Project Integration
Research Applied Computer Science

Jochem Arends
495637
Group 1

Academic Year: 2023-2024

Contents

Abbreviations

ACS	Applied Computer Science.
AIoT	Artificial Intelligence of Things.
ERD	Entity Relationship Diagram.
HTTP	Hypertext Transfer Protocol.
SDK	Software Development Kit.

Glossary

Bluetooth	A short-range wireless interconnection of mobile phones, computers, and other electronic devices.
WiFi	A wireless networking technology that uses radio waves to provide high-speed Internet access.

1 Introduction

As part of Project Integration, Applied Computer Science (ACS) students need to conduct research about the ESP32 microcontroller. Upon completion of this research, a set of assignments have to be completed. First, some introductory assignments in order to familiarize ourselves with concepts that are important for the project, followed by a final assignment that goes more in depth into specific topics that are of value for our final product.

2 The ESP32

The ESP32 is a series of microcontrollers developed by Espressif Systems that have integrated WiFi and Bluetooth modules (Espressif, 2024).

3 Espressif

Espressif Systems is a multinational semiconductor company that focusses among other things on developing wireless communication and Artificial Intelligence of Things (AIoT) (Espressif, n.d.). Popular products Espressif created, are the ESP32 series of chips, development boards, and modules (Espressif, n.d.). Espressif supports various open-source software projects such as Software Development Kits (SDKs), libraries, and tools. Espressif is a chinese company but has offices all around the world (Espressif, n.d.).

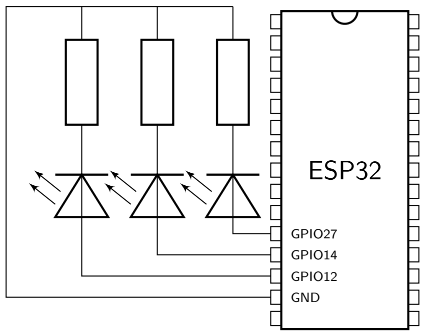
4 Lilygo

5 Assignment 1

Show that the ESP32 can turn three different LEDs on and off seperately using an internal loop with delays.

5.1 Circuit

For this assignment, the circuit used is quite straightforward. Connect three LEDs to the ESP32, put these LEDs in series with a resistor, and connect them to a common ground. For all subsequent assignments, it can be assumed that the same circuit is used unless explicitly stated otherwise.



5.2 Toolchain

For this assignment, I decided on using the ESP-IDF toolchain. More specifically, I used the `idf.py` command line tool in combination with a text editor. I never used the ESP-IDF toolchain before and was curious about what it would be like. One thing I really like about the ESP-IDF toolchain is that it provides a C++ compiler that supports both language and library features of more recent C++ standards, which is something the Arduino IDE lacks.

5.3 Software

```
#include <array>
#include <chrono>
#include <ranges>
#include <thread>

#include "driver/gpio.h"
#include "rom/gpio.h"

extern "C" void app_main() {
    const std::array<gpio_num_t, 3> led_pins{
        GPIO_NUM_12,
        GPIO_NUM_14,
        GPIO_NUM_27,
    };

    // configure the pins for output
    for (auto pin : led_pins) {
        gpio_pad_select_gpio(pin);
        gpio_reset_pin(pin);
        gpio_set_direction(pin, GPIO_MODE_OUTPUT);
    }

    // blink individual LEDs indefinitely
    for (auto pin : std::views::join(std::views::repeat(led_pins))) {
        gpio_set_level(pin, 1);
        std::this_thread::sleep_for(std::chrono::seconds{1});
        gpio_set_level(pin, 0);
    }
}
```

6 Assignment 2

Show that the ESP32 can turn three different LEDs on and off separately by sending commands over the serial interface.

6.1 Toolchain

Just like the previous one, for this assignment I have used the ESP-IDF toolchain. The ESP-IDF toolchain provides good C++ support. From the UART can be read using `std::cin`, but to get the usual behaviour, our program needs to initialize the UART driver first.

6.2 Software

```cpp
#include <iostream>
#include <map>
#include <ranges>

#include "driver/gpio.h"
#include "driver/uart.h"
#include "esp_vfs_dev.h"
#include "rom/gpio.h"

void toggle(gpio_num_t pin) {
    int level = gpio_get_level(pin) ^ 1;
    gpio_set_level(pin, level);
}

extern "C" void app_main() {
    // configure stdin to use blocking mode
    setvbuf(stdin, nullptr, _IONBF, 0);
    constexpr auto uart_num = CONFIG_ESP_CONSOLE_UART_NUM;
    uart_driver_install(static_cast<uart_port_t>(uart_num), 256, 0, 0, nullptr, 0);
    esp_vfs_dev_uart_use_driver(uart_num);

    const std::map<char, gpio_num_t> led_pins{
        {'0', GPIO_NUM_12},
        {'1', GPIO_NUM_14},
        {'2', GPIO_NUM_27},
    };

    // configure the pins for input and output
    for (auto pin : led_pins | std::views::values) {
        gpio_pad_select_gpio(pin);
        gpio_reset_pin(pin);
        gpio_set_direction(pin, GPIO_MODE_INPUT_OUTPUT);
    }

    for (char ch : std::views::istream<char>(std::cin)) {
        if (auto it = led_pins.find(ch); it != led_pins.end()) {
            toggle(it->second);
        }
    }
}
```

7 Assignment 4

Show that the ESP32 can turn three different LEDs on and off separately. Connect the ESP32 to a WiFi access point and host a webserver on the ESP32 to control the LEDs.

7.1 Toolchain

7.2 Software

`https://github.com/jochemarends/project-integration/tree/main/leds-wifi`

8 Final Assignment

For the final assignment, I am managing a database using SQL and the ESP32. I chose for this assignment because of the likelihood that the ESP32 has to interact with a database for the final product. The goal of this assignment is to make the ESP32 able to log certain events, which will be stored into a database. The event that will be logged is the pressing down of a push button. The ESP32 will not communicate directly with the database; instead, it communicates with the database via a web server. This is removes the need for the ESP32 to run a heavy SQL client and instead a much simpler HTTP client can be used.

8.1 Demonstration Setup

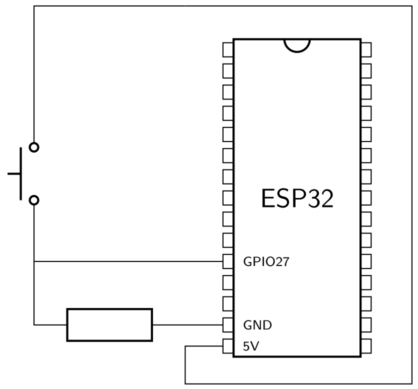


Figure 1: Electric Circuit

I used the circuit above to connect the push button to the ESP32. The locations of the pins may differ depending on which version of the ESP32 is being used. As for setting up the software, the client is flashed onto the ESP32 and the web server and the database are hosted a Raspberry Pi. I configured port forwarding on my home network (where the Raspberry Pi is connected to) so that the ESP32 also can communicate with the web server from other networks.

8.2 Development Environment and Tooling

For programming the ESP32, I decided on using the ESP-IDF. I have used the `idf.py` command-line tool for building and flashing the project. For configuring the web server and database I used OpenSSH, to establish a SSH connection. I wrote the code using the Vim text editor and used Git for version control.

8.3 Used Software Libraries

8.3.1 Client

For the client, I have used libraries that come with ESP-IDF. These libraries allowed me among other things to establish a WiFi connection and perform Hypertext Transfer Protocol (HTTP) requests. No third-party libraries were used.

8.3.2 Server

For the server, I have used Go's standard library packages for working with HTTP and SQL. However, Go's standard library package database/sql does not work with MySQL out of the box. I had to use a third-party package that contains a MySQL driver that implements the database/sql/driver interface (*Go MySQL Driver*, 2024).

8.4 Software Architecture

8.4.1 Basic Architecture

The software consists of three major components: a HTTP client, a HTTP server, and a relational database. The ESP32 is communicating with the web server and the web server communicates with the database. The client makes HTTP requests to the server, after which the server will query the database based on the request being made. The source code can be found on GitHub (Arends, 2024).

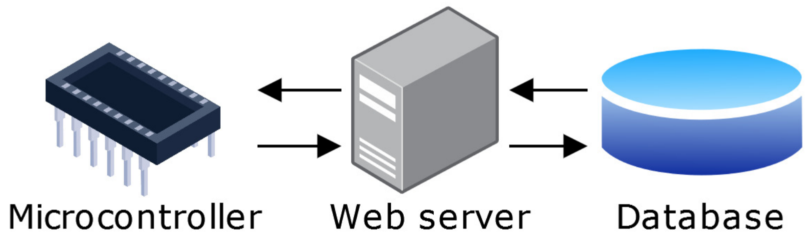


Figure 2: Basic Architecture Diagram

8.4.2 Database

The database is managed using MySQL, I have put the Entity Relationship Diagram (ERD) of the database below. A patient has a unique ID and a name, however, some additional information may be stored along this. Logs can be created that belong to a specific patient. The "Log" table contains a "kind" field, to account for the possibility of other events being added in the future.

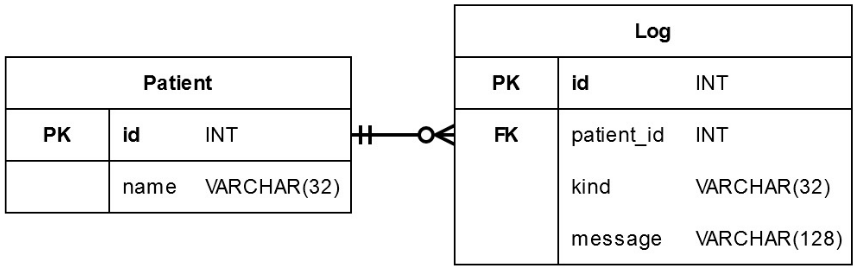


Figure 3: Entity Relationship Diagram

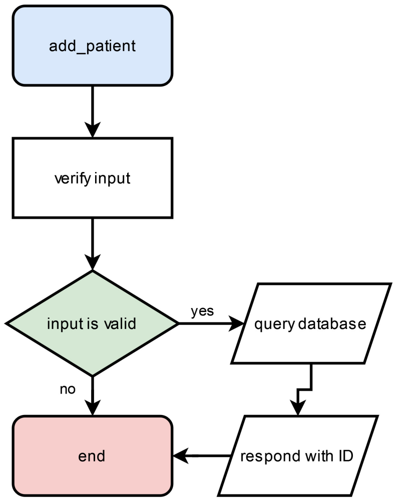
8.4.3 Web Server

The HTTP server provides two services: registering new patients and the logging events. Arguments are passed using form field. In the table below, the services are briefly documented.

Service	Path	Form Field(s)	Returns
Register Patient	`/add-patient`	`name=`*name*	An ID unique to the patient.
Log Event	`/log-event`	`id=`*id*`&msg=`*msg*	-

Table 1: Brief Description of Services

Below two flowchart can be seen, one for each service. The two are almost identical to each other, but the queries they perform are different. Also, the flowchart on the right side does not explicitly return data. Both flowcharts are used as submodule in the client's flowchart.



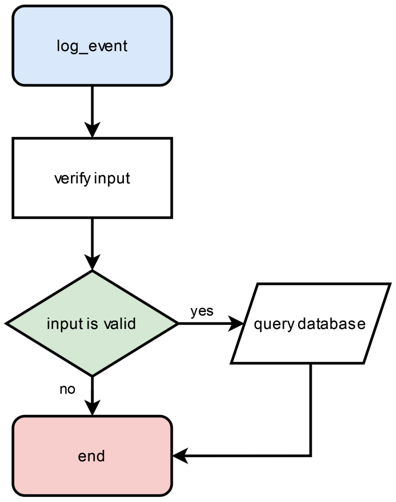


Figure 4: Flowchart of Server

8.4.4 Client

Upon booting, the client registers a new a patient by making a HTTP request to the web server. After this, the client will check for a button press, if one occurred, a new HTTP request will be made for logging the event. A flowchart for the client can be seen below.

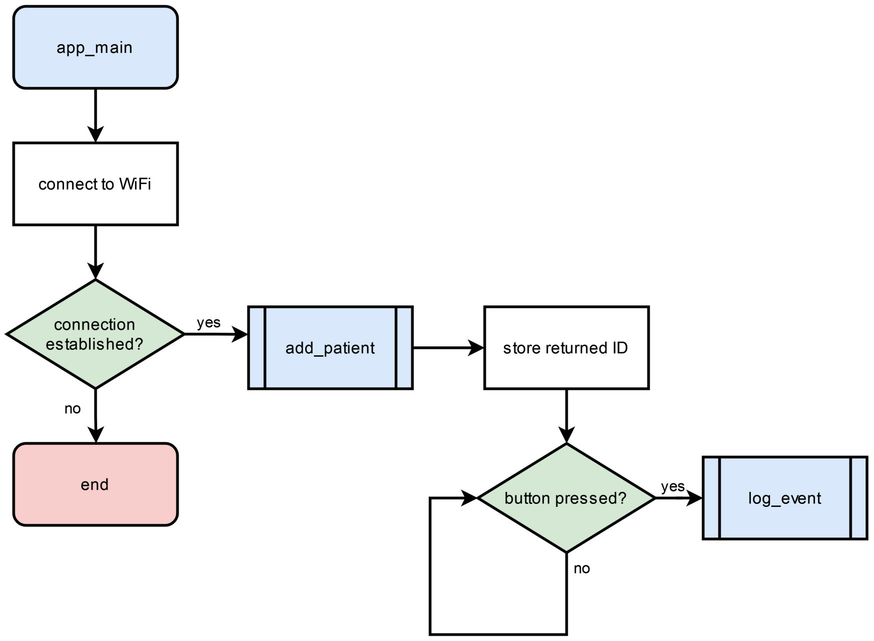


Figure 5: Flowchart of Client

8.5 Results

At first instance, I wanted to query the database directly from the ESP32. However, soon found out that this was not a trivial task as it would require the ESP32 to run a SQL client. I decided on using a web server, which I think is a better solution since it encapsulates access to the database and prevents one from performing malicious queries. This was the only challenge I have encountered.

8.6 Reccomendations

If I were to continue working on this small project, I would change several things. I would add some sort of authentication to prevent anyone from inserting data for arbitrary patients. Additionaly, I would change the client, so it would refrain from registrating a new user every time it boots. Lastly, I would add timestamps to the logs.

8.7 Accountability

I have done this assignment by myself.

References

Arends, J. (2024). *Source Code, Final Assignment.* `https://github.com/jochemarends/project-integration/tree/main/final-assignment`.

Espressif. (n.d.). *About Espressif.* `https://www.espressif.com/en/company/about-espressif`.

Espressif. (2024). *ESP32 Series Datasheet.* `https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf`.

Go MySQL Driver. (2024). `https://github.com/go-sql-driver/mysql`.