IoT system for sensors data acquisition and controlling devices via Web

Dominik Kędzierski, Piotr Matuszak

Intermediate Project under supervision of Witold Paluszyński Ph.D.
Embedded Robotics, Faculty of Electronics
Wrocław University of Science and Technology

January 30, 2019

Abstract

The report summarises works on event-based system which monitors a state of a room being opened or closed by detecting a key being hanged on or hanged up from an *intelligent key hanger* equipped with Wi-Fi module and informs users about these events via website. Additionally, system allows to control a light source in a room via Wi-Fi control module with minimal delay. The result of this project is a stack of services and embedded systems to control and acquire data with almost imperceptible delay via the Internet, using modern protocols and frameworks.

1 Introduction

1.1 Background of the project

Student organisations have their workshops in buildings of the university. Due to university's regulations, only a few of members of an organisation have necessary permission to get a key to their workshop from a concierge desk. Other members of student organisation would like to have an ability to check whether their organisation's workshop is open.

An ability to control a device from any location is a key part of all Internet of Things systems. A control part would prove a two-way nature of the proposed architecture by expanding its capabilities.

1.2 Assumptions and goals of the project

Project aims to solve a problem described in section 1.1 by creating an architecture for near real-time web-based system which gives a possibility to control and gather data from IoT modules. For the data acquisition part, a key module was designed and created, and for the control part of the project – a Wi-Fi controller for a LED strip.

2 Architecture and project design

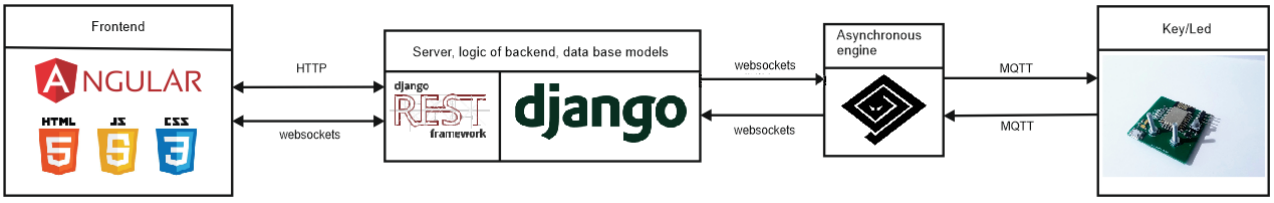


Figure 1: An architecture of the system

The whole system consists of the following parts:

- a website front-end, based on Angular framework,
- a server back-end, based on Django framework,
- an asynchronous engine based on `twisted`, `autobahn` and `paho-mqtt` libraries which extends the back-end capabilities to use MQTT protocol,
- IoT modules with ESP8266 Wi-Fi module: LED strip controller and Key module for control and data acquisition, able to communicate using Wi-Fi IEEE 802.11 LAN protocol.

Communication within a system is based on MQTT and Websocket protocols. All modules of the server use Docker containers.

2.1 Communication protocols

The choice of communication protocols was driven by their ability to facilitate real-time data transfer with low overheads.

2.1.1 Websocket

Websocket is a full-duplex protocol over a single TCP connection. It is a popular web protocol supported by all modern browsers and backend frameworks.

When the connection is first established between a server and a client, the protocol allows messages to be passed back and forth without client requesting for the data, while keeping the connection active. This ability makes Websocket a preferred protocol over HTTP, where a data packet from the server has to be requested by a client.

Websocket allows the server to push a message to all connected clients when an event occurs, for example a new data from a module appeared on the server.

Websocket was also used in communication between server back-end parts.

2.1.2 MQTT

MQTT is a publish-subscribe based protocol designed to operate in systems *where a small code footprint is required and/or network bandwidth is at a premium* [1]. For the proposed system, it is a preferred communication protocol between a web server and IoT modules. It allows to maintain an open full-duplex communication channel between modules and server and can be used to detect when an IoT module is offline. Its implementation is lightweight which makes it ideal for embedded systems.

2.2 Docker

The project is using Docker Containers to logically divide different system elements. Such approach enable programmers to easily create distributed system where most of functions are not assigned to hardware, but can run anywhere.

Docker containers wrap a piece of software in a complete filesystem that contains everything needed to run: code, runtime, system tools, system libraries – anything that can be installed on a server. This guarantees that the software will always run the same, regardless of its environment.

Docker containers running on a single machine share that machine's operating system kernel; they start instantly and use less compute and RAM. Images are constructed from filesystem layers and share common files. This minimizes disk usage and image downloads are much faster [2].

2.3 Website front-end

Angular is a TypeScript-based front-end framework used to build websites. It was chosen due to its simple-to-use two-way data binding mechanism [3].

Two way data binding allows the webpage to alter its DOM immediately when a data structure is updated. In case of this project, data is being sent to the front-end client via Websocket connection. Arriving data update the data structure on the client side. Two way binding then immediately updates the view presented in the browser (Figure 2).

When a new module starts to communicate with the server, it automatically appears on a website. No additional configuration on a website is needed.

2.4 Server back-end

Django is a Python back-end framework which allows to easily design a website back-end, create a database model, set an API and many other. Additionally, Django Channels extension was used to extend Django abilities to use Websocket protocol.

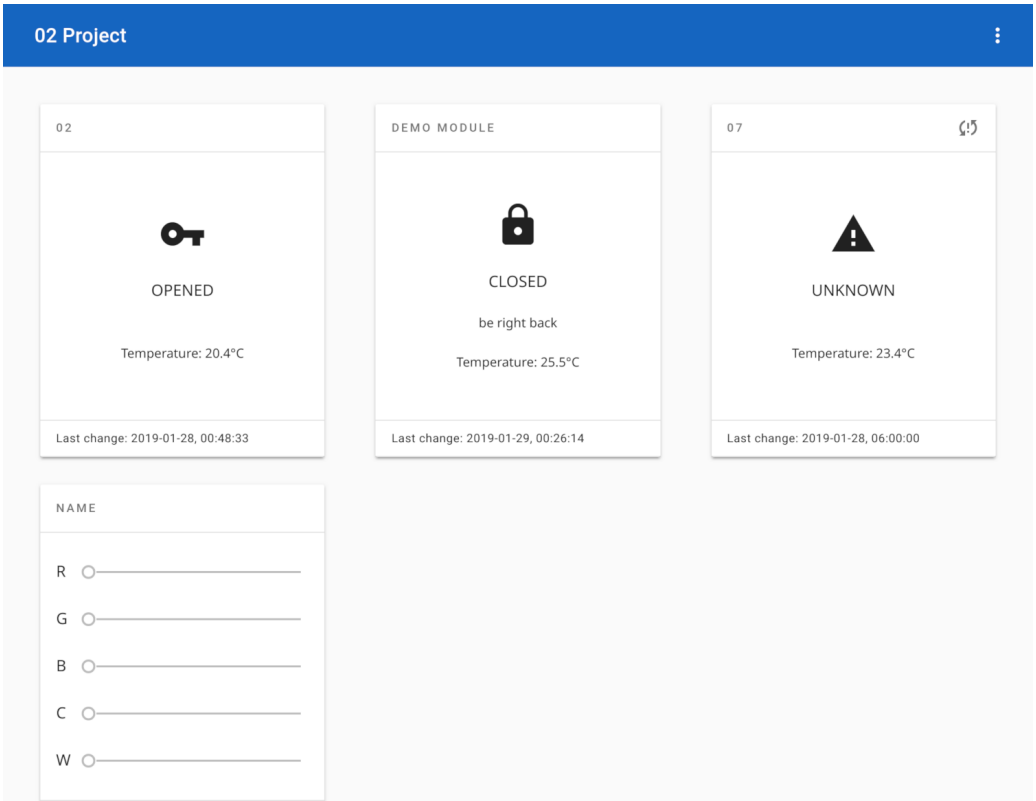


Figure 2: Website front-end

A key part of a quick webpage-side data update is a Django `signals` mechanism. It is a set of implementable events which are executed before or after a certain operation on a database occurs. In the project, when a new data on module appears in the database, a correct `signal` is triggered and data is pushed to clients. This is necessary for a quick data update on client side (either website or a module).

2.5 Asynchronous engine

This part of back-end is necessary to use MQTT protocol – front-end (browsers) and Django framework do not support MQTT protocol in easy way. Moreover it is worth to reduce a number of activities performed in the worker container with Django and database. It was decided to create a separate container with asynchronous engine to translate requests from Websocket to MQTT and vice versa. This solution works very well and relieves container with Django.

2.6 IoT modules

There are two types of modules in the system:

- Key module, which acquire data like: key hanged on / hanged up, temperature,
- LED controller module, which allows to control five-channel LED strip.

Both types of modules are based on ESP8266 WiFi module programmed using Arduino ESP8266 library.

Modules connect to WiFi network selected by user, react to events (hanging a key, pressing a button) by sending data using MQTT protocol to correct MQTT topics. LED controller subscribes necessary topics for control purposes and react on each message by changing a color intensity. Moreover, a Key module sends a temperature data regularly (every 60 seconds).

2.6.1 Key module

Key module can be powered using microUSB cable (5V) – a current standard for portable devices' chargers.

Key module primary goal is to detect a key on a key hanger, consisting of two metal screws, one connected to `GND`, the other to `ADC` of the ESP8266 board. Should a key be hanged, a change in voltage can be detected and the data can be sent to the server. This method was chosen as the most reliable. Other solution – capacity sensor – was rejected as a key detection method based on previous experience of such detection system: it was unreliable and detected an event in random moments.

An additional option to notify a "I'll be right back" (in case of hanged up key) or "I will leave soon" (in case of hanged on key) on a website is available under the button on the module.

By pressing a button for more than 2 seconds, a network to which a module is connected can be changed. When this happens, a module becomes an Access Point to which a user can connect and set up SSID and password. Additionally, user can change a name under which this module is shown on a website. After providing new network credentials, module goes back to the client mode and connects to the new router. This ability is implemented using `WiFiManager` library [4].

The temperature is measured using DS18B20U+ digital thermometer, which communicates with ESP8266 using OneWire protocol.

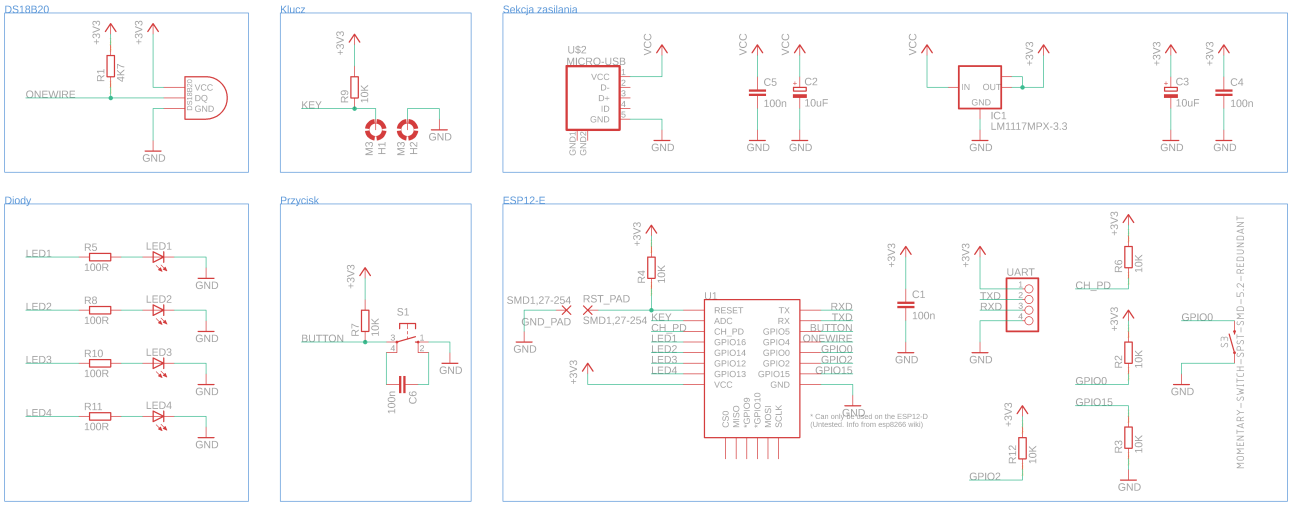
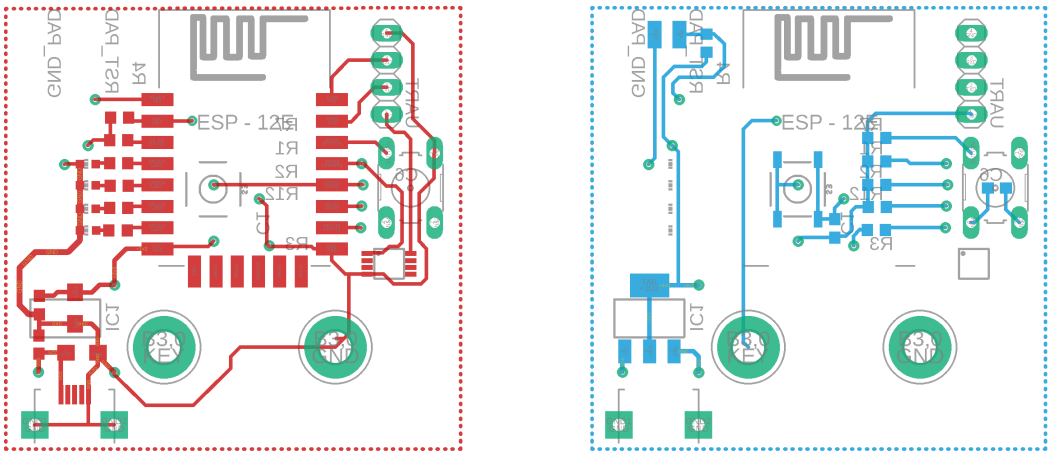


Figure 3: Schematic of key module



a) b)

Figure 4: PCB design of key module a) Top layer b) Bottom layer

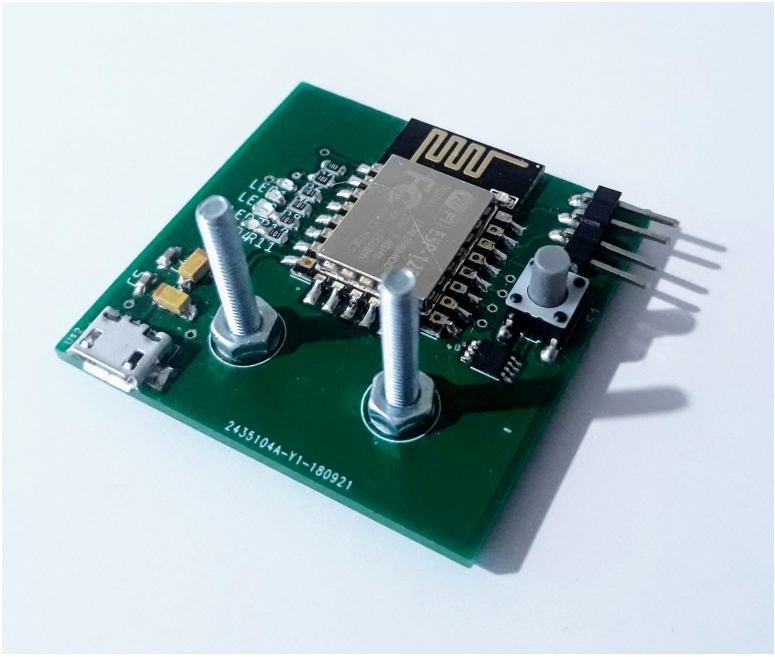


Figure 5: Soldered key module

2.6.2 LED module

LED module is powered from 24V DC (this is a requirement of a LED strip used). To power ESP module DC-DC converter was used.

Module consists of ESP8266 board connected to MOSFET transistors. These MOSFETs allow to control resultant voltage on strip by PWM signal. Similar transistors for [5], [6] available in local stores were searched. IRML2060 transistors was selected. This transistors has similar parameters.

Module subscribes MQTT topics for each of controlled colours (red, green, blue, cold, warm) and act accordingly, setting a PWM duty cycle thus controlling the intensity of each colour.

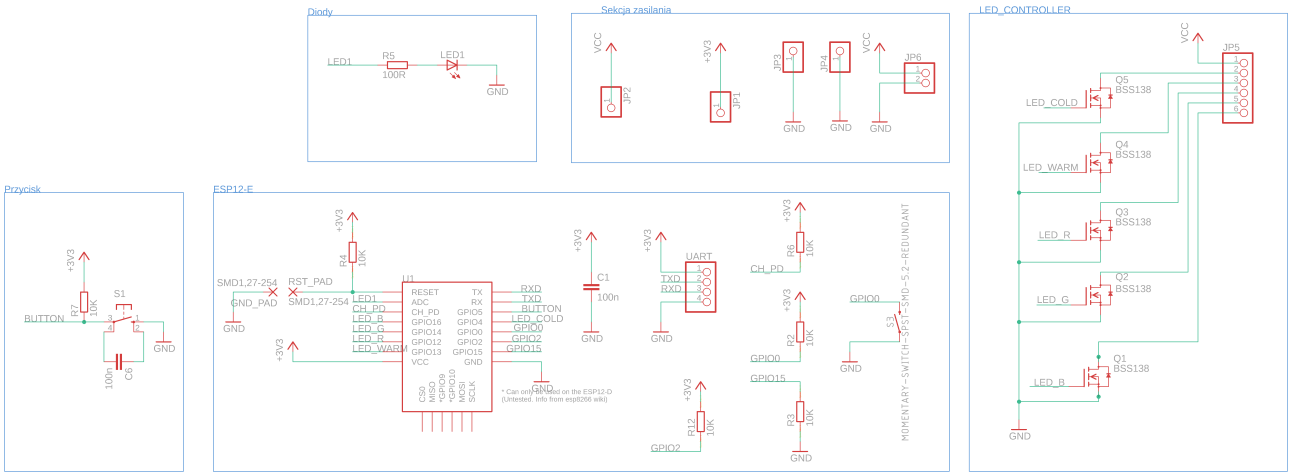
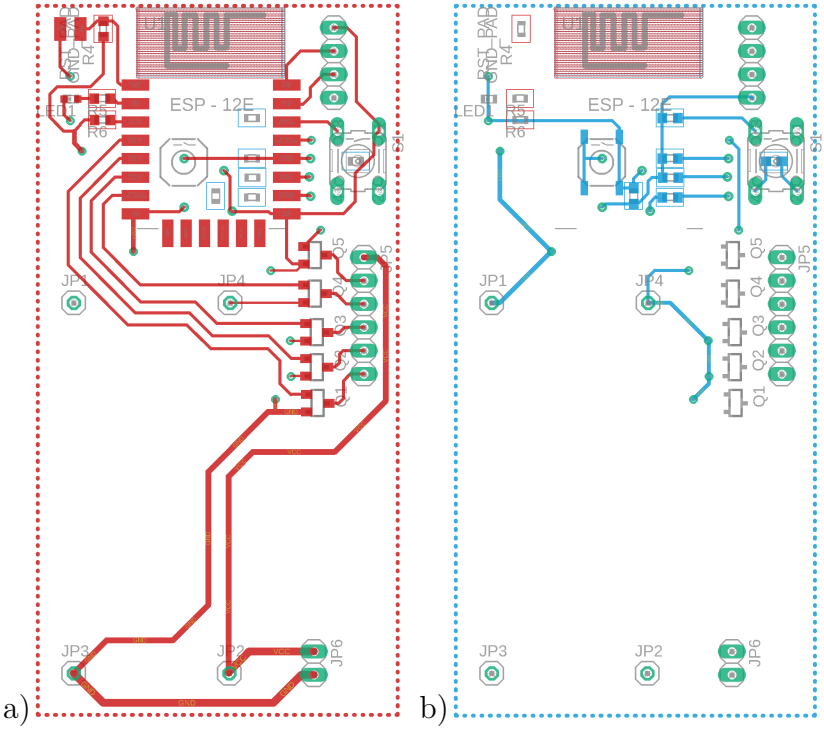


Figure 6: Schematic of LED controller module



a) b)

Figure 7: PCB design of led controller module a) Top layer b) Bottom layer

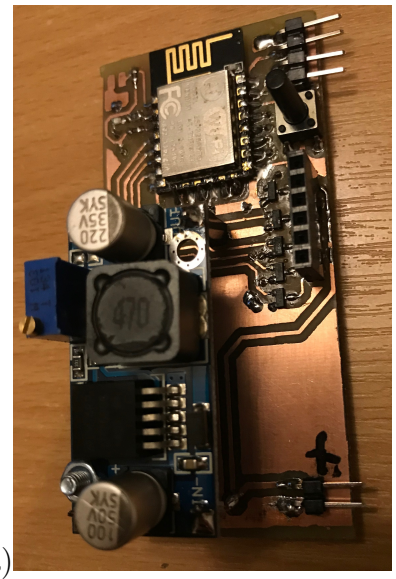
a)  b) 

Figure 8: Led controller module a) Soldered module b) Working module

3 Project results and summary

The project result is an architecture of a web-based system which is designed to minimise the delay between obtaining the data and presenting the data to the website user or user controlling the device via website interface and device reacting to the control signal. Moreover, the architecture ability can be proven using designed IoT modules.

The architecture designed to minimise the delay in transmission of the information between all parts of the system allows to control the LED strip with almost imperceptible delay (usually in a matter of less than half of a second), although the data is transmitted over the Internet (the results may vary depending on a quality of connection, Internet traffic etc.). Also, the data obtained by the Key module are pushed through the system and presented immediately to the user without any necessity to reload the page or to wait for specified amount of seconds.

Modules are programmed to be connected easily into the system and are manageable using any device with Wi-Fi capability. New modules are automatically registered on a website which minimises the amount of configuration steps needed for the user and makes the system as easy as possible to use.

References

[1] mqtt.org community members. mqtt.org. `http://mqtt.org/faq`.

[2] Docker. Why Docker. `https://www.docker.com/what-docker`.

[3] Google. Angular – NgModel. `https://angular.io/api/forms/NgModel`.

[4] tzapu. WiFiManager. `https://github.com/tzapu/WiFiManager`.

[5] Philippe Libioulle. A WiFi-enabled RGB LED strip controller. `https://www.stavros.io/posts/wifi-enabled-rgb-led-strip-controller/`.

[6] Stavros Korokithakis. RGB LED Strips Controller. `https://www.hackster.io/ThereIsNoTry/rgb-led-strips-controller-b15300`.