

BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

IMP project documentation

Display MQTT controlled by gestures
"Meteo App"

Contents

1	Overview	2
2	Usage	2
2.1	Initial screen	2
2.2	Weather screen	2
2.3	City selection screen	3
2.4	Waiting for data screen	3
3	Getting started	4
3.1	Hardware setup	4
3.1.1	Microcontroller	5
3.1.2	Display	5
3.1.3	Gesture sensor	5
3.2	Environment setup	5
4	Implementation	5
4.1	NVS initialization	5
4.2	Wi-Fi initialization	5
4.3	MQTT communication	6
4.4	Gesture task	6
4.5	Display task	7
5	Data exchange	7
6	References	8

1 Overview

This application is designed to provide weather information for a selection of cities. The application retrieves weather data in JSON format from an external source using MQTT connection. The data is then processed and displayed to the user.

The application also includes a graphical representation of the weather conditions. For example, the `sunny_icon` array appears to hold the pixel data for an icon representing sunny weather. The user can navigate through the list of cities, and the application will display the current weather conditions for the selected city. The navigation is controlled through gesture inputs.

This application is designed to run on an embedded system, by the use of the ESP-IDF library, FreeRTOS and other external libraries.

2 Usage

2.1 Initial screen

When the application starts, it displays a waiting screen. This screen is shown even while the application initializes and fetches the initial weather data, Wi-Fi and MQTT supporting structures. To start the application the user can use any gesture (supported ones are UP, DOWN, LEFT, RIGHT). "any gesture to start" is blinking every 0.5 seconds.



Figure 1: Initial screen

2.2 Weather screen

Once a city is selected, the application displays the current weather conditions for that city (initially it is Brno with rainy weather). The weather conditions are represented graphically. "Meteo App" supports four types of weather conditions: rainy, sunny, windy and snowy. Temperature is an integer, and the city has to fit in one row on display, which is 16 letters, otherwise it is truncated.

LEFT gesture moves us back to initial screen, UP and DOWN do nothing, and RIGHT opens a city selection screen.

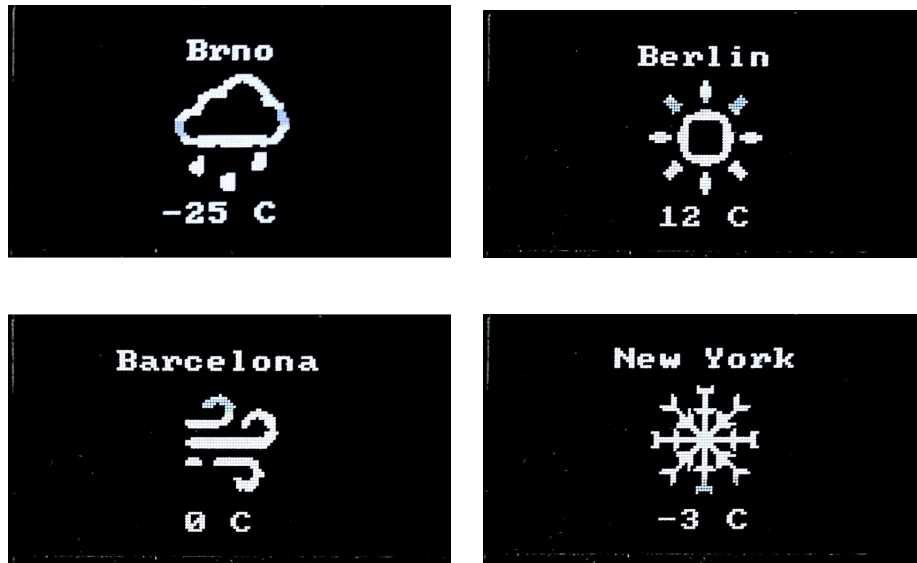


Figure 2: Weather screen

2.3 City selection screen

The application supports weather data for a selection of cities: Paris, Berlin, Prague, Brno, Barcelona, and New York. The user can navigate through this list of cities. The navigation is controlled through UP and DOWN gestures. LEFT returns us to weather screen, and RIGHT is a confirmation gesture of the selected city.

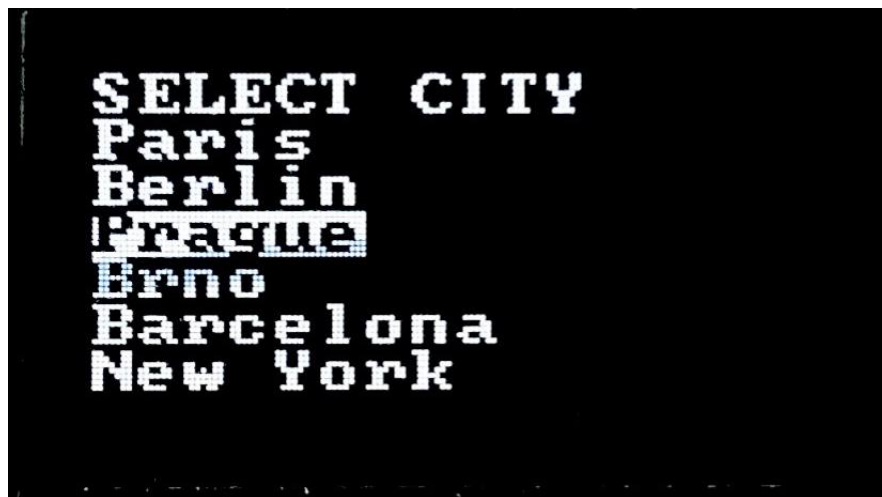


Figure 3: City selection screen

2.4 Waiting for data screen

This screen is showed once the city is selected. If the application receives the correct data (including matching the city name requested by the user), weather screen is shown with updated information. Otherwise after 30 seconds of waiting this screen disappears and the data remain unchanged. The whole screen is blinking every 0.5 seconds.



Figure 4: Waiting for data screen

3 Getting started

3.1 Hardware setup

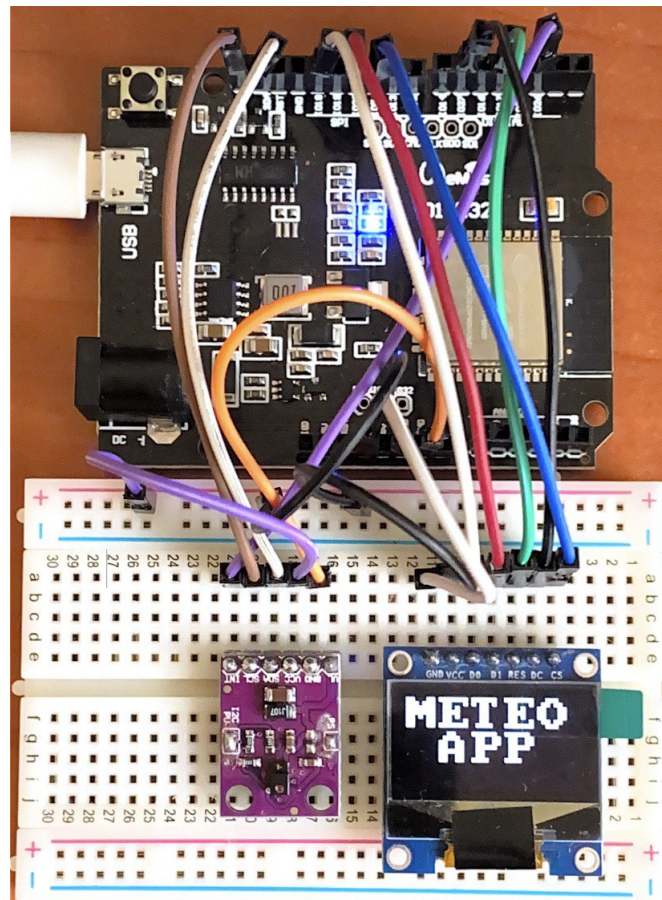


Figure 5: Hardware setup

3.1.1 Microcontroller

In this project, Wemos D1 R32 w/ ESP32 is used. Connect it with your PC using USB cable. You will also need a breadboard and wires to connect your microcontroller with other components. The pinout is described in [1]. Documentation: [6].

3.1.2 Display

In my case it is SSD1306 SPI display. Be aware that for I2C displays you will need to change the configuration. My connection from picture 5: GND - GND (IO32), VCC - 3V3 (IO15, using + on breadboard), D0 - SCK (GPIO18), D1 - MOSI (GPIO23), RES - GPIO17, DC - GPIO27, CS - GPIO5. Documentation: [8].

3.1.3 Gesture sensor

I am using GY-9960LLC APDS-9960. Connection from picture 5: INT - GPIO26, SCL - SCL (GPIO22), SDA - SDA (GPIO21), VCC - 3V3 (same as for display), GND - GND. Documentation: [10].

3.2 Environment setup

Make sure you have the following installed: ESP-IDF [3], PlatformIO IDE for VSCode [9]. Open this project in VSCode, connect your ESP32 device to your computer. Upload and monitor the project. You should now see the weather application running on your device.

4 Implementation

The whole project is implemented in one file `main.c` due to the multitude of global variables and parallel tasks. A file `weather.h` contains auxiliary constants for displaying the weather, `platformio.ini` is a configuration file for PlatformIO. It specifies the settings for the project.

4.1 NVS initialization

The `flash_init` function initializes the non-volatile storage (NVS) flash memory, which is used to store persistent data across reboots. If the initialization fails because there are no free pages in the NVS partition or a new version of the NVS partition format was found, it erases the NVS partition and tries to initialize it again.

The `ESP_ERROR_CHECK` macro checks the final result of the initialization. If it's not `ESP_OK`, the macro prints an error message to the console and aborts the program.

4.2 Wi-Fi initialization

The `wifi_init_sta` function initializes and configures the Wi-Fi station interface on the ESP32 device. It starts by initializing the TCP/IP stack and creating the default event loop. Then, it sets up the default Wi-Fi station and initializes the Wi-Fi driver with default configurations.

Event handlers for Wi-Fi and IP events are registered, which will handle events like Wi-Fi connection/disconnection and IP address changes.

The function then configures the Wi-Fi station with the SSID and password of the network it should connect to. By default I set up my mobile internet data, it has to be changed for a required Wi-Fi connection. The Wi-Fi mode is set to station mode, which means the ESP32 will connect to a Wi-Fi router.

Finally, the Wi-Fi station is started and a log message is printed to indicate the completion of the Wi-Fi setup and the details of the network it's trying to connect to.

4.3 MQTT communication

The `mqtt_init` function is responsible for setting up the MQTT client on the ESP32 device. It sets up the configuration for the MQTT client, specifying the address of the MQTT broker it should connect to, which in my case is `mqtt://broker.emqx.io:1883`.

Then, the event handler for the MQTT client is registered (`mqtt_event_handler`). It's called when an MQTT event occurs, such as when the MQTT client connects to the broker or when it receives data. Finally, it starts the MQTT client, which begins the process of connecting to the MQTT broker.

When the MQTT client connects to the broker, `mqtt_event_handler` subscribes to a topic (IMPproj). When data is received, the function prints the topic and data of the received message. If the first character of the data is not '{', it assumes it is not in JSON format, which means this is the data sent by our device itself, and breaks out of the switch case. If the data is in JSON format, it calls the `handle_incoming_mqtt_data` function to handle the data.

Received data is expected to be a JSON string containing information about a city's weather. `handle_incoming_mqtt_data` first parses the JSON data using an external library `cJSON` [2]. If the parsing fails, it logs an error and returns.

Then, it retrieves the city, weather, and temperature from the parsed JSON. If the city matches the selected city, it updates global variables with the city, weather, and temperature information. If the city does not match the selected city, it logs a message and returns.

4.4 Gesture task

The `i2c_bus_init` function configures and returns an I2C configuration in master mode. It specifies the GPIO numbers for the SDA and SCL pins, enables internal pull-up resistors for these pins, and sets the clock frequency for the I2C bus. This configuration is used to create a I2C bus instance.

The `apds9960_init` function initializes the APDS-9960 sensor. It creates the sensor, initializes the gesture sensor, and enables the gesture engine. If any of these operations fail, it logs an error message and returns.

For I2C bus and APDS-9960 I used these external libraries [5] [4].

The `xTaskCreate` function call creates a new FreeRTOS task named "gesture_task". This task runs the `gesture_task` function, which continuously reads gestures from the APDS-9960 sensor. If a gesture is detected, it sets a flag `gesture_changes` to 1 and stores the gesture. It then logs the opposite direction of the detected gesture (e.g., if the detected gesture is "DOWN", it logs "UP"), because in my connection sensor is turned.

The task then waits for 200 milliseconds before repeating the process. This delay helps to prevent the task from consuming all the CPU resources.

4.5 Display task

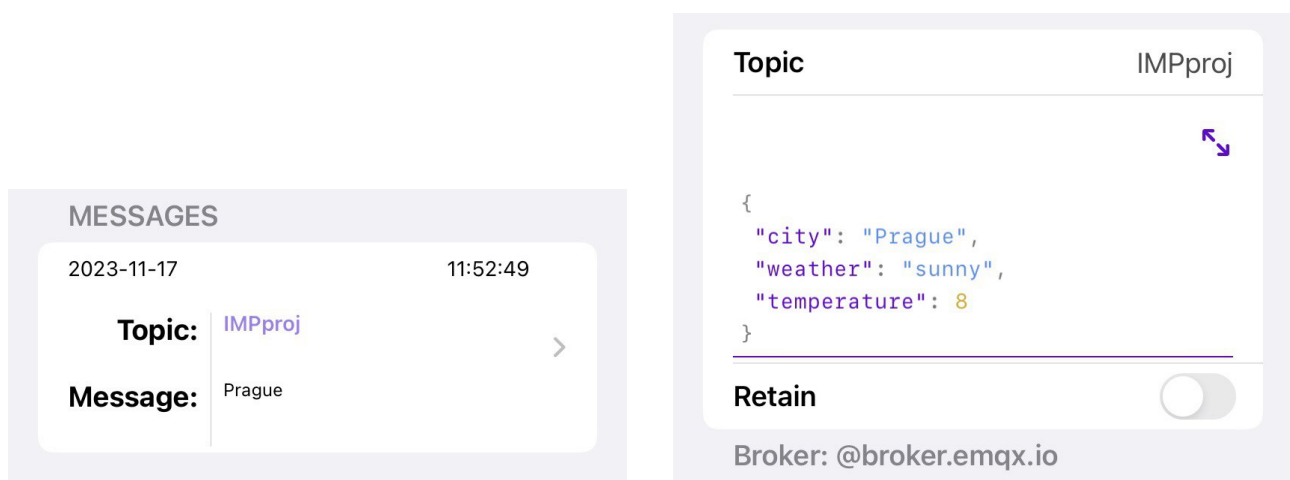
The `display_config` function configures and initializes an SSD1306 display connected via SPI. It initializes the SPI master with given GPIO numbers using the `spi_master_init` function. After that, it logs the panel size (128x64) and initializes the SSD1306 display with this size using the `ssd1306_init` function. SPI component is taken from [7].

Just like for gesture task, `xTaskCreate` creates a new FreeRTOS task named "display_task". The `display_task` function first clears the screen and sets the contrast. It then displays the title page. In an infinite loop, it displays the weather page and checks for gesture changes. The full logic of screen changes is described in 2. Each screen sets `gesture_changes` back to 0 to avoid several quick screen changes on one gesture.

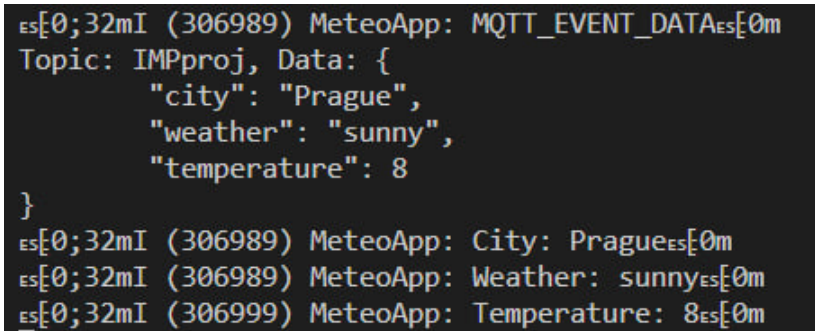
Global variable `can_change_data` ensures that stored weather data can change only when waiting screen (4) is displayed, otherwise incoming data is ignored.

5 Data exchange

For MQTT communication I use a mobile application called "EasyMQTT". When the user selects a city, a message with the corresponding city arrives on IMPproj topic. Based on received information, I send data from the application and the program processes them (4.3).



The screenshot shows the EasyMQTT mobile application interface. On the left, a 'MESSAGES' list displays a message received on the 'IMPproj' topic at 11:52:49 on 2023-11-17. The message content is 'Prague'. On the right, a detailed view of the message is shown, including the topic 'IMPproj', a JSON payload: `{ "city": "Prague", "weather": "sunny", "temperature": 8 }`, and a 'Retain' toggle switch. The broker is identified as '@broker.emqx.io'.



The terminal window displays MQTT event data for the 'IMPproj' topic. The first line shows the topic and a JSON payload: `Topic: IMPproj, Data: { "city": "Prague", "weather": "sunny", "temperature": 8 }`. Subsequent lines show individual data points being processed: `City: Prague`, `Weather: sunny`, and `Temperature: 8`.

Figure 6: Data exchange

6 References

- [1] Jamie C. D1 r32 – esp32. [Online] <https://designtech.blogs.auckland.ac.nz/d1-r32-esp32/>, 2021.
- [2] DaveGamble. cJSON. [Online] <https://github.com/DaveGamble/cJSON>.
- [3] Ltd. Espressif Systems (Shanghai) Co. Get started. [Online] <https://docs.espressif.com/projects/esp-idf/en/latest/esp32/get-started/index.html#installation-step-by-step>, 2016.
- [4] leeebo. apds9960. [Online] <https://github.com/espressif/esp-iot-solution/tree/master/components/sensors/gesture/apds9960>.
- [5] leeebo. bus. [Online] <https://github.com/espressif/esp-iot-solution/tree/master/components/bus>.
- [6] Wan Lei. esp-idf. [Online] <https://github.com/espressif/esp-idf/tree/master/examples/>, 2023.
- [7] nopnop2002. components/ssd1306. [Online] <https://github.com/nopnop2002/esp-idf-ssd1306/tree/master/components/ssd1306>.
- [8] nopnop2002. esp-idf-ssd1306. [Online] <https://github.com/nopnop2002/esp-idf-ssd1306>, 2023.
- [9] Platformio.org. Platformio ide for vscode. [Online] <https://platformio.org/install/ide?install=vscode>.
- [10] Avago Technologies. Apds-9960. [Online] https://www.laskakit.cz/user/related_files/av02-4191en_ds_apds-9960_2015-11-13.pdf, 2015.