Live Weather Updates on OLED Display using ESP32 and AccuWeather API

Introduction

In this project, we explore the exciting world of IoT (Internet of Things) by building a system that displays real-time time and weather information on an OLED screen.

The primary goal of this project is to showcase how we can leverage the ESP32's capabilities to connect to the internet, retrieve weather data from the AccuWeather API, and present it in a user-friendly manner on an OLED screen.

We utilize several libraries, including Wire, Adafruit_GFX, Adafruit_SH1106, WiFi, WiFiClientSecure, HTTPClient, ArduinoJson, NTPClient, and WiFiUdp, to enable seamless communication between the ESP32, the AccuWeather API, and the OLED display. Each library plays a crucial role in establishing Wi-Fi connectivity, making HTTP requests, parsing JSON data, and displaying the relevant information on the OLED screen.

The project starts by setting up the Wi-Fi connection, ensuring we have a stable internet connection. We then proceed to initialize the OLED display and prepare it for data presentation. Using the AccuWeather API, we fetch real-time weather data, including temperature and weather description. Additionally, we synchronize the time using an NTP server to ensure accuracy. Finally, we update the OLED display with the retrieved information, showcasing the temperature, weather conditions, and current time.

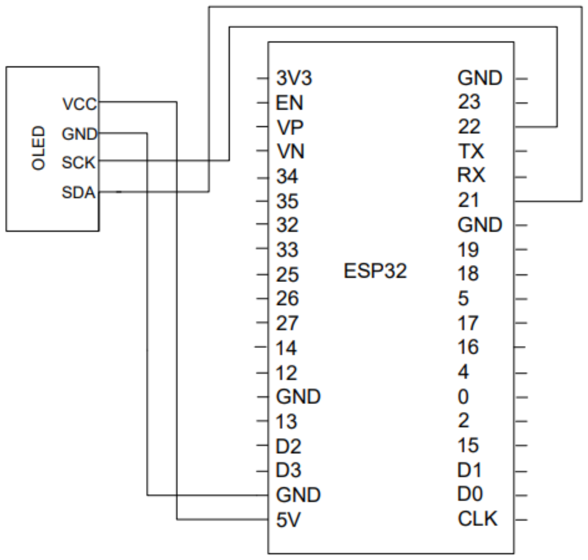

Figure 1Circuit Diagram

Pin Description

SDA: 22 || SCK: 21 || GND: GND || VCC: 5V

Step to generate AccuWeather API

1. Go to accuweather api (https://developer.accuweather.com)
2. Register
3. Go to my apps
4. Click on + Add a new App
5. Enter the details

Steps to Get your city code

1. Go to API Reference
2. Go to location API
3. In text Search select City Search
4. Enter your apikey and q(City Name)
5. Click on Sent this request
6. Scroll down to get your City Key

Library Details

Wire.h:

Purpose: This library provides the necessary functions to communicate with devices using the I2C (Inter-Integrated Circuit) protocol.

Significance: It is used to establish communication between the ESP32 and the OLED display, as the display uses I2C communication.

Adafruit_GFX.h:

Purpose: This library is a graphics library that provides common graphics functions and abstractions, such as drawing shapes, text, and pixels.

Significance: It is used to control and interact with the OLED display, allowing us to draw text and display information on the screen.

Adafruit_SH1106.h:

Purpose: This library is specifically designed to work with the SH1106-based OLED displays.

Significance: It provides the necessary functions and methods to initialize and control the SH1106-based OLED display used in the project.

WiFi.h and WiFiClientSecure.h:

Purpose: These libraries provide functions to connect to a Wi-Fi network and perform secure client operations over the network.

Significance: They are used to establish a Wi-Fi connection using the provided SSID and password. The connection is necessary to retrieve weather data from the AccuWeather API.

HTTPClient.h:

Purpose: This library enables making HTTP requests and handling responses.

Significance: It is used to send an HTTP GET request to the AccuWeather API, retrieve the weather data, and handle the response from the server.

ArduinoJson.h:

Purpose: This library allows parsing and generating JSON data.

Significance: It is used to parse the JSON response received from the AccuWeather API, extract the temperature and weather information, and store them in variables for display.

NTPClient.h and WiFiUdp.h:

Purpose: These libraries provide functions for retrieving accurate time from an NTP (Network Time Protocol) server.

Significance: They are used to synchronize the time on the ESP32 with an NTP server, ensuring accurate timekeeping for displaying on the OLED screen.

About NTPClient

WiFiUDP ntpUDP;

NTPClient timeClient(ntpUDP, "pool.ntp.org", utcOffsetInSeconds);

NTPClient timeClient(ntpUDP, "pool.ntp.org", utcOffsetInSeconds);: Here, we create an instance of the NTPClient class named timeClient. It takes three parameters:

ntpUDP: This is the WiFiUDP object we created earlier. It is passed as an argument to the NTPClient constructor, allowing the NTPClient to use the same UDP connection for time synchronization.

"pool.ntp.org": This is the NTP server domain name. The NTPClient will communicate with this server to retrieve the accurate time.

utcOffsetInSeconds: This is the UTC offset for our time zone in seconds. It specifies the time difference between our time zone and Coordinated Universal Time (UTC). For example, if our time zone is 5 hours ahead of UTC, we would set utcOffsetInSeconds to 5 * 3600 (5 hours * 3600 seconds per hour).

About HTTP

```
if (httpResponseCode == 200) {
  String payload = http.getString();
  DynamicJsonDocument doc(1024);
  deserializeJson(doc, payload);
```

if (httpResponseCode == 200): This condition checks if the HTTP response code is 200, which indicates a successful request. The HTTP response code is a standardized status code returned by the server to indicate the success or failure of the request. In this case, a code of 200 means the request was successful.

String payload = http.getString();: If the response code is 200, we retrieve the response content as a string using the getString() function of the HTTPClient library. The response contains the weather information in JSON format.

DynamicJsonDocument doc(1024);: Here, we create a DynamicJsonDocument object named doc to store and parse the JSON data. We specify a buffer size of 1024 bytes, which should be sufficient for the response data.

deserializeJson(doc, payload);: This line deserializes the JSON payload string and stores it in the doc object. It parses the JSON data into a structured format that can be easily accessed.