Group 6 - Week 2: Web Application Foundation and Mathematical Implementation

- Instead of building a custom web app, we used the Blynk IoT platform, which provides a mobile-based interface for controlling and monitoring IoT devices. While not a traditional web application, Blynk offers a robust app-like experience with cloud connectivity.

- We developed a program that enables the Blynk app to communicate with our ESP32-based hardware over the internet. Using virtual pins, the app sends commands and receives sensor data such as distance readings and activation states.

- The interface was kept minimalistic yet functional. We used classic buttons and number input widgets for intuitive control. While the current design is simple, it leaves room for visual enhancements if needed later.

- Basic status indicators such as LED states and buzzer activation can be inferred through the Blynk interface. Although we haven't integrated graphical data charts yet, we can include sensor data visualization (e.g., distance readings over time) using Blynk's gauge or chart widgets in the next development phase.

- The Blynk app uses four virtual buttons to manage the medication reminder system. Each button corresponds to a specific medication time slot and is paired with a number input widget for customizing the duration. When a button is pressed, it triggers the following actions:

- Activates the assigned LED as a visual reminder
- Sounds the buzzer to alert the user
- Moves the servo motor to open the medicine compartment

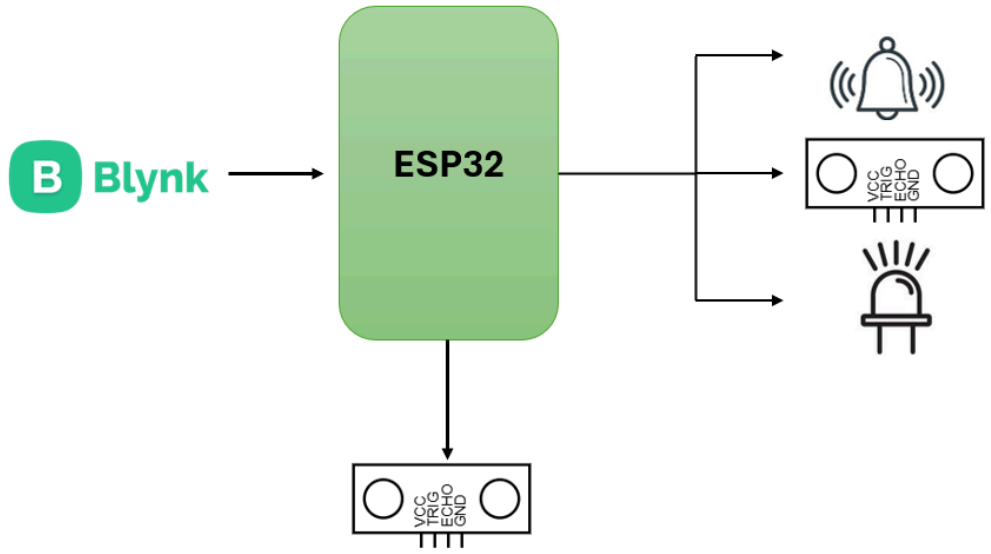
Fourier Series (Frequency Analysis)
We are not implementing Fourier analysis in this project since the system does not involve high-frequency signals or audio-based processing.

Laplace Transform (Control Systems)
This feature is not actively implemented, but our current design mimics basic control system behavior: timed LED activation, servo movement, and event-driven feedback. If needed, future improvements could involve modeling system timing and stability using Laplace-domain tools.

Initial Machine Learning Data Preparation
We have not integrated machine learning at this stage. However, future iterations may explore ML-based personalization of medication schedules, anomaly detection in user interaction, or sensor-based pattern recognition.

Integration Progress

|



Communication Protocols Between Subsystems:
1. Blynk IoT Platform & ESP32:

- Protocol: Wi-Fi / TCP-IP via Blynk Cloud
- Function: Enables remote control and scheduling via virtual pins (V1–V8). Commands are sent from the app to the ESP32 to trigger LED, servo, and buzzer actions.

2. Ultrasonic Sensor & ESP32:

- Protocol: GPIO (Digital Signal)
- Function: Measures object distance. If a user's hand is detected (e.g., <10 cm), it triggers the servo to open the medicine compartment.

3. Servo Motor & ESP32:

- Protocol: PWM (Pulse Width Modulation)

- Function: Moves the box lid to 90° to open it and back to 0° to close it after the medication duration ends.

4. Serial Communication (UART):

- Protocol: USB-UART
- Function: Used for uploading code and debugging via Arduino IDE Serial Monitor.

Data Flow Optimization
We designed our system to be event-driven, ensuring efficient data handling.

- ESP32 listens for user inputs only when necessary, reducing unnecessary polling.
- Sensor readings (like the ultrasonic sensor) are sampled every 500ms, balancing responsiveness and performance.
- Servo and buzzer are activated only when an action is required (e.g., medication time).
- Debug messages are throttled to 1-second intervals to avoid flooding the serial monitor.

This approach minimizes resource usage and keeps the system responsive and efficient.

Benchmarking
We've tested the system and confirmed that the major functions—LED control, servo operation, ultrasonic sensing, and buzzer activation—work as expected. However, fine-tuning and stability testing are still ongoing. We aim to enhance reliability and responsiveness further.

Power Optimization Ideas
Here are some suggestions for reducing energy consumption:

- Use deep sleep mode: ESP32 can be put into deep sleep between scheduled tasks.
- Optimize sensor polling: Reduce sensor check frequency during idle states.
- Turn off unused peripherals: Disable pins when components (e.g., buzzer or servo) are inactive.
- Use energy-efficient components: Consider low-power versions of sensors and actuators in future iterations.
- Battery-powered mode: Add a rechargeable battery and use power-efficient voltage regulators.

Documentation