ARDUINO-BASED SMART TRAFFIC LIGHT SYSTEM

IoT training

TEAM 11

PROJECT OVERVIEW

This system is designed to simulate a smart traffic management solution, demonstrating the integration of various sensors, actuators, and communication protocols on an ESP32 microcontroller.

KEY FEATURES OF IOT TRAFFIC LIGHT SYSTEM

TRAFFIC LIGHT CONTROL

manages red/yellow/green traffic lights with predefined durations.

GATE CONTROL

servo motor acts as a gate, opening when the traffic light is green and closing for red and yellow.

MOTION DETECTION

PIR sensor detects car movement. If a car is detected during a red or yellow light, it triggers a violation, activating a buzzer and keeping the gate closed.

NIGHT MODE

LDR (Light Dependent Resistor) detects ambient light levels. In low light conditions (night), a blue LED is activated, indicating night mode.

KEY FEATURES OF IOT TRAFFIC LIGHT SYSTEM

LCD DISPLAY

I2C LCD displays the current traffic light state, countdown timer, and messages related to car detection or manual overrides.

MQTT INTEGRATION

The system connects to an MQTT broker (HiveMQ) to enable remote control and monitoring. Users can manually open/close the gate, switch between day/night modes, and control traffic light states via MQTT messages.

WIFI CONNECTIVITY

The ESP32 connects to a specified WiFi network to facilitate MQTT communication.

TECH STACK

1

ESP32 Microcontroller

The main processing unit for controlling all components and handling communication.

2

PIR Motion Sensor

Detects infrared light emitted by objects, used here to sense car presence.

3

Servo Motor

Controls the gate, opening and closing it based on traffic light states or manual commands.

4

Red, Yellow, Green LEDs

Represent the traffic lights, indicating STOP, READY, and GO states respectively.

5

Buzzer

Provides an audible alert, primarily for violation detection.

TECH STACK

6 LDR (Light Dependent Resistor)

Measures ambient light intensity to determine day or night mode.

7 Blue LED

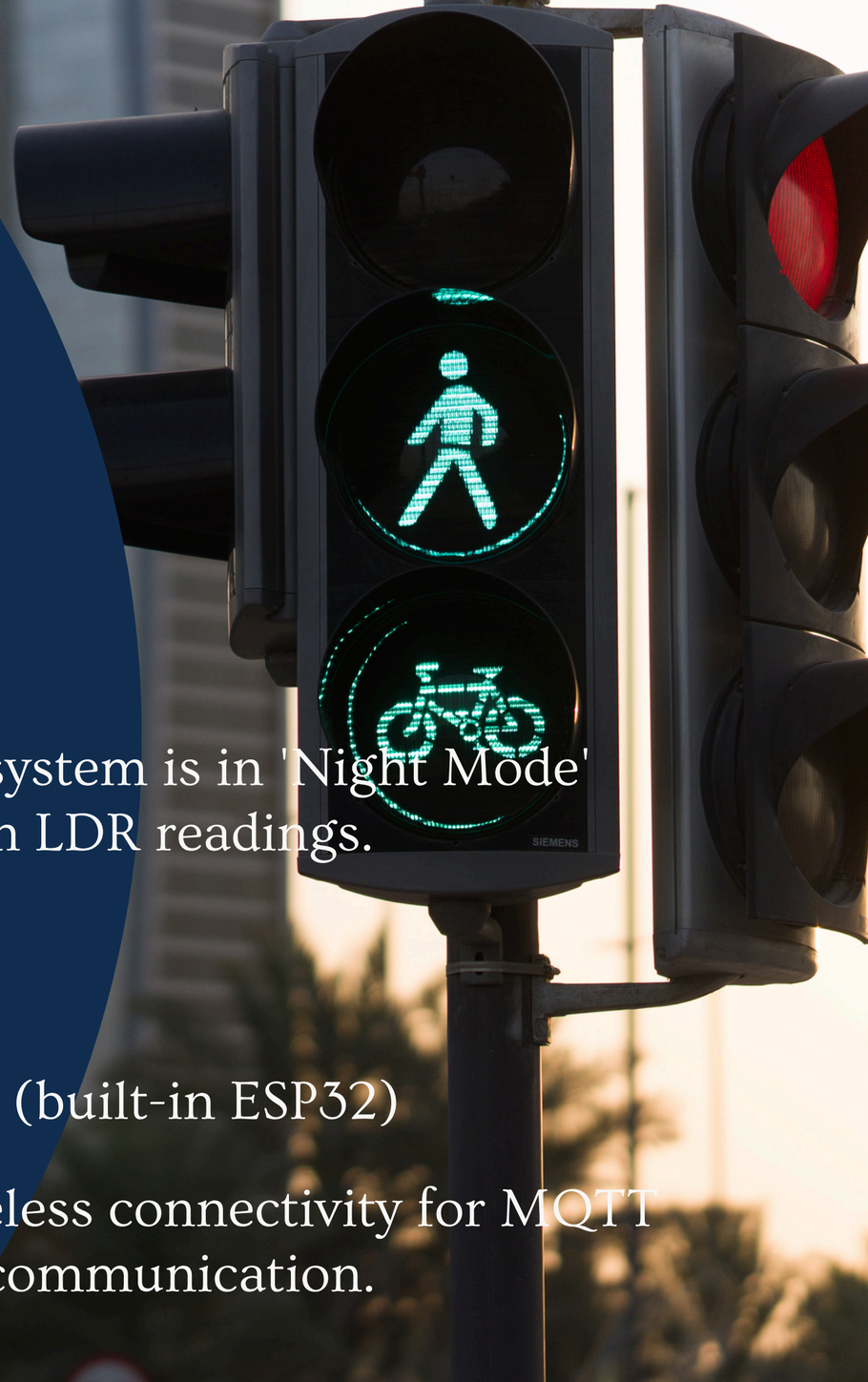
Indicates when system is in 'Night Mode' based on LDR readings.

8 LiquidCrystal_I2C LCD (16x2)

Displays system status, countdowns, and messages to the user.

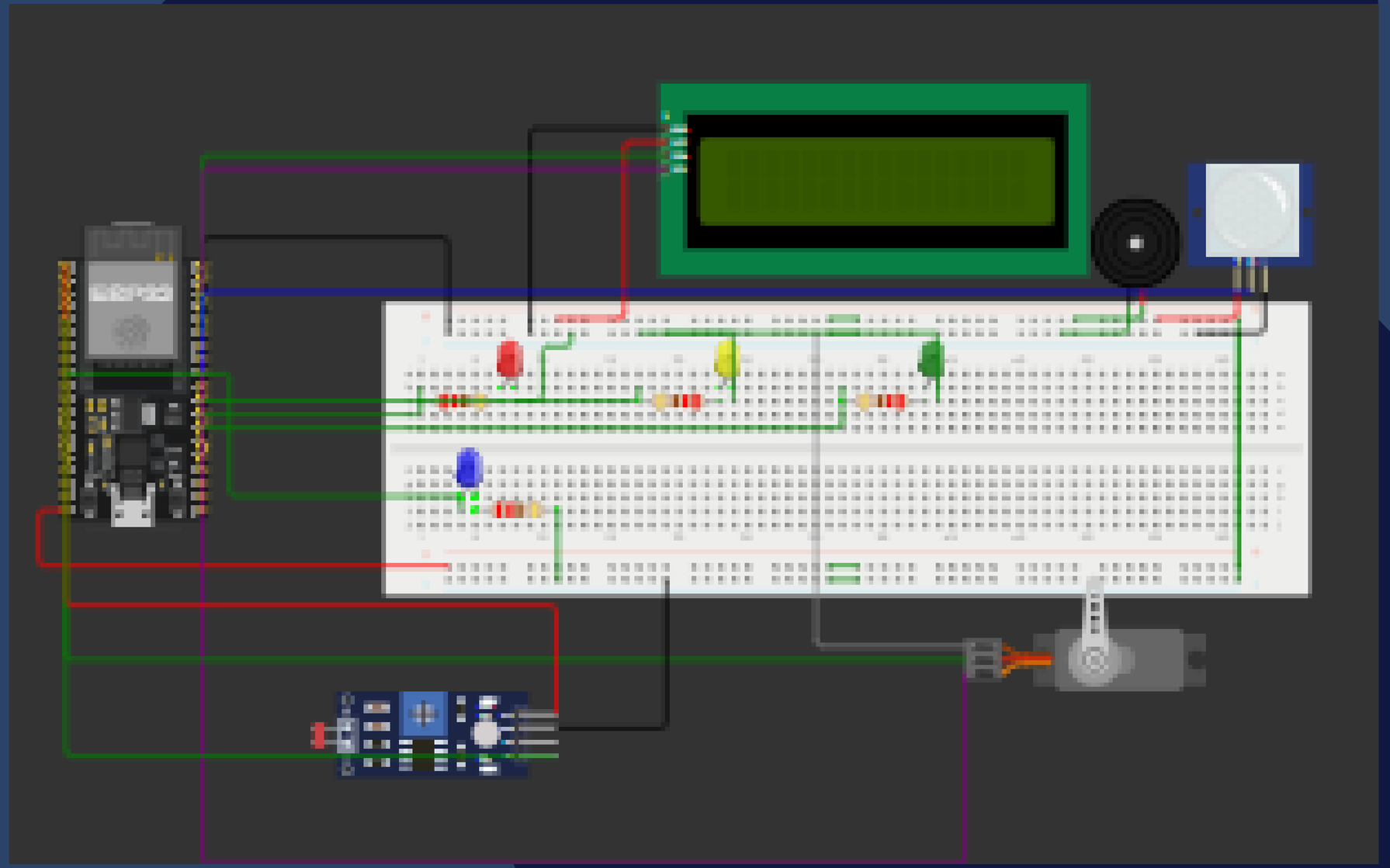
8 WiFi Module (built-in ESP32)

Enables wireless connectivity for MQTT communication.

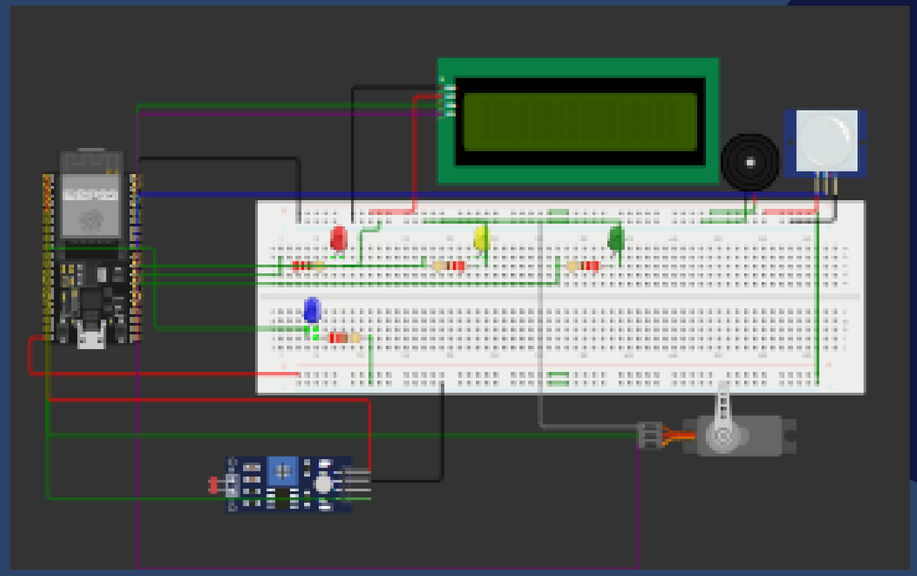


WOKWI CIRCUIT

All hardware components are connected to specific GPIO pins on the ESP32.



PIN DEFINITIONS



- PIR_PIN (GPIO 18): Input for the PIR motion sensor.
- SERVO_PIN (GPIO 19): Output for the servo motor controlling the gate.
- RED_LED (GPIO 16): Output for the red traffic light LED.
- YELLOW_LED (GPIO 17): Output for the yellow traffic light LED.
- GREEN_LED (GPIO 4): Output for the green traffic light LED.
- BUZZER_PIN (GPIO 23): Output for the buzzer.
- LDR_PIN (GPIO 34): Analog input for the LDR.
- BLUE_LED (GPIO 25): Output for the blue LED, indicating night mode.

LOGIC FLOW

The Arduino code for the traffic light system operates within a continuous loop, managing various states and functions after an initial setup.
The logic flow can be broken down into two main parts: the initialization in the setup() function and the main operational loop in the loop() function.

1

Initialization (setup Function)

When the ESP32 microcontroller starts, the setup() function runs once to prepare all components and configurations.

2

Main Program Logic (loop Function)

After the setup is complete, the loop() function runs continuously, managing the system's core functions.

INITIALIZATION (SETUP FUNCTION)

- Serial and WiFi: Initializes serial communication for debugging and connects to the specified WiFi network.
- MQTT: Configures the MQTT client, sets the callback function for incoming messages, and attempts an initial connection to the broker.
- Hardware Setup: Configures the pin modes for all LEDs, the buzzer, PIR sensor, and LDR as either INPUT or OUTPUT. It also allocates a timer for the servo motor and attaches it to its pin. All traffic light LEDs are turned off initially.
- LCD: Initializes the I2C LCD, turns on its backlight, clears the display, and shows a starting message.
- System State: A brief LED test sequence confirms the lights are working, and the initial traffic state is set to STOP.

```
119  void setup() {
120    Serial.begin(115200);
121    WiFi.begin(ssid, password);
122    wifiClient.setInsecure(); // Skip certificate validation
123
124    while (WiFi.status() != WL_CONNECTED) {
125      delay(500);
126      Serial.print(".");
127    }
128    Serial.println("\n WiFi connected");
129
130    client.setServer(mqtt_server, mqtt_port);
131    client.setCallback(callback);
132
133    while (!client.connected()) {
134      Serial.print("Connecting to MQTT...");
135      if (client.connect(client_id, mqtt_user, mqtt_pass)) {
136        Serial.println("connected!");
137      } else {
138        Serial.print("failed, rc=");
139        Serial.print(client.state());
140        delay(2000);
141      }
142    }
143    client.subscribe("trafficApp/gate");
144    client.subscribe("trafficApp/light");
145    client.subscribe("trafficApp/traffic");
146
147    // Initialize Servo
148    ESP32PWM::allocateTimer(0);
149    gateServo.setPeriodHertz(50);
150    gateServo.attach(SERVO_PIN);
151
152    // Initialize Pins
153    pinMode(PIR_PIN, INPUT);
154    pinMode(RED_LED, OUTPUT);
```

MAIN PROGRAM LOGIC (LOOP FUNCTION)

- MQTT Management: It continuously processes incoming and outgoing MQTT messages and checks the client is connected. If not, it calls the
- reconnectMQTT() function to re-establish the connection.
- Car Detection & Violations: The checkCarMovement() function is called to read the PIR motion sensor. If a car is detected during a
- STOP or READY state, a violation is triggered, which then handled by the handleViolation() function. During a violation, normal traffic light cycling is paused.
- Manual Override: A manualOverride flag, which is se to true by incoming MQTT messages, pauses the automatic traffic light cycling, allowing for remote control of the gate and traffic lights. The override is reset after a countdown.

```
197  void loop() {
198    client.loop(); // Handle MQTT messages
199    if (!client.connected()) {
200      reconnectMQTT();
201    }
202
203    unsigned long currentMillis = millis();
204
205    checkCarMovement(); // Check PIR sensor
206
207    // Handle violation buzzer and lockout
208    if (violationDetected) {
209      handleViolation();
210      return; // Skip rest of loop during violation
211    }
212
213    // Manual override logic
214    if (manualOverride) {
215      if (countdown == 0) {
216        manualOverride = false;
217        setTrafficState(STOP); // Resume from STOP or last known state
218        previousMillis = currentMillis;
219        countdown = stopDuration / 1000;
220      }
221    } else {
222      // Automatic traffic light cycling
223      switch (currentState) {
224        case STOP:
225          if (currentMillis - previousMillis >= stopDuration) {
226            setTrafficState(READY);
227            previousMillis = currentMillis;
228            countdown = readyDuration / 1000;
229          }
230          break;
```

MAIN PROGRAM LOGIC (LOOP FUNCTION)

- Automatic Cycling: If there is no manual override, a switch statement manages the transitions between the traffic states (STOP, READY, and GO) based on predefined durations and timing using millis(). The
- setTrafficState() function handles changing the LED colors and controlling the gate servo for each state.
- Countdown Display: The updateCountdown() function is called every second to show the remaining time on the LCD.
- Night Mode: The LDR sensor is read to detect ambient light levels. If it's dark, a blue LED is activated, and "Night Mode ON" is displayed on the LCD.

```
197   void loop() {
214     if (manualOverride) {
221     } else {
222       // Automatic traffic light cycling
223       switch (currentState) {
224         case STOP:
225           if (currentMillis - previousMillis >= stopDuration) {
226             setTrafficState(READY);
227             previousMillis = currentMillis;
228             countdown = readyDuration / 1000;
229           }
230           break;
231
232         case READY:
233           if (currentMillis - previousMillis >= readyDuration) {
234             setTrafficState(GO);
235             previousMillis = currentMillis;
236             countdown = goDuration / 1000;
237           }
238           break;
239
240         case GO:
241           if (currentMillis - previousMillis >= goDuration) {
242             setTrafficState(STOP);
243             previousMillis = currentMillis;
244             countdown = stopDuration / 1000;
245           }
246           break;
247       }
248     }
249
250     // Update countdown every second
```

FLUTTER APP

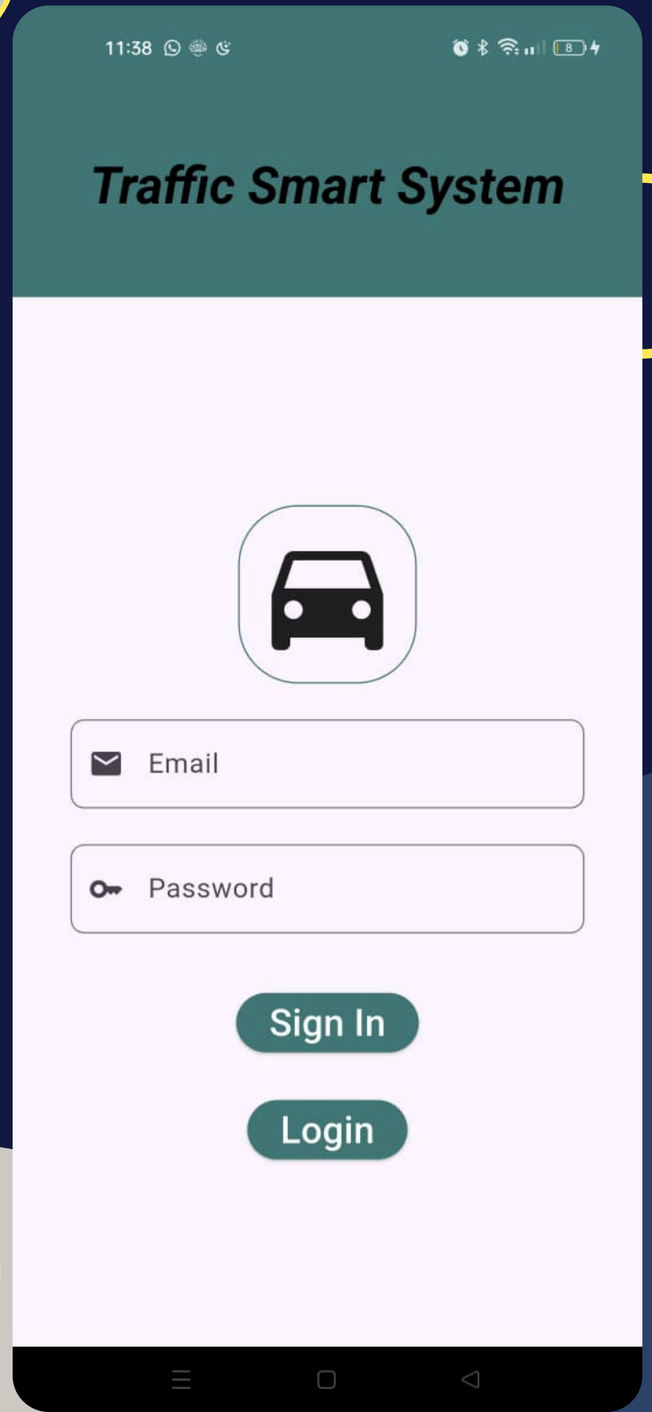
This screen shows app name and a car icon for 5 seconds when the app starts. After that, it automatically takes the user to the login page.

```
ort 'package:flutter/material.dart';
ort 'package:flutter_application_1/TrafficApp/login_page.dart';//import
in page
Define a stateless widget called Interface (splash screen)
ss Interface extends StatelessWidget {
onst Interface ({super.key});
override
idget build(BuildContext context) {
 // Schedule a task to run after a 5-second delay
 // This navigates to the LoginPage and replaces the current screen
 Future.delayed(const Duration(seconds: 5), () {
   Navigator.pushReplacement(
     context,
     MaterialPageRoute(builder: (_) =>  LoginPage()),
   );
 });
 return Scaffold(
   backgroundColor: const Color.fromARGB(255, 66, 119, 119),//set
kground color
   body: Center(// Center the content vertically and horizontally
```

SPLASH PAGE

FLUTTER APP

This is the login page for your app.
It shows two input fields: one for email and one for password.
The user can either create a new account by pressing Sign In, or log in to an existing account by pressing Login.



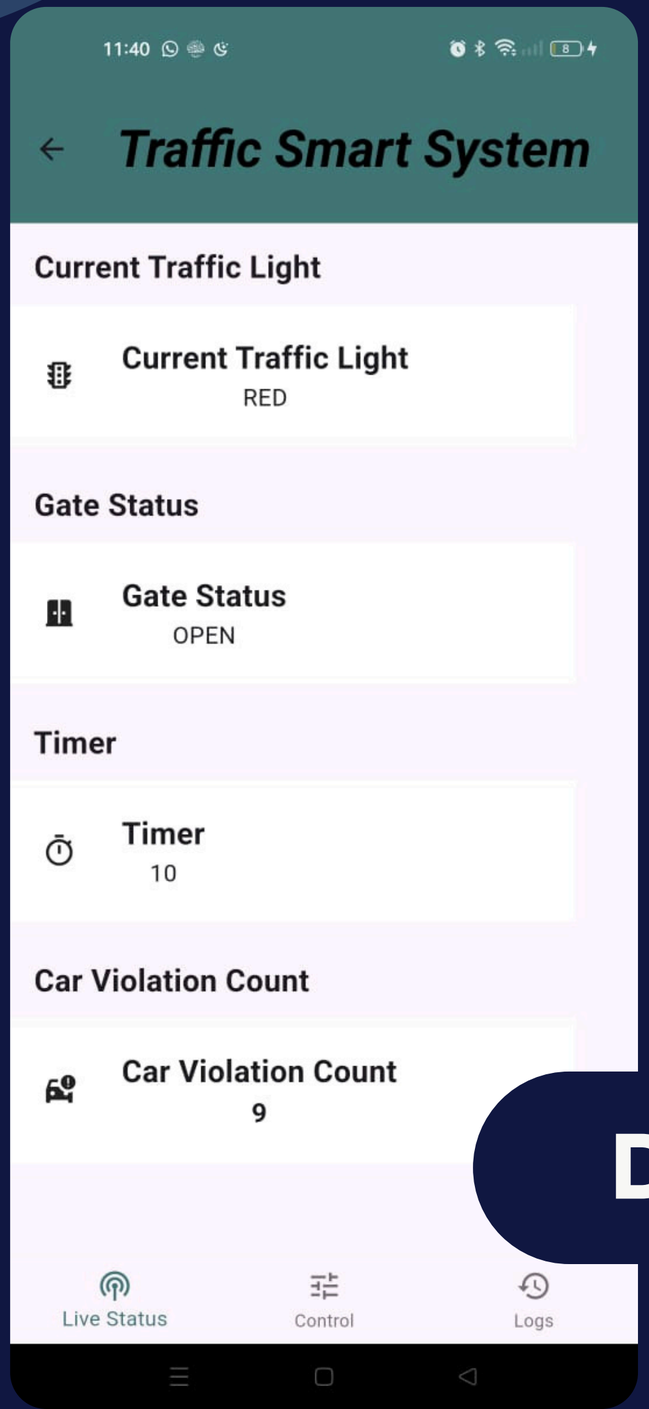
LOGIN PAGE

FLUTTER APP

This screen shows live updates from your traffic system. It displays:

- The current traffic light status
- The gate status
- A countdown timer
- The number of car violations

All of this data is streamed in real time from Supabase tables.



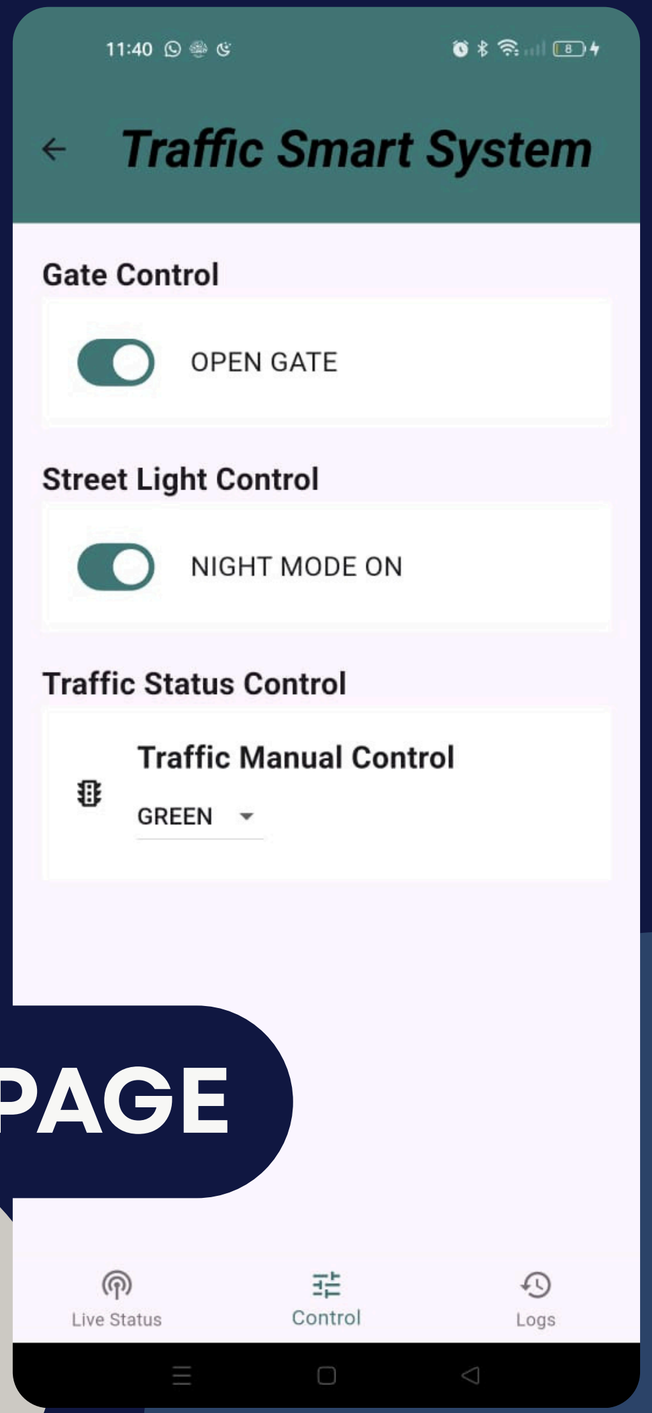
DASHBOARD

FLUTTER APP

This screen lets you manually control parts of your traffic system using MQTT. You can:

- Open or close the gate
- Switch street lights between night and day mode
- Change the traffic light color (red, yellow, green)

Each action sends a message to a specific MQTT topic so that connected devices can respond.



CONTROL PAGE

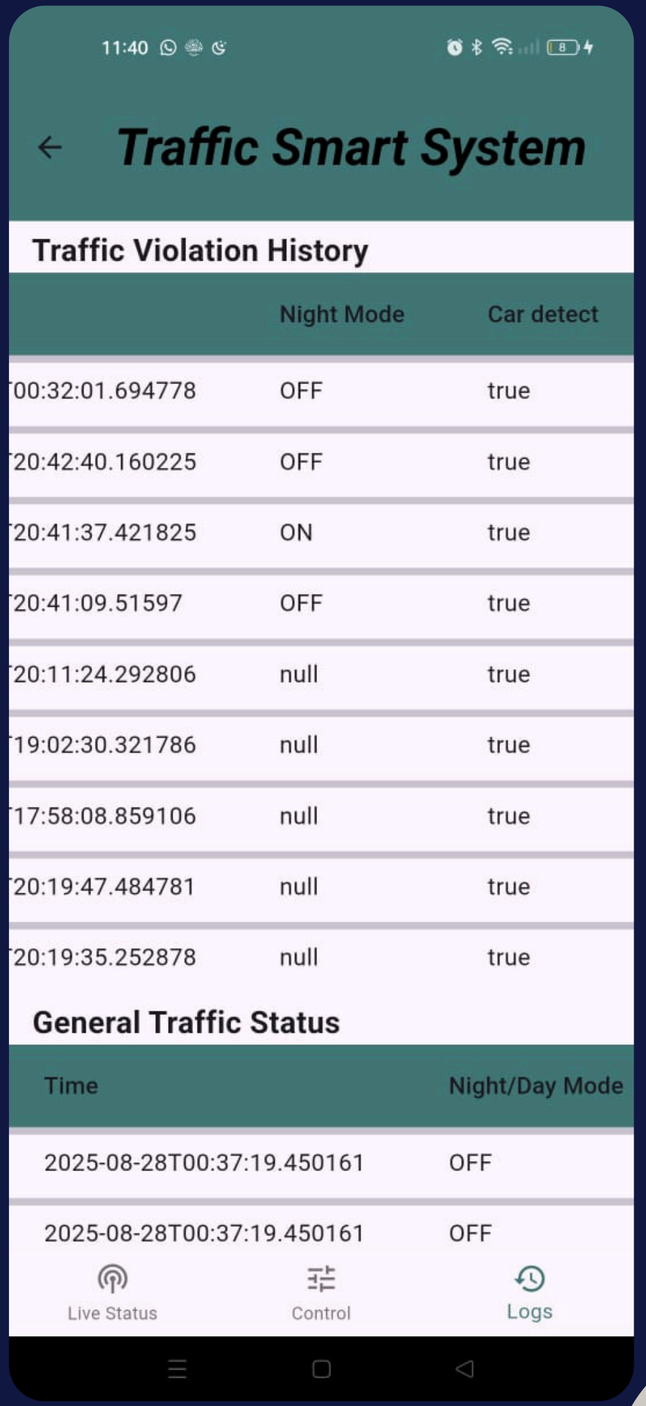
FLUTTER APP

This widget creates a real-time dashboard using Supabase streams. It shows:

- A table of traffic violations with time, night mode, and car detection.
- A table of general traffic system status with light, gate, and traffic light info.

Both tables update automatically when new data is inserted into the database or view.

LOGS PAGE



SUPABASE API

```
//Database info
Supabase db;
String gateTable = "gate_status";
String streetLight_table = "street_lights";
String trafficLights_table = "traffic_lights";
String violations_table = "violations";
```

Credentials project url and anonymous key

```
// Supabase credentials
const char* supabaseUrl ="https://ufrtmaqbbssewzieonnq.supabase.co";
const char* supabaseKey ="eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJzdXBhYmFzZSIsI
```

Supabase functions:

```
//supabase
void sendViolation();
String logTrafficLight();
String logGateStatus();
String logStreetLight();
```

These functions are written in C++ and are designed to send structured data to a Supabase backend. Each function logs a specific type of traffic-related event by constructing a JSON string and sending it via an HTTP request using a db.insert() method.

SUPABASE API

Logs a traffic violation when a car is detected.

```
int sendViolation(String nightMode) {
  String Json = "{\"car_detected\":true";

 if (nightMode != "") {
    Json += ",\"night_mode\":\"" + nightMode + "\"";
  }

  Json += "}";
  Serial.println("Sending JSON: " + Json);
  int responseCode = db.insert("violations", Json, false); // false = no upsert
  Serial.print("Supabase response: ");
  Serial.println(responseCode);
  return responseCode;
}
```

Logs the current status of a traffic light along with its countdown timer.

```
int logTrafficLight(String status, int countdown) {
  String Json = "{\"status\":\"" + status + "\",\"countdown\":" + String(countdown) + "}";
  int responseCode = db.insert("traffic_lights", Json, false); // false = no upsert
  Serial.print("HTTP Response Code: ");
  Serial.println(responseCode);
  return responseCode;

}
```

SUPABASE API

Logs the current status of a gate, such as "OPEN" or "CLOSED".

```
int logGateStatus(String status) {
  String Json = "{\"status\":\"" + status + "\"}";
  int responseCode = db.insert("gate_status", Json, false); // false = no upsert
  Serial.print("HTTP Response Code: ");
  Serial.println(responseCode);
  return responseCode;

}
```

Logs the current status of a street light, such as "ON" or "OFF".

```
int logStreetLight(String status) {
  String Json = "{\"status\":\"" + status + "\"}";
  int responseCode = db.insert("street_lights", Json, false); // false = no upsert
  Serial.print("HTTP Response Code: ");
  Serial.println(responseCode);
  return responseCode;
}
```

LANGCHAIN CHATBOT AGENT

```python
from fastapi import FastAPI, Request
from fastapi.responses import JSONResponse, HTMLResponse
import paho.mqtt.client as mqtt
from langchain.agents import initialize_agent, AgentType, Tool
from langchain_google_genai import ChatGoogleGenerativeAI
import uvicorn
```

Defining functions : on connect for connection and on message to perform messaging between human and bot.

```python
# --- MQTT Setup ---
BROKER = "db214e9cf2184882a22e1639d81e429f.s1.eu.hivemq.cloud"
PORT = 8883
TOPIC_GATE = "trafficApp/gate"
TOPIC_LIGHT = "trafficApp/light"
TOPIC_TRAFFIC = "trafficApp/traffic"

mqtt_client = mqtt.Client()


def on_connect(client, userdata, flags, rc):  1 usage
    print("Connected to MQTT with code", rc)
    client.subscribe("trafficApp/#")  # subscribe to all traffic top


def on_message(client, userdata, msg):  1 usage
    print(f"[MQTT] {msg.topic} => {msg.payload.decode()}")


mqtt_client.on_connect = on_connect
mqtt_client.on_message = on_message
mqtt_client.connect(BROKER, PORT, keepalive: 60)
mqtt_client.loop_start()
```

LANGCHAIN CHATBOT AGENT

Defining functions for :
1-openning the gate
2-closing the gate
3-set traffic light to red or yellow or green
4-then the mode (day or night).

```python
# --- Tool Functions ---
def open_gate(_=None):  1 usage
    mqtt_client.publish(TOPIC_GATE,  payload: "OPEN")
    return "Gate opened"


def close_gate(_=None):  1 usage
    mqtt_client.publish(TOPIC_GATE,  payload: "CLOSE")
    return "Gate closed"


def set_traffic_light(color: str):  1 usage
    color = color.upper()
    if color not in ["RED", "YELLOW", "GREEN"]:
        return "Invalid color. Use RED, YELLOW, or GREEN."
    mqtt_client.publish(TOPIC_TRAFFIC, color)
    return f"Traffic light set to {color}"


def set_mode(mode: str):  1 usage
    mode = mode.upper()
    if mode not in ["DAY", "NIGHT"]:
        return "Invalid mode. Use DAY or NIGHT."
    mqtt_client.publish(TOPIC_LIGHT, mode)
    return f"Mode set to {mode}"
```

LANGCHAIN CHATBOT AGENT

langchain tools : which uses the functions we defined.
Then we clarify system prompt:
In llm we use the Gemini model 2.5 flash and the api key i get from Gemini page.

```
# --- LangChain Tools ---
tools = [
    Tool(name="Open Gate", func=open_gate, description="Opens the gate barrier."),
    Tool(name="Close Gate", func=close_gate, description="Closes the gate barrier."),
    Tool(name="Set Traffic Light", func=set_traffic_light,
         description="Set traffic light to RED, YELLOW, or GREEN."),
    Tool(name="Set Mode", func=set_mode, description="Set mode to DAY or NIGHT."),
]

system_prompt = """
You are a Traffic Assistant Chatbot.
You control a smart traffic light and gate system.

Available tools:
- Open Gate
- Close Gate
- Set Traffic Light (RED, YELLOW, GREEN)
- Set Mode (DAY, NIGHT)

Rules:
1. ALWAYS use the tools — don't just explain.
2. Only valid commands are accepted.
3. If invalid, reply with: "I cannot do that."
"""

llm = ChatGoogleGenerativeAI(
    model="gemini-2.5-flash",
    google_api_key="AIzaSyBRQIH-V_VOmvplsjRtIgCKVUewrMAEruM"
)
```

LANGCHAIN CHATBOT AGENT

Initializing agent and how to deal with errors and returning outputs.
Then the chat term :
Making request and respond to those requests using system prompts and uvicorn running

```python
agent = initialize_agent(
    tools=tools,
    llm=llm,
    agent=AgentType.STRUCTURED_CHAT_ZERO_SHOT_REACT_DESCRIPTION,
    verbose=True,
    handle_parsing_errors=True,
    return_only_outputs=True
)
# --- FastAPI App ---
/
app = FastAPI()

/chat
@app.post("/chat")
async def chat(request: Request):
    data = await request.json()
    user_input = data.get("text", "")
    try:
        reply = agent.invoke({
            "input": user_input,
            "chat_history": [{"role": "system", "content": system_prompt}]
        })
        return JSONResponse({"reply": reply.get("output", "")})
    except Exception as e:
        return JSONResponse( content: {"error": str(e)}, status_code=500)

if __name__ == "__main__":
    uvicorn.run( app: "bot:app", host="0.0.0.0", port=8000, reload=True)
```

FUTURE WORK

- Expanded Sensor Integration: Add more sensors to gather additional environmental data. This could include a humidity sensor to provide weather information or an air quality sensor to monitor pollution levels at the intersection.

- Smart Traffic Management: Implement a machine learning model to analyze traffic flow patterns, allowing the system to dynamically adjust traffic light timings to optimize traffic flow, rather than relying on predefined durations.

- Advanced Violation Handling: Upgrade the violation detection system to include a camera module for recording images or video of red-light runners. This would provide stronger evidence for handling violations.