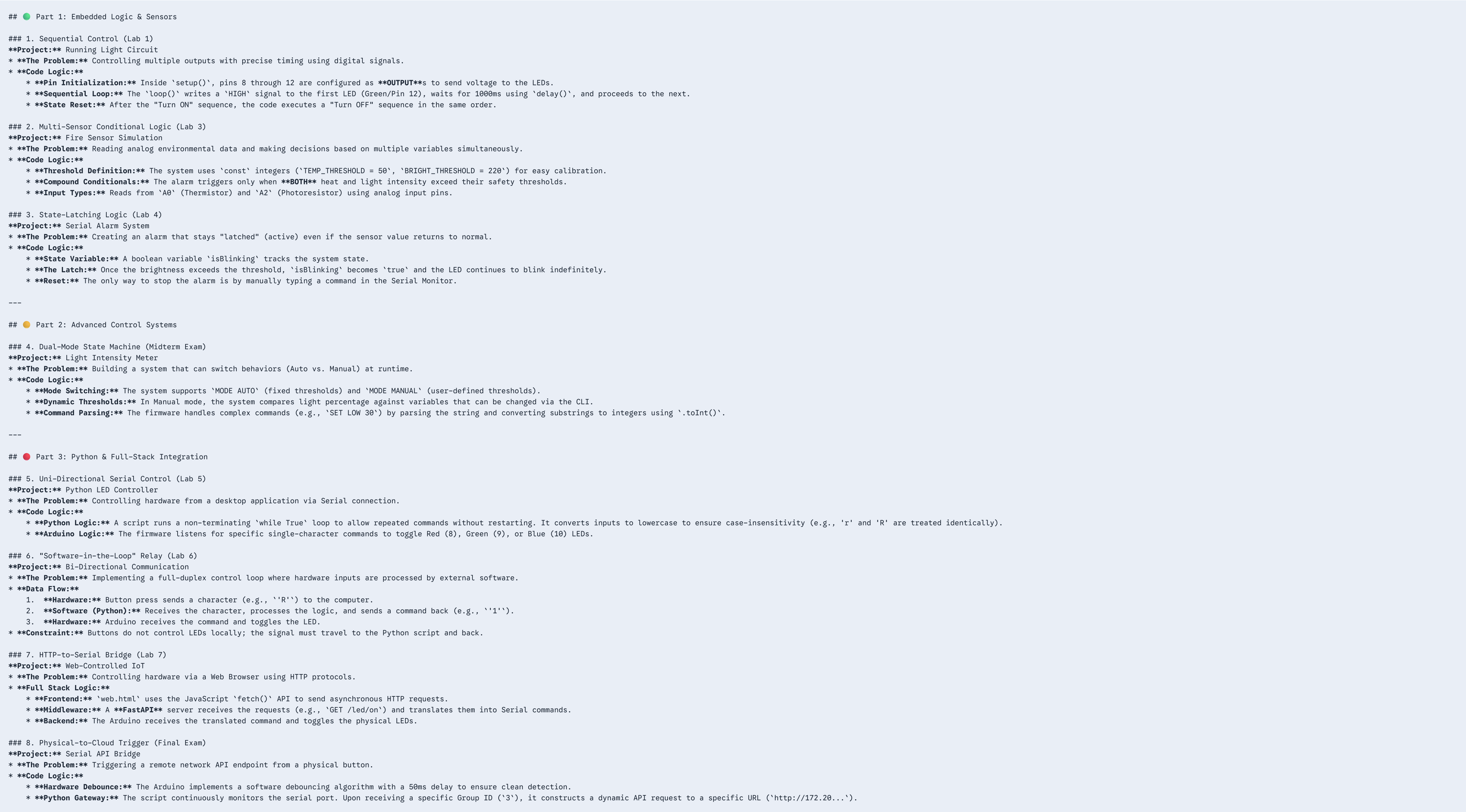
- **The Problem:** Triggering a remote network event from a physical button.
- **Code Logic:**
  - **Hardware Debounce:** The Arduino code implements a 50ms delay to prevent one button press from sending multiple signals ("bouncing").
  - **Input Pull-up:** Uses `INPUT_PULLUP` on Pin 4, simplifying the circuit by removing external resistors.
  - **Python Gateway:** The script monitors the Serial port. Upon receiving the specific Group ID (`3`), it constructs an HTTP request using the `requests` library and hits a remote endpoint (`http://172.20...`), bridging the physical gap to the local network.

put this to a README.md

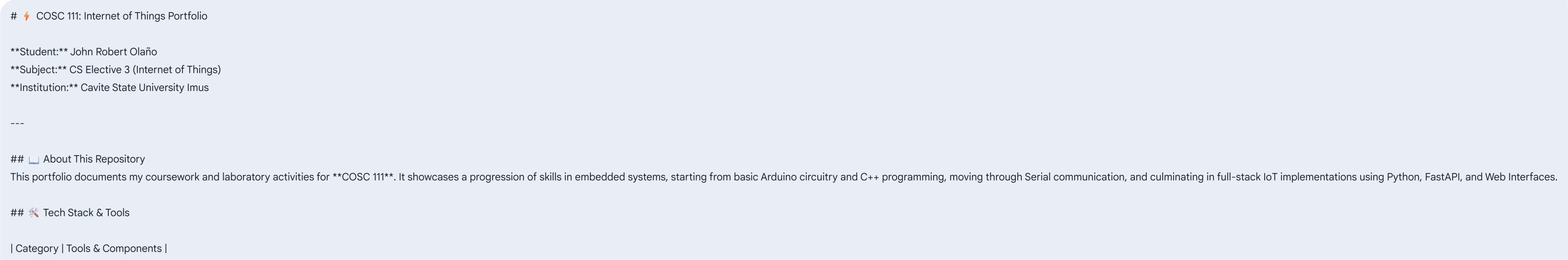
Here is the content formatted as a clean, professional `README.md` file. You can copy the code block below and save it as `TECHNICAL_GUIDE.md` (or replace your main `README.md` if you prefer this technical detail).

Markdown

```
# 📘 Technical Guide & Codebase Architecture

This document provides a detailed breakdown of the logic, algorithms, and hardware integration used in the **COSC 111 Internet of Things** portfolio. It explains how the firmware (Arduino) interacts with the software (Python/Web) and the physical hardware.

---
```



```
## 🟢 Part 1: Embedded Logic & Sensors

### 1. Sequential Control (Lab 1)
**Project:** Running Light Circuit
* **The Problem:** Controlling multiple outputs with precise timing using digital signals.
* **Code Logic:**
    * **Pin Initialization:** Inside `setup()`, pins 8 through 12 are configured as **OUTPUT**s to send voltage to the LEDs.
    * **Sequential Loop:** The `loop()` writes a `HIGH` signal to the first LED (Green/Pin 12), waits for 1000ms using `delay()`, and proceeds to the next.
    * **State Reset:** After the "Turn ON" sequence, the code executes a "Turn OFF" sequence in the same order.

### 2. Multi-Sensor Conditional Logic (Lab 3)
**Project:** Fire Sensor Simulation
* **The Problem:** Reading analog environmental data and making decisions based on multiple variables simultaneously.
* **Code Logic:**
    * **Threshold Definition:** The system uses `const` integers (`TEMP_THRESHOLD = 50`, `BRIGHT_THRESHOLD = 220`) for easy calibration.
    * **Compound Conditionals:** The alarm triggers only when **BOTH** heat and light intensity exceed their safety thresholds.
    * **Input Types:** Reads from `A0` (Thermistor) and `A2` (Photoresistor) using analog input pins.

### 3. State-Latching Logic (Lab 4)
**Project:** Serial Alarm System
* **The Problem:** Creating an alarm that stays "latched" (active) even if the sensor value returns to normal.
* **Code Logic:**
    * **State Variable:** A boolean variable `isBlinking` tracks the system state.
    * **The Latch:** Once the brightness exceeds the threshold, `isBlinking` becomes `true` and the LED continues to blink indefinitely.
    * **Reset:** The only way to stop the alarm is by manually typing a command in the Serial Monitor.

---

## 🟡 Part 2: Advanced Control Systems

### 4. Dual-Mode State Machine (Midterm Exam)
**Project:** Light Intensity Meter
* **The Problem:** Building a system that can switch behaviors (Auto vs. Manual) at runtime.
* **Code Logic:**
    * **Mode Switching:** The system supports `MODE AUTO` (fixed thresholds) and `MODE MANUAL` (user-defined thresholds).
    * **Dynamic Thresholds:** In Manual mode, the system compares light percentage against variables that can be changed via the CLI.
    * **Command Parsing:** The firmware handles complex commands (e.g., `SET LOW 30`) by parsing the string and converting substrings to integers using `.toInt()`.

---

## 🔴 Part 3: Python & Full-Stack Integration

### 5. Uni-Directional Serial Control (Lab 5)
**Project:** Python LED Controller
* **The Problem:** Controlling hardware from a desktop application via Serial connection.
* **Code Logic:**
    * **Python Logic:** A script runs a non-terminating `while True` loop to allow repeated commands without restarting. It converts inputs to lowercase to ensure case-insensitivity (e.g., 'r' and 'R' are treated identically).
    * **Arduino Logic:** The firmware listens for specific single-character commands to toggle Red (8), Green (9), or Blue (10) LEDs.

### 6. "Software-in-the-Loop" Relay (Lab 6)
**Project:** Bi-Directional Communication
* **The Problem:** Implementing a full-duplex control loop where hardware inputs are processed by external software.
* **Data Flow:**
    1.  **Hardware:** Button press sends a character (e.g., `'R'`) to the computer.
    2.  **Software (Python):** Receives the character, processes the logic, and sends a command back (e.g., `'1'`).
    3.  **Hardware:** Arduino receives the command and toggles the LED.
* **Constraint:** Buttons do not control LEDs locally; the signal must travel to the Python script and back.

### 7. HTTP-to-Serial Bridge (Lab 7)
**Project:** Web-Controlled IoT
* **The Problem:** Controlling hardware via a Web Browser using HTTP protocols.
* **Full Stack Logic:**
    * **Frontend:** `web.html` uses the JavaScript `fetch()` API to send asynchronous HTTP requests.
    * **Middleware:** A **FastAPI** server receives the requests (e.g., `GET /led/on`) and translates them into Serial commands.
    * **Backend:** The Arduino receives the translated command and toggles the physical LEDs.

### 8. Physical-to-Cloud Trigger (Final Exam)
**Project:** Serial API Bridge
* **The Problem:** Triggering a remote network API endpoint from a physical button.
* **Code Logic:**
    * **Hardware Debounce:** The Arduino implements a software debouncing algorithm with a 50ms delay to ensure clean detection.
    * **Python Gateway:** The script continuously monitors the serial port. Upon receiving a specific Group ID (`3`), it constructs a dynamic API request to a specific URL (`http://172.20...`).
```



```
# ⚡ COSC 111: Internet of Things Portfolio

**Student:** John Robert Olaño
**Subject:** CS Elective 3 (Internet of Things)
**Institution:** Cavite State University Imus

---

## 📖 About This Repository
This portfolio documents my coursework and laboratory activities for **COSC 111**. It showcases a progression of skills in embedded systems, starting from basic Arduino circuitry and C++ programming, moving through Serial communication, and culminating in full-stack IoT implementations using Python, FastAPI, and Web Interfaces.

## 🛠️ Tech Stack & Tools

| Category | Tools & Components |
```

~~~
| :--- | :--- |
| **Hardware** | Arduino Uno, ESP8266/32, Sensors (Thermistor, Photoresistor), Actuators (LEDs, Buzzers) |
| **Firmware** | C++ (Arduino IDE), Serial Communication (UART) |
| **Backend** | Python 3, PySerial, FastAPI, Uvicorn |
| **Frontend** | HTML5, JavaScript (Fetch API) |
| **Protocols** | HTTP, REST API, Serial (9600 Baud) |

---

## 📁 Project Index

### 🟢 Part 1: Fundamentals of Embedded Systems
*Basic circuit logic, digital I/O, and analog sensors.*

| Activity | Project Name | Description | Key Concepts |
| :--- | :--- | :--- | :--- |
| **Lab 1** | **[Running Light Sequence](./Lab_Activity_1)** | A sequential LED lighting circuit that demonstrates digital output control and timing loops. | `digitalWrite`, `delay`, Loops |
| **Lab 3** | **[Fire Sensor Simulation](./Lab_Activity_3)** | An automated alarm system that triggers a buzzer/LED when heat and light thresholds are exceeded. | `analogRead`, `if/else`, Thermistor/LDR |
| **Lab 4** | **[Latching Serial Alarm](./Lab_Activity_4)** | A security system logic where an alarm "latches" (stays on) after a trigger until manually reset via Serial command. | State Flags, Boolean Logic, Serial Monitor |

### 🟡 Part 2: Advanced Logic & State Machines
*Complex control flows and user-configurable systems.*

| Activity | Project Name | Description | Key Concepts |
| :--- | :--- | :--- | :--- |
| **Midterm** | **[Light Intensity Meter](./Midterm_Exam)** | A smart metering system with two modes: **Automatic** (fixed thresholds) and **Manual** (user-defined thresholds via CLI). | State Machines, String Parsing, EEPROM (Conceptual) |

### 🔴 Part 3: Python Integration & Full-Stack IoT
*Bridging hardware with software using Python and Web Technologies.*

| Activity | Project Name | Description | Key Concepts |
| :--- | :--- | :--- | :--- |
| **Lab 5** | **[Python Serial Controller](./Lab_Activity_5)** | A Python script that acts as a remote control for the Arduino, allowing LED manipulation via keyboard commands. | `PySerial`, Python Scripting, Remote Control |
| **Lab 6** | **[Bi-Directional Relay](./Lab_Activity_6)** | A "Software-in-the-Loop" system where Arduino buttons send data to Python, which processes logic and sends commands back to LEDs. | Full-Duplex, Loopback Logic, Latency Management |
| **Lab 7** | **[HTTP Web Control](./Lab_Activity_7)** | A modern IoT architecture using a **Web Browser** to control hardware. Uses **FastAPI** to bridge HTTP requests to Serial commands. | **FastAPI**, REST, JavaScript Fetch, CORS |
| **Final** | **[Physical-to-Cloud API Bridge](./Final_Exam)** | The capstone project: A physical button press on Arduino triggers a remote network API call via a Python gateway. | LAN Networking, HTTP Requests, Middleware |

---

## 🚀 How to Run These Projects

Most projects in this portfolio require a split setup between the Hardware (Arduino) and the Software (Computer).

1.  **Upload Firmware:** Open the `.ino` file in the `Arduino_Firmware` folder of the specific activity and upload it to your board.
2.  **Close Serial Monitor:** If using a Python script, you must close the Arduino IDE Serial Monitor to free up the port.
3.  **Run Python:** Navigate to the `Python_Backend` or `Python_Controller` folder and run:
    ```bash
    pip install -r requirements.txt  # If applicable, or install pyserial/fastapi manually
    python main.py
    ```

---

# 📘 Technical Guide & Codebase Architecture

This document provides a detailed breakdown of the logic, algorithms, and hardware integration used in the **COSC 111 Internet of Things** portfolio. It explains how the firmware (Arduino) interacts with the software (Python/Web) and the physical hardware.

---

## 🟢 Part 1: Embedded Logic & Sensors

### 1. Sequential Control (Lab 1)
**Project:** Running Light Circuit
* **The Problem:** Controlling multiple outputs with precise timing using digital signals.
* **Code Logic:**
    * **Pin Initialization:** Inside `setup()`, pins 8 through 12 are configured as **OUTPUT**s to send voltage to the LEDs.
    * **Sequential Loop:** The `loop()` writes a `HIGH` signal to the first LED (Green/Pin 12), waits for 1000ms using `delay()`, and proceeds to the next.
    * **State Reset:** After the "Turn ON" sequence, the code executes a "Turn OFF" sequence in the same order.

### 2. Multi-Sensor Conditional Logic (Lab 3)
**Project:** Fire Sensor Simulation
~~~

* **The Problem:** Reading analog environmental data and making decisions based on multiple variables simultaneously.
* **Code Logic:**
   * **Threshold Definition:** The system uses `const` integers (`TEMP_THRESHOLD = 50`, `BRIGHT_THRESHOLD = 220`) for easy calibration.
   * **Compound Conditionals:** The alarm triggers only when **BOTH** heat and light intensity exceed their safety thresholds.
   * **Input Types:** Reads from `A0` (Thermistor) and `A2` (Photoresistor) using analog input pins.

### 3. State-Latching Logic (Lab 4)
Project: Serial Alarm System
* **The Problem:** Creating an alarm that stays "latched" (active) even if the sensor value returns to normal.
* **Code Logic:**
   * **State Variable:** A boolean variable `isBlinking` tracks the system state.
   * **The Latch:** Once the brightness exceeds the threshold, `isBlinking` becomes `true` and the LED continues to blink indefinitely.
   * **Reset:** The only way to stop the alarm is by manually typing a command in the Serial Monitor.

## 🟡 Part 2: Advanced Control Systems

### 4. Dual-Mode State Machine (Midterm Exam)
Project: Light Intensity Meter
* **The Problem:** Building a system that can switch behaviors (Auto vs. Manual) at runtime.
* **Code Logic:**
   * **Mode Switching:** The system supports `MODE AUTO` (fixed thresholds) and `MODE MANUAL` (user-defined thresholds).
   * **Dynamic Thresholds:** In Manual mode, the system compares light percentage against variables that can be changed via the CLI.
   * **Command Parsing:** The firmware handles complex commands (e.g., `SET LOW 30`) by parsing the string and converting substrings to integers using `.toInt()`.

## 🔴 Part 3: Python & Full-Stack Integration

### 5. Uni-Directional Serial Control (Lab 5)
Project: Python LED Controller
* **The Problem:** Controlling hardware from a desktop application via Serial connection.
* **Code Logic:**
   * **Python Logic:** A script runs a non-terminating `while True` loop to allow repeated commands without restarting. It converts inputs to lowercase to ensure case-insensitivity (e.g., 'r' and 'R' are treated identically).
   * **Arduino Logic:** The firmware listens for specific single-character commands to toggle Red (8), Green (9), or Blue (10) LEDs.

### 6. "Software-in-the-Loop" Relay (Lab 6)
Project: Bi-Directional Communication
* **The Problem:** Implementing a full-duplex control loop where hardware inputs are processed by external software.
* **Data Flow:**
   1. **Hardware:** Button press sends a character (e.g., `'R'`) to the computer.
   2. **Software (Python):** Receives the character, processes the logic, and sends a command back (e.g., `'1'`).
   3. **Hardware:** Arduino receives the command and toggles the LED.
* **Constraint:** Buttons do not control LEDs locally; the signal must travel to the Python script and back.

### 7. HTTP-to-Serial Bridge (Lab 7)
Project: Web-Controlled IoT
* **The Problem:** Controlling hardware via a Web Browser using HTTP protocols.
* **Full Stack Logic:**
   * **Frontend:** `web.html` uses the JavaScript `fetch()` API to send asynchronous HTTP requests.
   * **Middleware:** A **FastAPI** server receives the requests (e.g., `GET /led/on`) and translates them into Serial commands.
   * **Backend:** The Arduino receives the translated command and toggles the physical LEDs.

### 8. Physical-to-Cloud Trigger (Final Exam)
Project: Serial API Bridge
* **The Problem:** Triggering a remote network API endpoint from a physical button.
* **Code Logic:**
   * **Hardware Debounce:** The Arduino implements a software debouncing algorithm with a 50ms delay to ensure clean detection.
   * **Python Gateway:** The script continuously monitors the serial port. Upon receiving a specific Group ID (`3`), it constructs a dynamic API request to a specific URL (`http://172.20...`).

## 📬 Contact
John Robert Olaño *Student, Cavite State University Imus* [GitHub Profile](https://github.com/JohnRobert-Olano)

Fix this and put it in a readme.md file

Here is the corrected and formatted content, ready to be pasted directly into a `README.md` file. I have ensured the Markdown syntax (tables, code blocks, and headers) is optimized for GitHub.

Markdown

~~~markdown
# ⚡ COSC 111: Internet of Things Portfolio

**Student:** John Robert Olaño
**Subject:** CS Elective 3 (Internet of Things)
**Institution:** Cavite State University Imus

---

## 📖 About This Repository
This portfolio documents my coursework and laboratory activities for **COSC 111**. It showcases a progression of skills in embedded systems, starting from basic Arduino circuitry and C++ programming, moving through Serial communication, and culminating in full-stack IoT implementations using Python, FastAPI, and Web Interfaces.

## 🛠️ Tech Stack & Tools

| Category | Tools & Components |
| :--- | :--- |
| **Hardware** | Arduino Uno, ESP8266/32, Sensors (Thermistor, Photoresistor), Actuators (LEDs, Buzzers) |
| **Firmware** | C++ (Arduino IDE), Serial Communication (UART) |
| **Backend** | Python 3, PySerial, FastAPI, Uvicorn |
| **Frontend** | HTML5, JavaScript (Fetch API) |
| **Protocols** | HTTP, REST API, Serial (9600 Baud) |

---

## 📂 Project Index

### 🟢 Part 1: Fundamentals of Embedded Systems
*Basic circuit logic, digital I/O, and analog sensors.*

| Activity | Project Name | Description | Key Concepts |
| :--- | :--- | :--- | :--- |
| **Lab 1** | **[Running Light Sequence](./Lab_Activity_1)** | A sequential LED lighting circuit that demonstrates digital output control and timing loops. | `digitalWrite`, `delay`, Loops |
| **Lab 3** | **[Fire Sensor Simulation](./Lab_Activity_3)** | An automated alarm system that triggers a buzzer/LED when heat and light thresholds are exceeded. | `analogRead`, `if/else`, Thermistor/LDR |
| **Lab 4** | **[Latching Serial Alarm](./Lab_Activity_4)** | A security system logic where an alarm "latches" (stays on) after a trigger until manually reset via Serial command. | State Flags, Boolean Logic, Serial Monitor |

### 🟡 Part 2: Advanced Logic & State Machines
*Complex control flows and user-configurable systems.*

| Activity | Project Name | Description | Key Concepts |
| :--- | :--- | :--- | :--- |
| **Midterm** | **[Light Intensity Meter](./Midterm_Exam)** | A smart metering system with two modes: **Automatic** (fixed thresholds) and **Manual** (user-defined thresholds via CLI). | State Machines, String Parsing, EEPROM (Conceptual) |

### 🔴 Part 3: Python Integration & Full-Stack IoT
*Bridging hardware with software using Python and Web Technologies.*

| Activity | Project Name | Description | Key Concepts |
| :--- | :--- | :--- | :--- |
| **Lab 5** | **[Python Serial Controller](./Lab_Activity_5)** | A Python script that acts as a remote control for the Arduino, allowing LED manipulation via keyboard commands. | `PySerial`, Python Scripting, Remote Control |
| **Lab 6** | **[Bi-Directional Relay](./Lab_Activity_6)** | A "Software-in-the-Loop" system where Arduino buttons send data to Python, which processes logic and sends commands back to LEDs. | Full-Duplex, Loopback Logic, Latency Management |
| **Lab 7** | **[HTTP Web Control](./Lab_Activity_7)** | A modern IoT architecture using a **Web Browser** to control hardware. Uses **FastAPI** to bridge HTTP requests to Serial commands. | **FastAPI**, REST, JavaScript Fetch, CORS |
| **Final** | **[Physical-to-Cloud API Bridge](./Final_Exam)** | The capstone project: A physical button press on Arduino triggers a remote network API call via a Python gateway. | LAN Networking, HTTP Requests, Middleware |

---

## 🚀 How to Run These Projects

Most projects in this portfolio require a split setup between the Hardware (Arduino) and the Software (Computer).

1. **Upload Firmware:** Open the `.ino` file in the `Arduino_Firmware` folder of the specific activity and upload it to your board.
2. **Close Serial Monitor:** If using a Python script, you must close the Arduino IDE Serial Monitor to free up the port.
3. **Run Python:** Navigate to the `Python_Backend` or `Python_Controller` folder and run:
    ```bash
    pip install -r requirements.txt  # If applicable, or install pyserial/fastapi manually
    python main.py
    ```

---

# 📘 Technical Guide & Codebase Architecture

This section provides a detailed breakdown of the logic, algorithms, and hardware integration used throughout the portfolio. It explains how the firmware (Arduino) interacts with the software (Python/Web) and the physical hardware.

## 🟢 Part 1: Embedded Logic & Sensors

### 1. Sequential Control (Lab 1)
**Project:** Running Light Circuit
* **The Problem:** Controlling multiple outputs with precise timing using digital signals.
* **Code Logic:**
    * **Pin Initialization:** Inside `setup()`, pins 8 through 12 are configured as **OUTPUT**s to send voltage to the LEDs.
    * **Sequential Loop:** The `loop()` writes a `HIGH` signal to the first LED (Green/Pin 12), waits for 1000ms using `delay()`, and proceeds to the next.
    * **State Reset:** After the "Turn ON" sequence, the code executes a "Turn OFF" sequence in the same order.

### 2. Multi-Sensor Conditional Logic (Lab 3)
**Project:** Fire Sensor Simulation
* **The Problem:** Reading analog environmental data and making decisions based on multiple variables simultaneously.
* **Code Logic:**
    * **Threshold Definition:** The system uses `const` integers (`TEMP_THRESHOLD = 50`, `BRIGHT_THRESHOLD = 220`) for easy calibration.
    * **Compound Conditionals:** The alarm triggers only when **BOTH** heat and light intensity exceed their safety thresholds.
    * **Input Types:** Reads from `A0` (Thermistor) and `A2` (Photoresistor) using analog input pins.

### 3. State-Latching Logic (Lab 4)
**Project:** Serial Alarm System
* **The Problem:** Creating an alarm that stays "latched" (active) even if the sensor value returns to normal.
~~~

```
* **Code Logic:**
    * **State Variable:** A boolean variable `isBlinking` tracks the system state.
    * **The Latch:** Once the brightness exceeds the threshold, `isBlinking` becomes `true` and the LED continues to blink indefinitely.
    * **Reset:** The only way to stop the alarm is by manually typing a command in the Serial Monitor.

---

## 🟡 Part 2: Advanced Control Systems

### 4. Dual-Mode State Machine (Midterm Exam)
**Project:** Light Intensity Meter
* **The Problem:** Building a system that can switch behaviors (Auto vs. Manual) at runtime.
* **Code Logic:**
    * **Mode Switching:** The system supports `MODE AUTO` (fixed thresholds) and `MODE MANUAL` (user-defined thresholds).
    * **Dynamic Thresholds:** In Manual mode, the system compares light percentage against variables that can be changed via the CLI.
    * **Command Parsing:** The firmware handles complex commands (e.g., `SET LOW 30`) by parsing the string and converting substrings to integers using `.toInt()`.

---

## 🔴 Part 3: Python & Full-Stack Integration

### 5. Uni-Directional Serial Control (Lab 5)
**Project:** Python LED Controller
* **The Problem:** Controlling hardware from a desktop application via Serial connection.
* **Code Logic:**
    * **Python Logic:** A script runs a non-terminating `while True` loop to allow repeated commands without restarting. It converts inputs to lowercase to ensure case-insensitivity (e.g., 'r' and 'R' are treated identically).
    * **Arduino Logic:** The firmware listens for specific single-character commands to toggle Red (8), Green (9), or Blue (10) LEDs.

### 6. "Software-in-the-Loop" Relay (Lab 6)
**Project:** Bi-Directional Communication
* **The Problem:** Implementing a full-duplex control loop where hardware inputs are processed by external software.
* **Data Flow:**
    1. **Hardware:** Button press sends a character (e.g., `'R'`) to the computer.
    2. **Software (Python):** Receives the character, processes the logic, and sends a command back (e.g., `'1'`).
    3. **Hardware:** Arduino receives the command and toggles the LED.
* **Constraint:** Buttons do not control LEDs locally; the signal must travel to the Python script and back.

### 7. HTTP-to-Serial Bridge (Lab 7)
**Project:** Web-Controlled IoT
* **The Problem:** Controlling hardware via a Web Browser using HTTP protocols.
* **Full Stack Logic:**
    * **Frontend:** `web.html` uses the JavaScript `fetch()` API to send asynchronous HTTP requests.
    * **Middleware:** A **FastAPI** server receives the requests (e.g., `GET /led/on`) and translates them into Serial commands.
    * **Backend:** The Arduino receives the translated command and toggles the physical LEDs.

### 8. Physical-to-Cloud Trigger (Final Exam)
**Project:** Serial API Bridge
* **The Problem:** Triggering a remote network API endpoint from a physical button.
* **Code Logic:**
    * **Hardware Debounce:** The Arduino implements a software debouncing algorithm with a 50ms delay to ensure clean detection.
    * **Python Gateway:** The script continuously monitors the serial port. Upon receiving a specific Group ID (`3`), it constructs a dynamic API request to a specific URL.

---

## 📬 Contact
**John Robert Olaño** *Student, Cavite State University Imus* [GitHub Profile](https://github.com/JohnRobert-Olano)
```