Bachelor Thesis

Thesis title

Author:
YourName YourSurname

Supervisor:
Name of Supervisor

A thesis submitted in fulfillment of the requirements for the degree of Engineer (Ing.)

in the

Department of Cybernetics

April 5, 2021

Declaration of Authorship

I, YourName YourSurname, declare that this thesis titled, "Thesis title" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.

Signed:

Date:

"*Something supersmart.*"

Your hero

UNIVERSITY OF WEST BOHEMIA

Abstract

Faculty of Applied Sciences

Department of Cybernetics

Engineer (Ing.)

Thesis title

by YourName YourSurname

Your abstract goes here...

Acknowledgements

Your acknowledgements go here...

Contents

List of Figures

List of Tables

List of Abbreviations

REST	**RE**presentational **S**tate **T**ransfer
JSON	**J**ava**S**cript **O**bject **N**otation
BSON	**B**inary **JSON**
MQTT	**M**essage **Q**ueuing **T**elemetry **T**ransport
HTTP	**H**yper**t**ext **T**ransfer **P**rotocol

Chapter 1

Introduction

Your intro... Thesis ref example: **bulin:2016**, Misc ref example: **smidl:pc**,
Article ref example: **mcculloch:neuron**, Online webpage ref example: **online:xor_solution**

1.1 State of the Art

1.2 Thesis Objectives

1.3 Thesis Outline

Chapter 2

Dialog Systems

Chapter 3

Backend

For backend is developed own engine that runs on Raspberry Pi 3. The whole engine is coded in python, and aim practices wrote below.

- Simplicity: Write straightforward code that is easily understandable for later rewriting.
- Modifiable: Write independent modules that can connect to others.
- Modularity: Write simple parts connected by clean interfaces
- Robustness: Robustness is the child of modularity and simplicity

3.1 Diagram description

This section will briefly describe the architecture of the engine that is figured on a diagram - see Fig. 3.1.

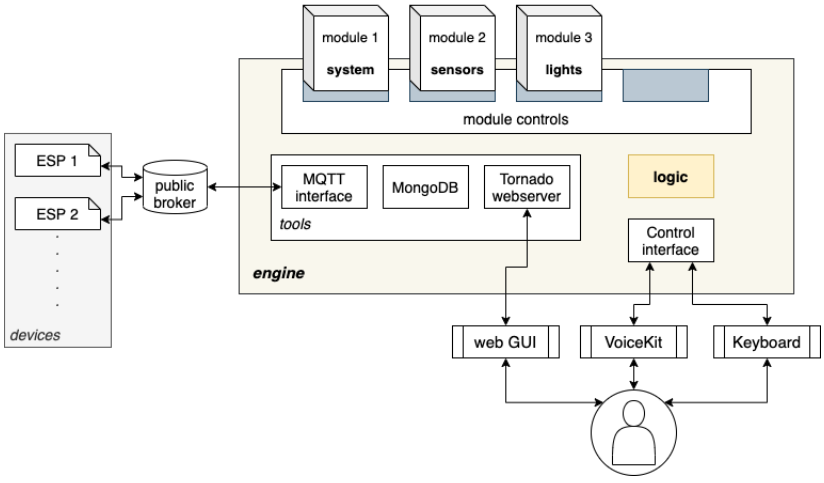

Figure 3.1: Project architecture

Engine use tools like MQTT[1], MongoDB, Tornado web server that will be described later. Each of them runs in its thread and concurrently. These tools create a basis for modules and mediate main functionalities such as database, web server and communication.

[1] **M**essage **Q**ueuing **T**elemetry **T**ransport

The engine is designed to easily remove, add or update any modules mutually independent that define functions used by a user interface. Each module will be described in the chapter 4.

The engine also contains a separate block for logic. This block captures a command from the Voice Kit or keyboard interface, then browsing a predefined list of each module's commands and determines the best match for the user voice command or command written on the keyboard.

3.2 Database

An auxiliary database is needed to store all the sensors data. For this purpose, it is used MongoDB due to the following advantages.

MongoDB (Jayaram, 2020) is an open-source document database build upon a NoSQL database and written in C++. Database's horizontal, scale-out architecture can support vast volumes of both data and traffic. One document can have others embedded in itself, and there is no need to declare the structure of documents to the system - documents are self-describing.

Before using this type of database, we have to be familiar with different terminology instead of traditional SQL databases.

SQL Server	MongoDB
Database	Database
Table	Collection
Index	Index
Row	Document
Column	Field
Joining	Linking & Embedding
Partition	Sharding (Range Partition)
Replication	ReplSet

Table 3.1: MongoDB terminology

We use this type of database because it is famous for its use in agile methodologies, and the project tends to enlarge in the future. The main benefits are:

- Document Oriented
- High availability - replication
- High scalability - sharding
- Dynamic - no rigid schema
- Flexible - field addition/deletion have less or no impact on the application
- Data representation in JSON[2] or BSON[3]
- Document-based query language that is nearly as powerful as SQL

[2] **J**ava**S**cript **O**bject **N**otation
[3] **B**inary **JSON**

For adding a new field, the field can be created without affecting all other documents in the collection, without updating a central system catalog, and without taking the system offline.

In the project, we save all incoming messages from MQTT to MongoDB to a collection based on a name of interest module.

3.3 Communication

The backbone of the whole project is communication between a variety range of devices over the internet. Therefore had to been found robust, scalable and cost-effective protocols that will transmit messages and data securely. Based on the survey was chosen three protocols that, in combination, satisfy all our requirements, and we will delve deeper into them in the following sections.

3.3.1 MQTT

MQTT (Malý, 2016) is a standardized protocol by the OASIS MQTT Technical Committee used for message and data exchange. The protocol is designed specifically for the Internet of Things. The protocol is developed in vast language diversity from low-level to high-level programming language and designed at light versions for low-performance devices. Hence, it suits our use-case perfectly because each module possesses tons of various devices with limited resources that are already included or will arise later on.

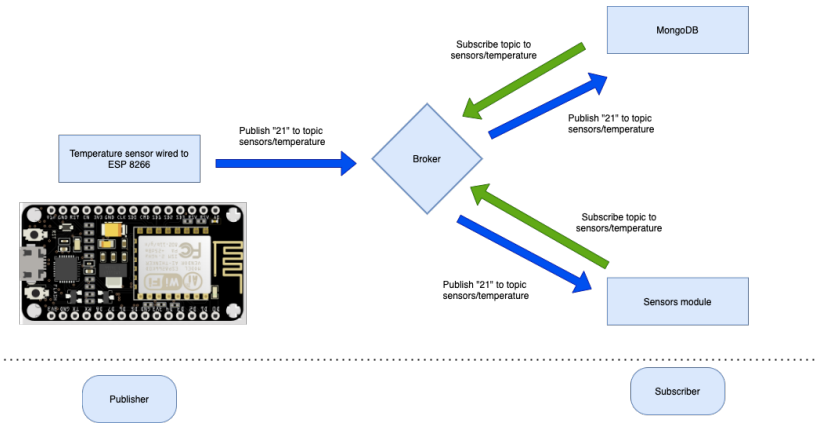

FIGURE 3.2: MQTT pub/sub pattern

The design principles are to minimize network bandwidth and device resource requirements whilst also attempting to ensure reliability and some degree of assurance of delivery. The protocol check errors by TCP and orchestrate communication by the central point - broker. The protocol architecture uses a publish/subscribe pattern (also known as pub/sub) shown in Fig. 3.2, which provides an alternative to traditional client-server architecture. Architecture decouples publishers and subscribers who never contact each other directly and are not even aware that the other exists. The decoupling giving us the following advantage:

- Space decoupling: Publisher and subscriber do not need to know each other.
- Time decoupling: Publisher and subscriber do not need to run at the same time.
- Synchronization decoupling: Operations on both components do not need to be interrupted during publishing or receiving.

So, after some publisher publishes his message, it is handled by the broker who filters all incoming messages and distributes them to accredited subscribers. Filtering can base on topic or subject, content and type.

In our MQTT case, the messages are subject-based filtering, which means the filtering is based on a subject or topic and is part of each message. The client subscribes to the topics he is interested in, and the broker distributes the messages accordingly as shown in Fig. 3.3.

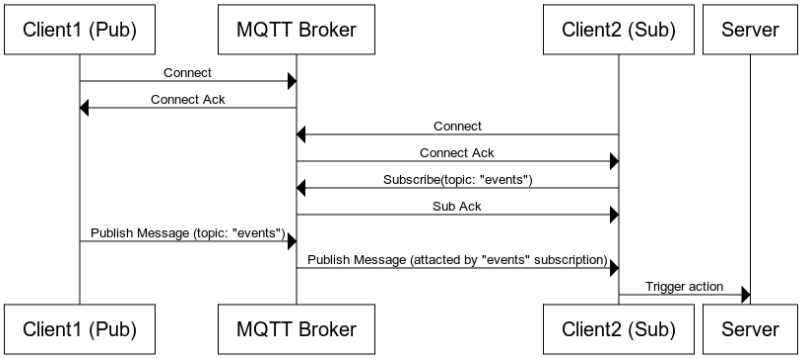

FIGURE 3.3: MQTT pub/sub communication diagram

The topics are generally strings with a hierarchical structure that allow different subscription levels. It is feasible to use wildcards to subscribe, for example, sensors/# to receive all messages related to the sensors, such as sensors/temperature or sensors/illuminance.

The MQTT protocol has the Quality of Service (QoS) levels essential to any communication protocol. The level of QoS can be specified for each message or topic separately according to its importance.

In MQTT, there are 3 QoS levels:

- QoS 0: This level is often called "fire and forget" when a message is not stored and retransmitted by a sender.
- QoS 1: Is is guarantees that a message is delivered at least one time to the receiver. The message is stored on a sender until it gets a PUBACK packet from a receiver.
- QoS 2: It is the highest level, and it guarantees that each message received only once by the intended recipients.

It is vital to mention MQTT have the feature retained messages that are mechanisms where the broker stores the last retained message for a specific topic. This feature allows a client does not have to wait until a new message is published to know the last known status of other devices.

3.3.2 WebSocket

Now, all we need is communication between the client and the engine. WebSocket (Wang, Salim, and Moskovits, 2013) provides a low-latency, persistent, full-duplex connection between a client and server over TCP. The protocol is chiefly used for a real-time web application because it is faster than HTTP[4] concerning more transfers by one connection. The protocol belongs to the stateful type of protocols, which means the connection between client and server will keep alive until either client or web server terminate it. The protocol fits for us in use between client and web server in case of real-time response.

The main benefits are:

- Persistent - After an initial HTTP handshake, the connection keeps alive using a ping-pong process, in which the server continuously pings the client for a response. It is a more efficient way than establishing and terminating the connection for each client request and server response. Server terminating connection after an explicit request from the client, or implicitly when the client goes offline.
- Secure - WebSocket Secure uses standard SSL and TLS encryption to establish a secure connection. Although we do not pursue this issue in our work, it is a valuable feature to add later.
- Extensible - Protocol is designed to enabling the implementation of subprotocols and extensions of additional functionality such as MQTT, WAMP, XMPP[5], AMQP[6], multiplexing and data compression. This benefit makes WebSockets a future-proof solution for the possible addition of other functionalities.
- Low-latency - WebSocket significantly reduces each message's data size, drastically decreasing latency by eliminating the need for a new connection with every request and the fact that after the initial handshake, all subsequent messages include only relevant information.
- Bidirectional - This enables the engine to send real-time updates asynchronously, without requiring the client to submit a request each time, as is the case with HTTP.

We will apply this protocol for transfer between clients such as Voice Kit, keyboard or web interface and engine in case of real-time response.

3.3.3 REST

In other cases like fetching data only once or data that is not required very frequently, we use RESTfull[7] web service on a web server. This service enables us to transfer a lightweight data-interchange format JSON trivially and reliably - see Fig. 3.4. We use a standard GET REST request on a defined URI and then decode it like JSON for fetching data.

[4] **H**yper**t**ext **T**ransfer **P**rotocol
[5] E**x**tensible **M**essaging and **P**resence **P**rotocol
[6] **A**dvanced **M**essage **Q**ueuing **P**rotocol
[7] **RE**presentational **S**tate **T**ransfer

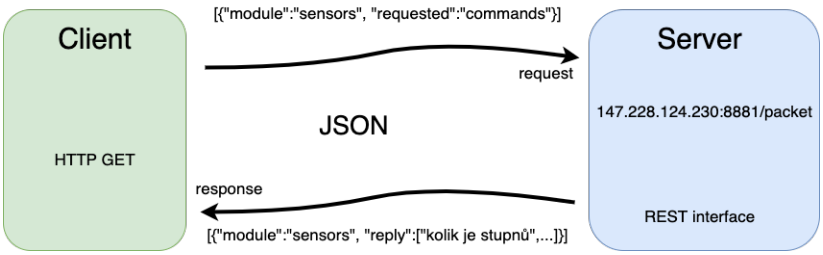

FIGURE 3.4: REST principle

3.4 Controllers

3.4.1 Keyboard

The keyboard is a python script with a particular purpose for developing new voice commands. This script opens up a CLI[8] built upon a voicehome controller. Over the command-line can be easily typed a voice command with high accuracy, thus fully debugged in various forms.

3.4.2 Voice Kit

Voice kit (*Voice Kit*) is a building kit made by google that lets users create their natural language processor and connect it to the Google Assistant or Cloud Speech-to-Text service. By pressing a button on top, users can ask questions and issue voice commands to their programs. All of this fits in a handy little cardboard cube powered by a Raspberry Pi.

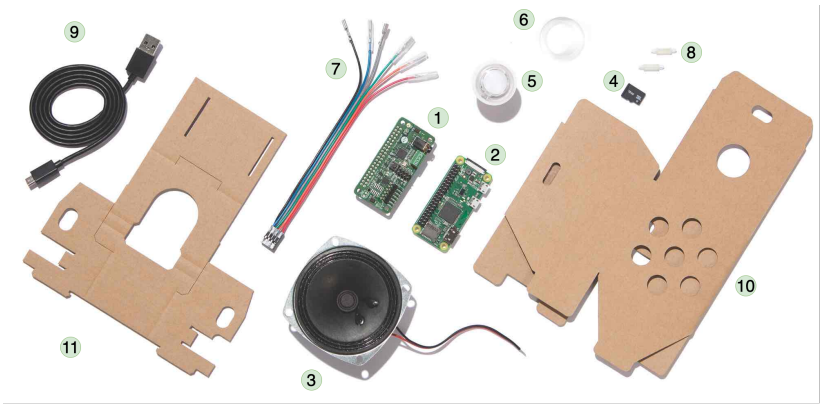

FIGURE 3.5: Voice kit materials

The user must assemble the cube himself from the following components showed in Fig. 3.5:

1. Voice bonnet
2. Raspberry Pi Zero WH
3. Speaker
4. Micro SD card

[8] **c**ommand-**l**ine **i**nterface

5. Push button
6. Button nut
7. Button harness
8. Standoffs
9. Micro USB cable
10. Speaker box cardboard
11. Internal frame cardboard

In this project, we connected the Voice kit to the faculty Speechcloud from chapter 2.

3.4.3 Website

As the second interface next to the already mentioned Voice Kit is a website. The server is coded in python with the tornado framework.

The website's architecture aims to use it via a portable device like a smartphone and tablet or touch screen attached to the wall. Therefore the website is constructed to be responsible, straightforward and touch-friendly. The website's use-cases are to able the user to monitor ESP, sensors, lights, weather and voice commands, display historical sensors data, feasible voice commands and description of them, trigger lights and modules.

To build up a web page are further used libraries and frameworks like Bootstrap, JQuery and Dygraphs, used to facilitate work and make the page more robust. Each of them is described in the sections below. Web site use for fetching data from engine already mentioned WebSocket and REST API.

Chapter 4

Modules

test

Chapter 5

Examples

5.1 XOR Function

Chapter 6

Discussion

Discussion starter...

6.1 Recapitulation of Methods

6.2 Summary of Results

Chapter 7

Conclusion

Conclusion text...

MongoDB project's application is as simple as possible because it is not the topic of the thesis. Is there plenty of room for improvement and streamlining.

7.1 Future Work

Outlook...

Bibliography

[1] Vanessa Wang, Frank Salim, and Peter Moskovits. "The WebSocket API". In: *The Definitive Guide to HTML5 WebSocket* (2013), 13–32. DOI: 10.1007/978-1-4302-4741-8_2.

[2] Martin Malý. *Protokol MQTT: komunikační standard pro IoT*. 2016. URL: https://www.root.cz/clanky/protokol-mqtt-komunikacni-standard-pro-iot/.

[3] Prashanth Jayaram. *When to Use (and Not to Use) MongoDB - DZone Database*. 2020. URL: https://dzone.com/articles/why-mongodb.

[4] *Voice Kit*. URL: https://aiyprojects.withgoogle.com/voice/.

Appendix A1

Structure of the Workspace