Bachelor Thesis

Voice-Enabled Smart Home Modules

Author:
Josef Šanda

Supervisor:
Ing. Martin Bulín, MSc.

A thesis submitted in fulfillment of the requirements for the degree of Bachelor (Bc.)

in the

Department of Cybernetics

May 23, 2021

Declaration of Authorship

I, Josef Šanda, declare that this thesis titled, “Voice-Enabled Smart Home Modules” 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:

“Intellectuals solve problems, geniuses prevent them.”

Albert Einstein

UNIVERSITY OF WEST BOHEMIA

Abstract

Faculty of Applied Sciences

Department of Cybernetics

Bachelor (Bc.)

Voice-Enabled Smart Home Modules

by Josef Šanda

The bachelor thesis aims to construct several electrical circuits of ESP8266, sensors and lights and develop voice-enabled smart home modules. The first step design a hardware solution and the physical implementation of individual sensors and lights with ESP8266. As next step are ESP8266 programmed and built engine as a software environment for data communication, data storage and voice-enabled modules. Furthermore, a VoiceKit is connected to the engine to allow a user to control modules by voice. The central part of the project is the web page visualizing all states of components, listed voice commands, currently uttered user commands, and graph data from sensors. The outcome of this project is a functional example use of modules in a basic smart home.

Acknowledgements

Throughout the writing of this thesis, I have received a great deal of support and assistance.

I would first like to thank my supervisor, Ing. Martin Bulín, MSc., whose expertise was invaluable in formulating the thesis questions and methodology. Your insightful feedback pushed me to sharpen my thinking and brought my work to a higher level.

I would also like to thank my tutor, Ing. Jan Švec, Ph.D, for sharing his ideas and willing comments whenever I asked for them. You provided me with the tools that I needed to choose the right direction and successfully complete my thesis.

In addition, I would like to thank my parents for their continuous support of my studies, wise counsel and sympathetic ear. You are always there for me.

Contents

List of Figures

List of Tables

List of Algorithms and Code Parts

List of Abbreviations

ASR	**A**utomatic **S**peech **R**ecognition
CLI	**C**ommand-**L**ine **I**nterface
HMM	**H**idden **M**arkov **M**odels
HTTP	**H**yper**t**ext **T**ransfer **P**rotocol
JSON	**J**ava**S**cript **O**bject **N**otation
MQTT	**M**essage **Q**ueuing **T**elemetry **T**ransport
TTS	**T**ext to **S**peech
REST	**RE**presentational **S**tate **T**ransfer

Chapter 1

Introduction

The use of modern technologies in the home opens up new possibilities for managing the user's time and efficient energy management. The smart home responds to the user's needs and uses automatic control to increase the comfort of living in the home. The smart home centre is a complex system with various separate modules, which collects locally measured data and, based on this data and active interventions from the user, automates operation in the home. The features of a smart home include the ability for the user to interact with the system easily. This is often addressed through a website or voice interaction. This project focuses on one aspect of the smart home - enabling the user to built modules and control them by voice.

The voice-enabled modules help the user control the smart home more easily, and the entire solution's comfort is increased. Thus, the user can minimize the energy expended on operating a smart home, where he simply says his command or question and gets answers in a voice with the system's action, where he does not have to go anywhere or even interrupt his work.

The work aims are to build a modular functional system with voice-enabled modules for the smart home, which fulfil specific real applications. The project consists of several development boards connected to several lights and sensors measuring different physical quantities. These control and analytical components are connected to a central modular functional system. An essential part of this thesis is creating voice-enabled modules implement that to the central control system and their connection to a virtual assistant. By using this virtual assistant, the user is able to control the function modules by voice. The entire project is accessible to the user via a web interface, which clearly provides all available information.

1.1 State of the Art

The most famous intelligent personal assistants include Alexa, Siri, Google Assistant. These virtual assistants work on a very similar principle as follows. The assistant constantly listens to its surroundings to see if a wake-up word has been spoken. Assistant process this analysis of wake-up word on its hardware. After saying the wake-up word,

the assistant starts recording a sound and analyzes simultaneously if no one is talking anymore. This recorded sound then assistant send to the appropriate servers for processing. The server handles the relevant device or service according to the processed user command and sends a synthesized audio response back to the assistant.

Siri's first assistant was created by Apple in 2010 and shortly followed by Cortana by Microsoft in 2013 and Alexa by Amazon in 2014. The growing power of computers and advancing cloud technology allows scientists and software engineers to train voice assistants more easily. Over time, voice assistants can respond to the user more naturally and give the user the feeling of talking to a person.

In addition to these tasks, the user can connect the voice assistant to web services (see Fig. 1.1) like Tasker, IFTTT and other features (often called "skills") developed by third-party developers. By these additions, the user adds a new palette of commands such as automating social media posts, ordering a usual drink from a local Starbucks or summoning an Uber or Lyft using connected account data.

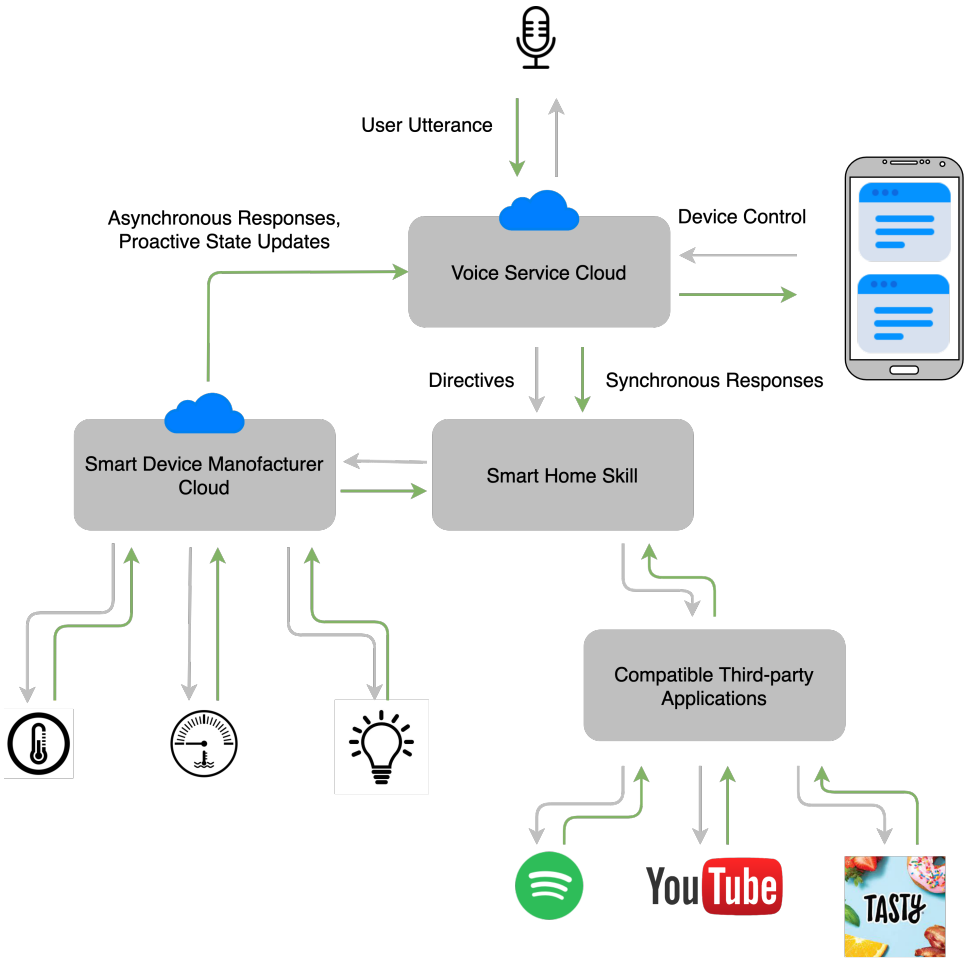


Figure 1.1: Connection schema of voice assistant service

Although each currently available voice assistant has unique features, they share some similarities and are able to perform the following basic

tasks [6]:

- send and read text messages, make phone calls, and send and read email messages;
- answer basic informational queries (“What time is it? What’s the weather forecast? How many ounces are in a cup?”);
- set timers, alarms, and calendar entries;
- set reminders, make lists, and do basic math calculations;
- control media playback from connected services such as Amazon, Google Play, iTunes, Pandora, Netflix, and Spotify;
- control Internet-of-Things-enabled devices such as thermostats, lights, alarms, and locks; and
- tell jokes and stories.

1.1.1 Comparison of SotA Assistant

Because each company develop its voice assistant independently and protect its knowledge, these assistants are quite different despite their common ground. Figure Fig. 1.2 determine the most capable assistant by asking 800 questions that consist of categories like [7]:

- Local – Where is the nearest coffee shop?
- Commerce – Order me more paper towels.
- Navigation – How do I get to Uptown on the bus?
- Information – Who do the Twins play tonight?
- Command – Remind me to call Jerome at 2 pm today.

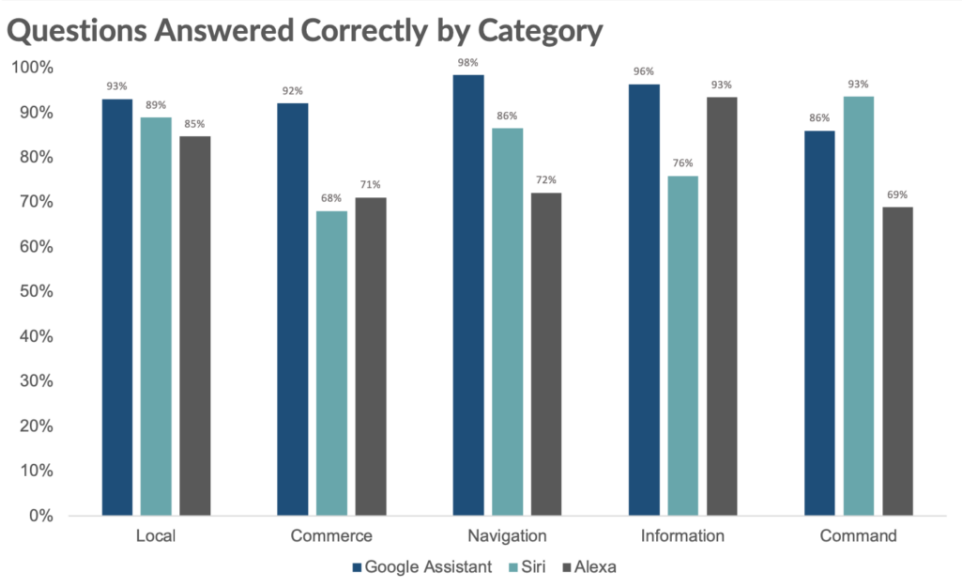


Figure 1.2: Voice assistant comparison by types of questions

Google Assistant has answered 93% correctly and has understood all 800 questions correctly. Siri has been next, has answered 83% correctly and has misunderstood only two questions. Alexa has answered 80% correctly and has misunderstood only one. According to the data shown in Fig. 1.2, Google Assistant has better results overall but lacks in the command category. Amazon Alexa has excellent results only in the information category, where it climbs just below the results of Google Assistant. Siri is brilliant in the command category for such functions as a calling, sending SMS or playing music.

If several users occupy the room, each voice assistant has its way of handling this situation. For example, Amazon Alexa and Google Assistant create multiple voice profiles, which allows the user to train the assistant to recognize his voice specifically and therefore offer different data and use separate accounts for services. This is a very complex task, and no one can cope with it at the desired level.

1.2 The market gap

This project responds to the gap in the market given by the following factors:

- *Dialogue in the Czech Language*: The Czech language is a significant gap in the market in the virtual assistant sector. This gap in the market is caused by the number of people who speak this language. Because the development of this technology is still not complete and costs much money, this low demand market is not interesting for large institutions building virtual assistants.

  Another reason why large technological institutions are not interested in developing virtual assistants for Czech-speaking people is the grammatical and verbal complexity of the Czech language. The Czech language is much more complicated than English. It has many declension, gradation of adjectives, words depending on the sentence and different conjugation.

  So far, known activities in this market are that Siri does not speak in the Czech language, and there are no publicly known prospects that she would be able to do shortly. Google Assistant does not offer to speak in the Czech language either, but it has TTS and ASR support in the Czech language.

- *Modularity of Virtual Assistants*: Although this problem has been largely resolved, as described in Section 1.1, the user is still widely limited and cannot ask the virtual assistant anything he would like. However, technology companies are working hard to resolve this issue to become more and more an issue of the past overtime.

- *Open-source Projects*: There are several open-source projects on the market, such as openHAB, Home Assistant and OpenMotics.

However, setting them up is complicated and requires much technical knowledge. These projects already have a relatively large community and have many packages and modules to connect.

1.3 Thesis Objectives

The objectives of this study are:

1. to test selected modules for use in the project;
2. to implement hardware-dependent modules physically;
3. to design an interface for communication between the user and selected modules;
4. to add an interface for voice interaction;
5. to enrich the project with other modules according to the time possibilities.

1.4 Thesis Outline

This thesis consists of 7 chapters following the standard skeleton of scientific publications.

Chapter 1 describes the current trends of virtual assistants and compares them. It then explains the gap in the market that this project meets and its advantages over other solutions.

Chapter 2 lists the general methods used in automatic speech recognition and automatic speech synthesis fields and briefly describes them. Furthermore, the chapter briefly describes the functions and architecture of the SpeechCloud used as ASR and TTS interface for the project.

Chapter 3 discusses in more detail the architecture used for the system running on the server. The chapter further specifies the type of database used and its benefits for the project. Furthermore, three used communication protocols and their benefits for the project are listed here. At the end of this chapter, we describe the three controllers users use to communicate with the modules and how they work or when they are used.

Chapter 4 lists five created modules. The chapter describes its functions, a code diagram of the functions, the voice commands triggering the functions (a detailed list of the voice commands is then given in appendix A2), used electrical circuits and message structures in more detail for each module.

Chapter 5 shows and describes the created web controller. The chapter defines the used libraries, frameworks and server address. It also describes all the available features and how the data on the page changes over time.

Chapter 6 comes with the discussion about the results. It also contains a comparison of the developed system to the presented state-of-the-art technology from section 1.1. Then, ideas for future work are suggested. The study is concluded in Chapter 7.

Appendix A1 contains a figure of algorithm diagram of the ESP that did not fit the main text but can still be interesting for some readers. As mentioned above, appendix A2 gives a detailed list of voice commands for each module described in chapter 4.

Chapter 2

Dialogue Systems

2.1 Automatic Speech Recognition

Automatic Speech Recognition (ASR) is a way of converting sound into text.

Sound is nothing more than vibrations of the air that we humans are trained exceptionally well to decode. Moreover, now, we are teaching our computers how to do this. In the beginning, we have a stream of words that a person has uttered. The sound is picked up by a microphone and converted to a digital signal through a sound card, which means a stream of ones and zeroes.

One of the possible approaches in ASR modelling is, for example, at the level of phonemes or the level of whole words. We will only give an example here at the phoneme level, as the other approaches are very similar.

The first step the ASR system do is process the sound. It steps the sound to have chunks of speech that can be worked with and that can be mapped to letters. These chunks are called phones.

The part of ASR responsible for mapping sound to phones is called the *acoustic model* as a set of building blocks, boxes which contain models for all phones in a given language as showing in Fig. 2.1.

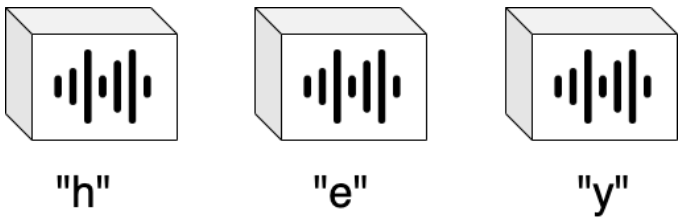


Figure 2.1: Phones boxes

There are boxes labelled, for example, A, B, C, depending on which phones are used in the particular language. On top of that, part of this construction set is also contextual probabilities. It means how likely a phone is to follow another. The acoustic model's task is to guess which phones have been pronounced and how they combine into a word. The

acoustic model processes the sound and compares it to the models of individual phones from its boxes. Since speech is very complex in a real scenario, the chunks that a person uttered will be similar to more than one box. The acoustic model takes this into account and also looks at the neighbouring chunks and their contextual probabilities. For example, in the string "HELLO", the second phoneme that a person uttered might have been E. However, it also could have been ə, A or even I, with different degrees of certainty. The next phoneme is probably L, but it also could be R. There are different probabilities of these phones in context, for example, H followed by E is more likely, at least in English, than H followed by I. The ASR system combines these bits of information and outputs the most likely result - a string of phones.[11]

The next step is to convert it into words. Nevertheless, this part can be tricky because the ASR does not know when a word starts or ends. Contrary to popular belief, there are no pauses between words in fluent speech. This particular string "heloumaj..." of phones can constitute several different phrases, for example, "hell oh my nay miss" or "hello mine aim is", or "hello my name is". The part of ASR responsible for mapping phones to words and phrases is called the language model.

Hidden Markov Models (HMM) are widely used for the statistical approach for automatic speech recognition. Suppose that $W = \{w_1, w_2, \ldots, w_N\}$ is a sequence of words, and $O = \{o_1, o_2, \ldots, o_N\}$ is a sequence of phones. These sequences are taken with a period of 10 ms for segments of speech of length from 20 to 40 ms. The Bayes Theorem for conditional probability is used to figure out which phones have been pronounced and how they combine into a word.

$$\boldsymbol{W}' = \underset{w}{\operatorname{argmax}} P(\boldsymbol{W} \mid \boldsymbol{O}) = \underset{w}{\operatorname{argmax}} \frac{P(\boldsymbol{W})P(\boldsymbol{O} \mid \boldsymbol{W})}{P(\boldsymbol{O})} \tag{2.1}$$

where P(W) is the a priori probability of word W, P(O|W) is the probability that the sequence of phones O will be generated under the conditions of pronouncing the sequence of words W, P(O) is the a priori probability of the sequence of phones O.

Since the probability P(O) is independent of the sequence of words W, it is possible to modify the equation into the form shown in Fig. 2.2:

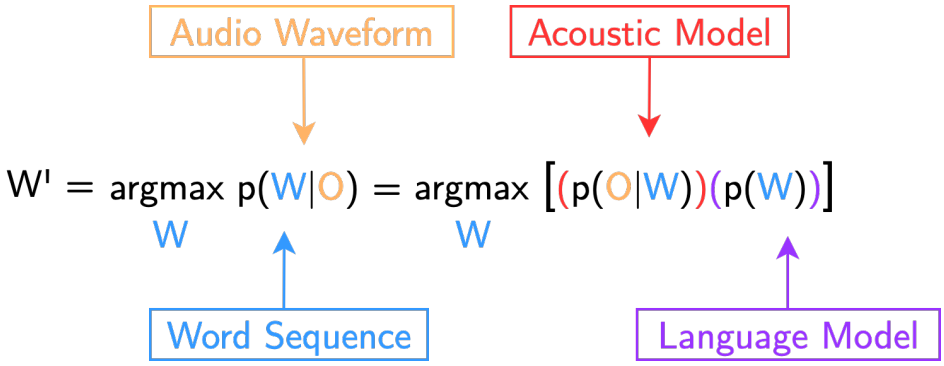


Figure 2.2: The relation among acoustic model, language model and Bayes theorem

Nowadays, HMM is no longer used much, and Neural Networks techniques have become more common.

2.2 Automatic Speech Synthesis

The task of generating a speech out of text information has originally two approaches:

1. concatenative (unit selection);
2. statistical parametric.

The concatenative synthesis is based on sequential combining of shot prerecorded samples of the speech. These samples can be stored in a database as of whole sentences, phrases, words and different phonemes. It depends on the application of the solutions. Building the unit selections synthesis model consists of three steps:

1. Recording of the whole selected speech units in no possible context.
2. Labelling segmentation of units.
3. Choosing the most appropriate units.

The concatenative method is the most straightforward approach to the speech generation. Disadvantages include the requirement to have an ample storage for recorded units and an inability to apply various changes to a voice.

The statistical parametric synthesis consists of two parts, as shown in Fig. 2.3. The training step's approach is to extract excitation parameters like fundamental frequency and dynamic features, and spectral parameters from the speech database. Then we estimate them using one of the statistical models. The Hidden Markov Model (HMM) is the most widely used for this task. It should be noted that HMM is conscious dependent. It means that in this step, in addition to phonetic context, linguistic and prosodic context is taken into account. In the synthesis part, at first given sentence is converted into points with a dependent

label sequence, and then their chance HMM is constructed according to this sequence. Next, spectrum and excitation parameters are generated from the utterance HMM, and finally, speech waveforms are synthesized from these parameters using excitation generation and the speech synthesis filter. The advantages of the statistical parametric approach are:

1. Small footprint
2. No need to store the speech waveforms, only statistics language independence.
3. Flexibility in changing voice characteristics speaking styles and emotions.

The most noticeable drawback is the quality of a synthesized speech.

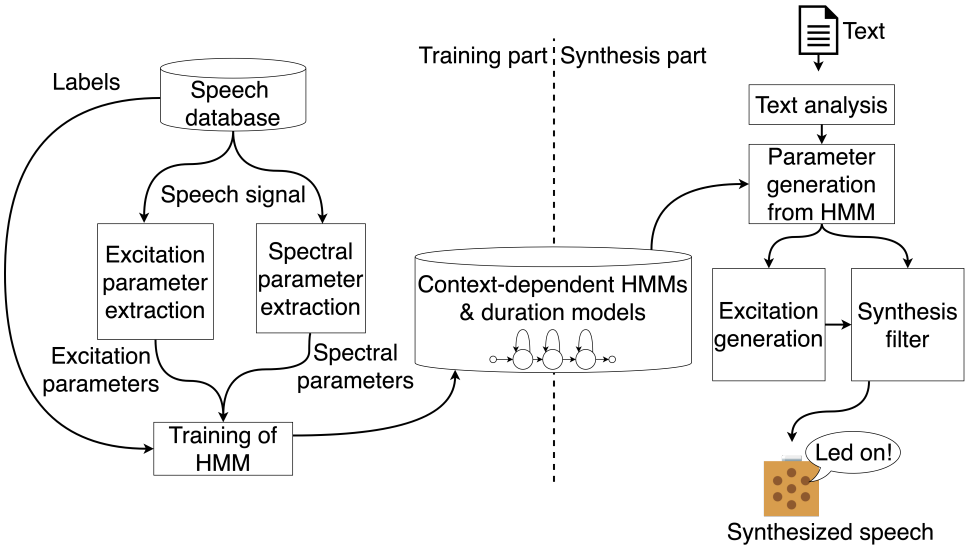


Figure 2.3: Statistical parametric speech synthesis [1]

Procedures have changed over time, and Neural Networks (NN) and Long-Term Memory (TSLM) techniques have become more common in statistical parametric synthesis.

2.3 The SpeechCloud Platform

The SpeechCloud platform, developed at the Department of Cybernetics of the University of West Bohemia, is a system that connects ASR and TTS systems operating together via one interface. It is then possible to use these systems by many applications simultaneously through this interface. An independent instance is created for each dialogue system, allowing a client to create a characteristic language model, send a speech record to recognize, and receive the synthesized speech.

SpeechCloud provides the same services to all clients unless limited or specified otherwise. Each client should have the same functions, but each device, experiment or project is separated from the others, so the results are not affected by the unwanted intervention.

The architecture of the SpeechCloud and the connection to the client is briefly visualized in Fig. 2.4. The SpeechCloud using the module SCAPIServer as a primary point to establish a connection with the client application. Thus, the module negotiates with the client a specific application configuration, a control communication channel and the authentication of the session. The SCAPIServer then provides these pieces of information to other modules. The SIPSwitch module mediates the audio data transfer service between the SCWorker component and the client application. One instance of the SCWorker component is reserved for each client that holds one ASR and TTS instance. The SCWorker component has access to a TCP/IP network connection to collect additional data sources.

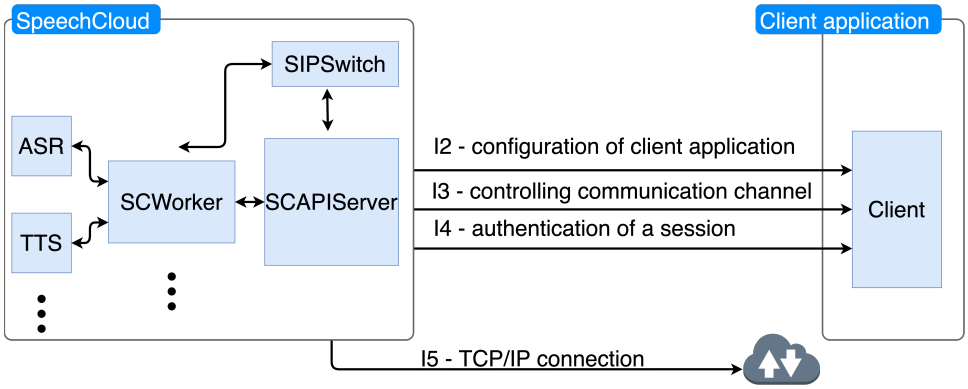


Figure 2.4: SpeechCloud schema

Solving the subject of the connection and transmission of data to the SpeechCloud via Internet communication protocols is not the content of this work hence are used ready-made software components and the SpeechCloud platform is used as a service.

Chapter 3

Backend

Own engine running on Raspberry Pi 4 has been developed and serves as the backend for the project. The whole engine is coded in Python, and adheres to the following principles:

- *Simplicity*: write a straightforward code that is easily understandable for later rewriting.
- *Modifiability*: write a code with the ability to admit changes due to a new requirement or detect an error that needs to be fixed.
- *Modularity*: write a well-encapsulated code of modules, which do particular, well-documented functions.
- *Robustness*: write a code focusing on handling unexpected termination and unexpected actions.

3.1 Diagram Description

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

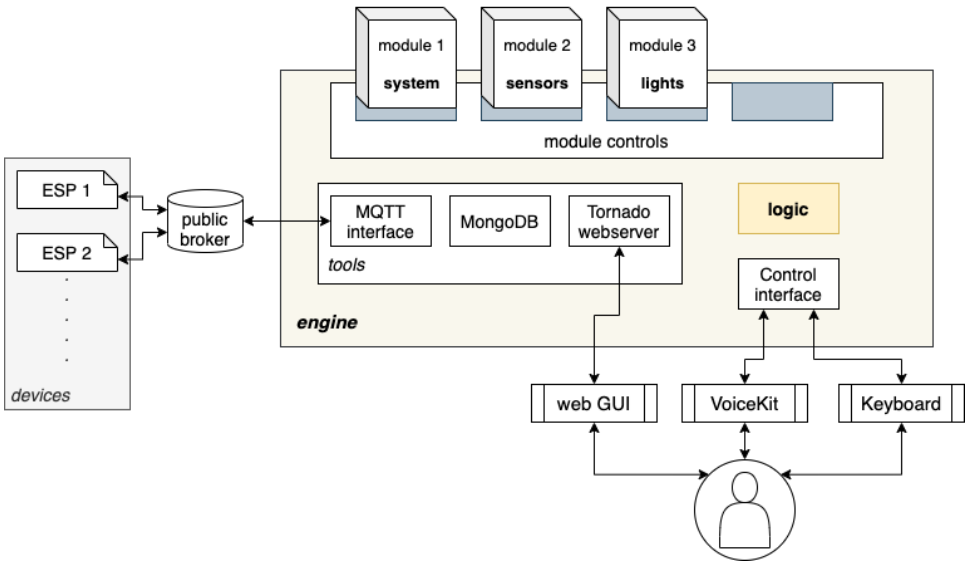


Figure 3.1: Project architecture

Engine uses tools like MQTT, MongoDB, Tornado web server that is 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.

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

The engine also contains a separate block for *logic*. This block captures a command from the VoiceKit 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.

The block has basic *logic*, which searches for predefined commands (loaded together with modules from file "`voicehome\modules\<module_name>\metadata.json`") and looks for whether the uttered command contains all the words from the predefined command. If the block finds the match, then the function name and module name send to the engine to be executed. If it does not find the voice command in lists, it replies that the command has not found with the recognized command.

The block is written in a way that it is easy to change this *logic* at any time, and it is up to the developers to best deal with this complex issue of search.

3.2 Database

MongoDB is an open-source document database built upon a NoSQL database and written in C++. Database's horizontal, scale-out architecture 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.[10]

Before using this type of database, we have to be familiar with different terminology compare to 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:

- MongoDB is easy to scale.
- Schema-less database: we do not need to design the database's schema because the code we write defines the schema, thus saves much time.
- The document query language supported by MongoDB is simplistic as compared to SQL queries.
- There is no need for mapping application's objects to database's objects in MongoDB.
- No complex joins are needed in MongoDB. There is no relationship among data in MongoDB.
- Because of using JSON[1] format to store data, it is effortless to store arrays and objects.
- MongoDB is free to use. There is no cost for it.
- MongoDB is simple to set up and install.

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

Communication is the backbone of the whole project among several devices over the internet. Therefore, it had to be found robust, scalable, and cost-effective protocols that transmit messages and data securely.Based on the survey, we choose 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 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. [4]

[1] **J**ava**S**cript **O**bject **N**otation

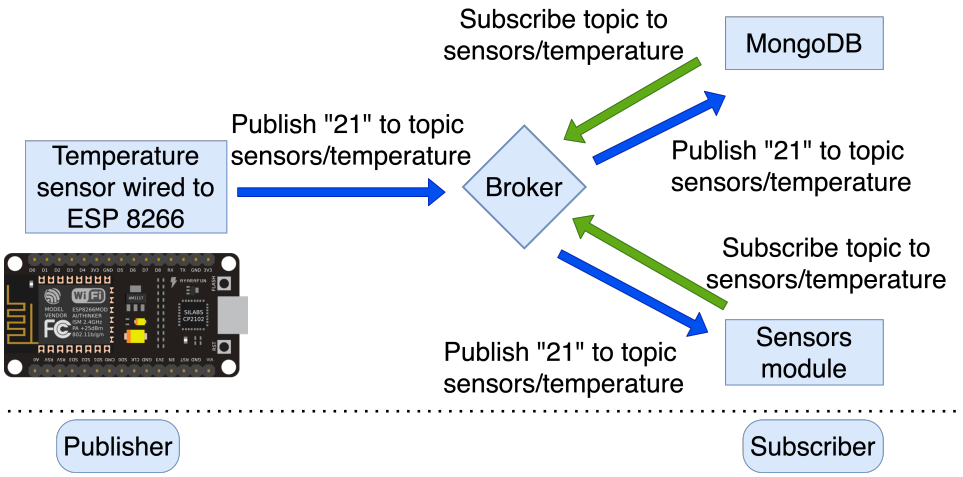


Figure 3.2: MQTT publisher/subscriber pattern

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 determines errors by TCP and orchestrates 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 exist. The decoupling give 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.

When the publisher sends his message, it is handled by the broker who filters all incoming messages and distributes them to accredited subscribers. The filtering is based on topic or subject, content and type.

In the case of MQTT, the filtering is subject-based and therefore, every message including a subject or a topic. The client subscribes to the topics he is interested in, and the broker distributes the messages accordingly as shown in Fig. 3.3.

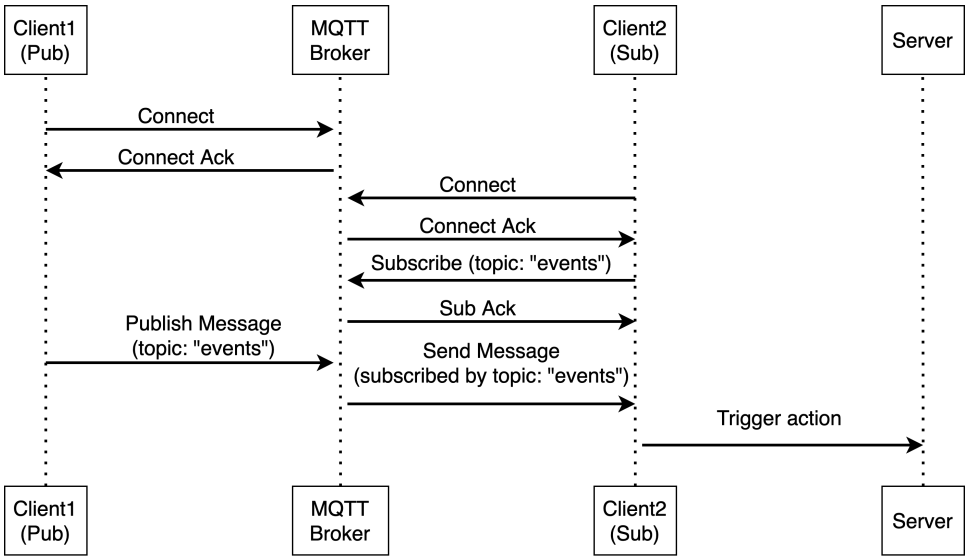


Figure 3.3: Diagram illustrating how communication in MQTT flow.

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, for example, 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

In this work, WebSockets are used to provide communication between the client and the engine. WebSocket 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 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.[3]

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[2], AMQP[3], 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 VoiceKit, 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 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.

[2] E**x**tensible **M**essaging and **P**resence **P**rotocol - messaging and presence protocol based on XML and mainly used in a near-real-time exchange of structured data.

[3] **A**dvanced **M**essage **Q**ueuing **P**rotocol - an open standard application layer protocol for message-oriented middleware.

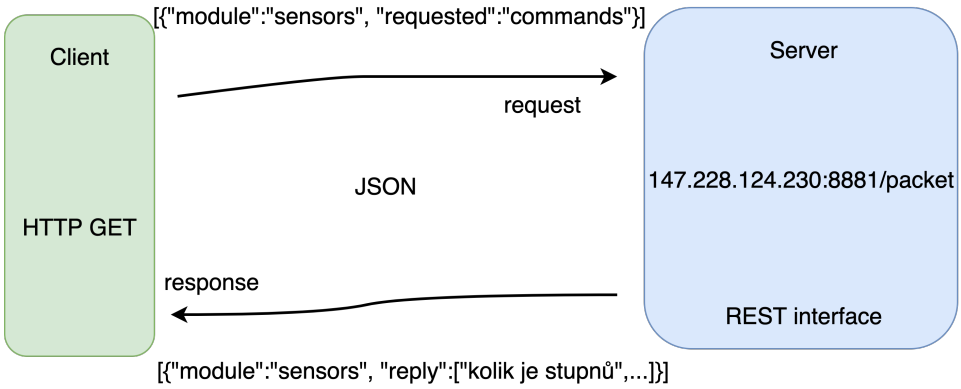


Figure 3.4: REST principle

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 built upon a voicehome controller. The developer can quickly type a voice command with high accuracy through the command-line and debug the command thoroughly in various forms.

3.4.2 VoiceKit

VoiceKit (see Fig. 3.5 is a building kit made by Google [12] 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: Photo of the assembled VoiceKit [9]

Fig. 3.6 show a diagram of messages flows during a conversation. It is evident from the diagram that all communication with a user and the SpeechCloud mediate VoiceKit thus engine can manipulate just with a text.

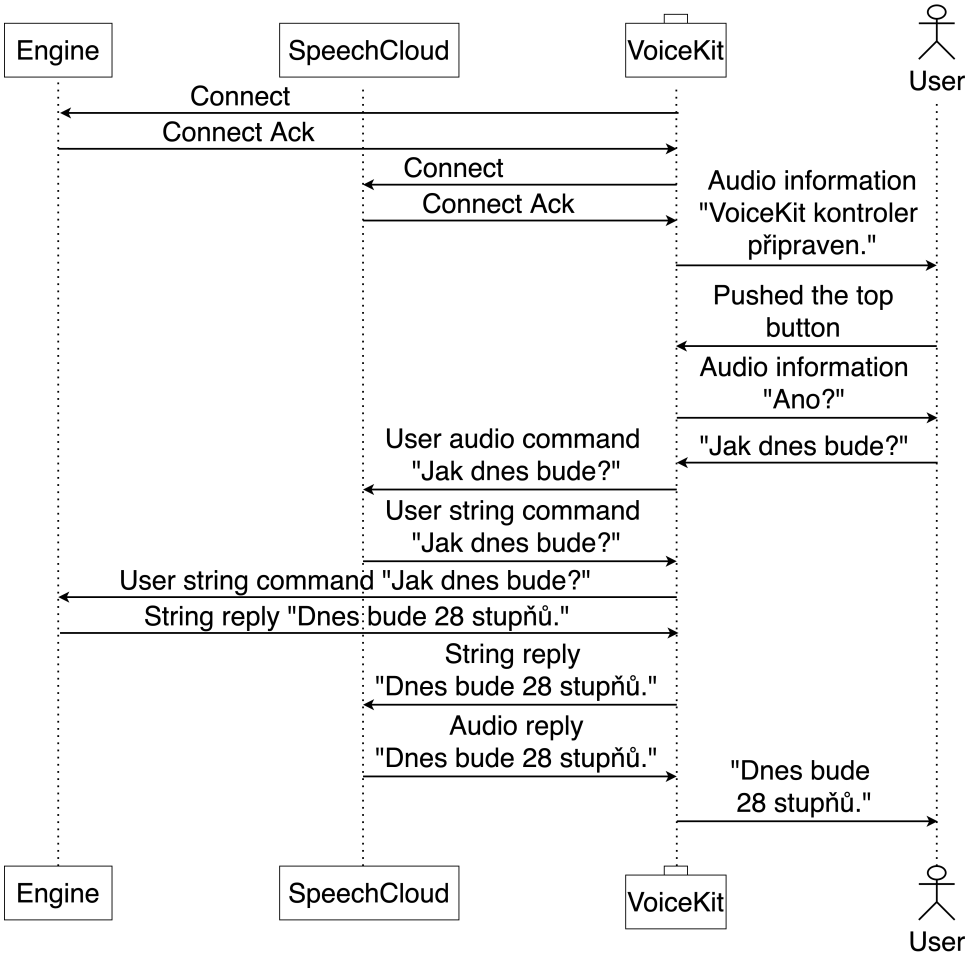


Figure 3.6: Diagram of messages flows during a conversation

3.4.3 Website

The second interface next to the already mentioned Voice Kit is a website. The web server is implemented in Python using 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.

The website communicates with the engine by the already mentioned WebSocket. Figure 3.7 show an example of communication between the

web site and the ESP to turn on an onboard led. Communication uses JSON as a data format and is evident from the figure that web site and engine use for communicating protocol WebSocket, whereas ESP and engine use MQTT.

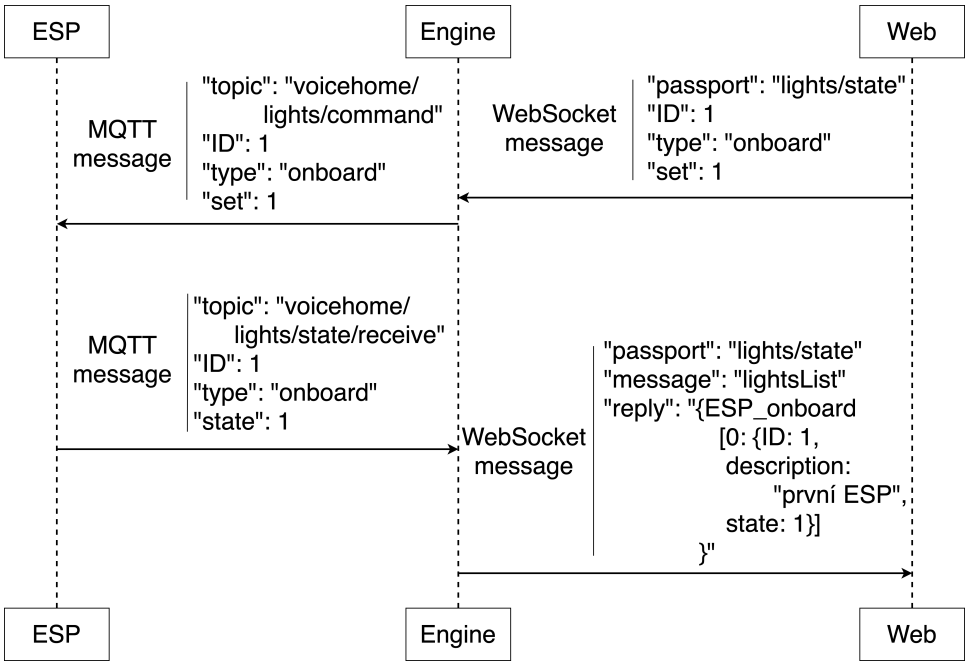


Figure 3.7: Diagram of message flows to turn on/off led on ESP by the website.

Chapter 4

Modules

Modules are well-encapsulated code written to provide functional and control elements (moves) above home to the user. Each module inherits from VoicehomeModule class that provide communicating interface to each module by WebSocket, MQTT and mediate writing and reading from MongoDB. Each module defines its topic for MQTT and a passport for WebSocket that subscribe from these services. Messages containing these topics or passports are passed through the engine to the modules.

Thus each module has to be created by the following approach:

1) Create a new folder in `"voicehome\modules\"`, the name of this folder is the module's name.
2) Create two mandatory files
   (a) `"voicehome\modules\<module_name>\metadata.json"` that include object with following variables:
      - `"module_id"`: contain the name of the module
      - `"description"`: contain a string with a brief description of the module
      - `"mqtt_topics"`: contain a list of MQTT topics module wants to subscribe
      - `"websocket_passports"`: contain a list of WebSocket passports passing messages to the module
      - `"moves"`: list of objects that define moves this module is capable of
        - `"move_id"`: a unique ID that follows the convention `<module_name>_<order_in_this_list>`
        - `"method_name"`: contain the name of a Python function in `<module_name>.py` called when this move is activated
        - `"description"`: contain a brief description of the move

- `"calls"`: list of calls (voice commands to VoiceKit) activating this move; it is a list of lists of words (must fit the chosen logic algorithm)

(b) `"voicehome/modules/<module_name>/<module_name>.py"` that define class of module. This class have to inherit from VoicehomeModule and include all methods listed in the previous file `metadata.json`.

```
1 from modules.voicehome_module import VoicehomeModule
2
3
4 class Module_name (VoicehomeModule):
5
6     def __init__(self, engine, dir_path):
7         VoicehomeModule.__init__(self, engine,
              dir_path)
```

Part of Code 4.1: Template for creating a new module

After accomplishing these requirements, it is not necessary to restart the entire engine but can simply use the voice command `"načti moduly"` from the System module. Each particular module can be turned off or on using the web interface in the Modules tab, which is specified in more detail in Section 5.3.

4.1 Lights

The system module provides the user commands to control lights by voice. The user not only turns on, off or blinks lights but can also identify the development boards by lighting an onboard LED on a specific board. The module keeps in memory a list of all lights with their current status and detailed description.

The onboard LEDs are mounted on the board from the factory on pin 2. The other lights have their specific wiring, but one LED is prepared for demonstration purposes, which by our definition is located in the living room and is wired according to the diagram in Fig. 4.1.

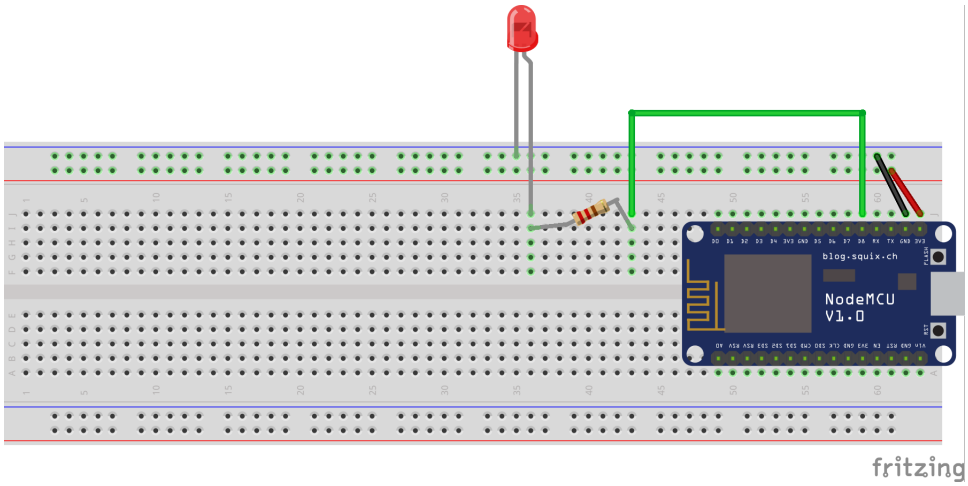


Figure 4.1: LED "living room" wiring diagram

The diagram (see Fig. 4.2) shows the steps for turning on/off light in a simplified way. This process is the same for all types of lights. Only an ID and type of light is different in a message. The ESP development board have this ID connect to the pin of the light in its configuration file.

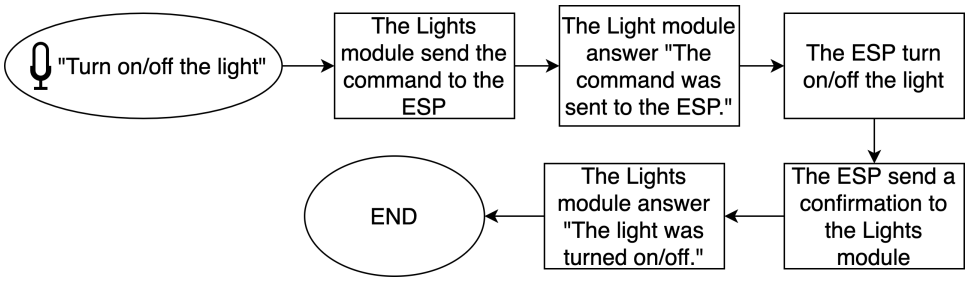


Figure 4.2: Diagram of a process turning on a light

4.1.1 Implemented functions

This subsection show table of 11 functions implemented in the *Lights* module. The number of calls (commands) registered in the language model is summed up in Table 4.1. A complete list of all possible commands is provided in appendix A2.1.

ID	Description of the function	# calls
1	Turn on all the onboard LEDs.	2
2	Turn off all the onboard LEDs.	2
3	Turn on the light 1	4
4	Turn off the light 1	4
5	Turn on the onboard LED number 1	2
6	Turn off the onboard LED number 1	2
7	Turn on the onboard LED number 2	2
8	Turn off the onboard LED number 2	2
9	Turn on the onboard LED number 3	2
10	Turn off the onboard LED number 3	2
11	Voicekit answer which lights are turned on	1

Table 4.1: Implemented functions of the *Lights* module (for detail see appendix A2.1)

4.1.2 Messages Structure

The engine uses the following topics and messages for maintaining lights:

- "voicehome/lights/command" - to turn the light on/off

```
{
    "ID": integer,
    "type": string,
    "set": integer
}
```

Part of Code 4.2: Structure of JSON message to turn on/off the light in module *Lights*

- "voicehome/lights/state/command" - to ask the light for state

```
{
    "ID": integer,
    "type": string
}
```

Part of Code 4.3: Structure of JSON message to asking for the state of the light in module *Lights*

- "voicehome/lights/state/receive" - to receive state of the light

```
{
    "type": string,
    "state": integer,
    "ID": integer
}
```

Part of Code 4.4: Structure of JSON message to receive state of the light in module *Lights*

4.2 Sensors

The sensors module provides the user commands to communicate directly with sensors wired to the ESP development board or ask for statistics information such as average. A diagram of the algorithm of the ESP development board is placed in appendix A1. The sensors connected to the module are:

- sensor measuring temperature, humidity and pressure bme280 (see Fig. 4.4);
- sensor measuring temperature ds18b20 (see Fig. 4.5);
- sensor measuring illuminance tsl2591 (see Fig. 4.6).

The diagram (see Fig. 4.3) shows the steps for measuring current temperature in a simplified way. Other current measurements are very similar to temperature measurements, and only the measured data are from another sensor.

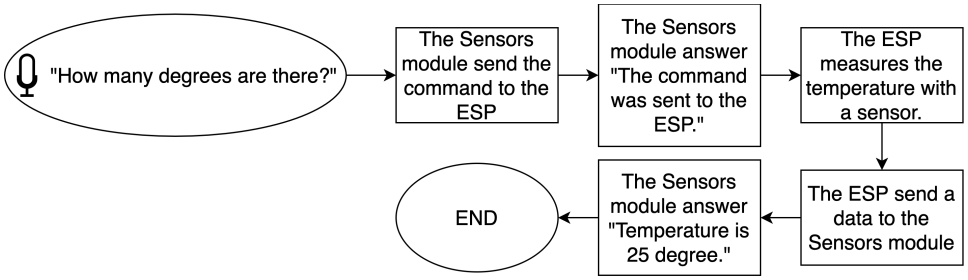


Figure 4.3: Diagram of a process measuring current temperature

The functions to determine average use the Python library to obtain past data from MongoDB. The functions then calculate the average on the engine side and send it to the VoiceKit to answer.

4.2.1 Implemented functions

This subsection show table of 12 functions implemented in the *Sensors* module. The number of calls (commands) registered in the language model is summed up in Table 4.2. A complete list of all possible commands is provided in appendix A2.2.

ID	Description of the function	# calls
1	Sends a command via MQTT to measure current temperature	3
2	Sends a command via MQTT to measure current pressure	3
3	Sends a command via MQTT to measure current humidity	3
4	Sends a command via MQTT to measure current illuminance	3
5	Voicekit answer average temperature for the last day	2
6	Voicekit answer average pressure for the last day	2
7	Voicekit answer average humidity for the last day	2
8	Voicekit answer average illuminance for the last day	2
9	Voicekit answer average temperature for the last week	2
10	Voicekit answer average pressure for the last week	2
11	Voicekit answer average humidity for the last week	2
12	Voicekit answer average illuminance for the last week	2

Table 4.2: Implemented functions of the Sensors module (for detail see appendix A2.2)

4.2.2 Messages Structure

The engine uses the following topics and messages for maintaining sensors:

- "voicehome/sensors/quantity_type" - to receiving data from sensors

```
{
    "location": string,
    "sensor_id": string,
    "state": string,
    "timestamp": string,
    "quantity_units": string,
    "temperature_value": float,
    "quantity_type": string,
    "owner": string
}
```

Part of Code 4.5: Structure of JSON message for receiving data from sensors in module *Sensors*

Example of use for temperature sensor DS18B20

```
{
    "location":"room_2",
    "sensor_id":"ds18b20_1",
    "state":"ok",
    "timestamp":"2021-05-15 03:18:19",
    "quantity_units":"degree",
    "temperature_value":21.5625,
    "quantity_type":"temperature",
    "owner":"jsanda"
}
```

Part of Code 4.6: Structure of JSON message for receiving data from temperature sensor DS18B20 in module *Sensors*

- "voicehome/sensors/command" - to command sensor to measure current data

```
{
    "command": string,
    "sensor_ID": string,
    "quantity_type": string,
    "who_asking": string
}
```

Part of Code 4.7: Structure of JSON message to command sensor to measure current data in module *Sensors*

Example of use for temperature sensor DS18B20

```
{
    "command":"measure_now",
    "sensor_ID":"ds18b20_1",
    "quantity_type":"temperature",
    "who_asking":"voicekit"
}
```

Part of Code 4.8: Structure of JSON message to command sensor DS18B20 to measure current data in module *Sensors*

4.2.3 Pressure Sensor (BME280)

The BME280 [5] module is used to measure the barometric pressure, internal temperature and humidity. The sensor is used mainly for pressure measuring in this thesis. The measuring range of this sensor is in case of ambient temperature from -40 °C to +85 °C, in case of humidity from 0% to 100% and in case of barometric pressure from 300 hPa to 1100 hPa. Temperature values are measured with an accuracy of ±1 °C, humidity values with an accuracy of ±3% and barometric pressure values with an accuracy of ±1 Pa. The sensor is supplied with power from a 3.3 V output on the microchip and grounded to the microchip. The SDA and SCL outputs are wired to the D6 and D5 pins of the microchip to provide I2C communication. The wiring diagram of the BME280 sensor is shown in Fig. 4.4

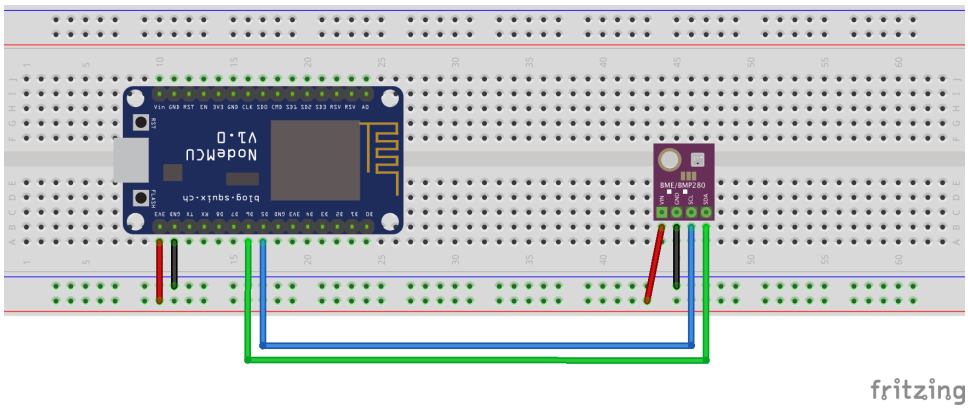


Figure 4.4: BME 280 wiring diagram

4.2.4 Temperature Sensor (DS18B20)

The DS18B20 [8] is a temperature sensor, and its waterproof variant was chosen as an illustration. The module's output is the value of temperature in the degree Celsius unit. The sensor's measuring range is from -55 to +125 degrees with an accuracy of ± 0,5 °C within limits from -10 °C to +85 °C. The sensor communicates via the One-Wire interface and uses one pin for data transmission. The sensor is connected to the microchip by three wires - one wire for power supply 3,3V, one for grounding and one for data communication. It is necessary to connect a resistor of resistance 4.7 kΩ to the circuit, connecting the data and power wires. The wiring diagram of the DS18B20 sensor is shown in Fig. 4.5

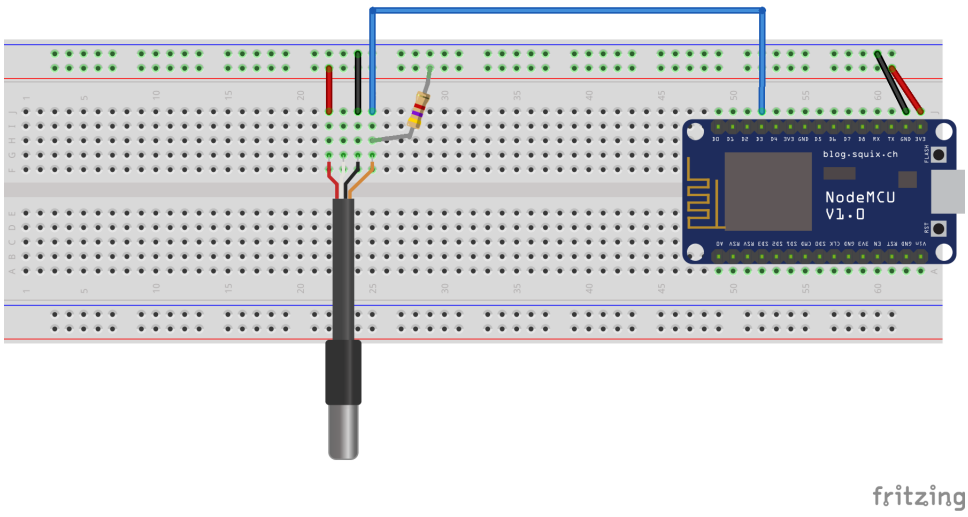


Figure 4.5: DS18B20 wiring diagram

4.2.5 Illuminance Sensor (TSL2591)

The TSL2591 [2] is a light intensity sensor that converts the light intensity into a digital output transmitted by the I2C bus. The module's output is the value of light intensity in the lux unit with an accuracy of 4 decimal places, and the measuring range is from 188 μLux to 88,000 Lux. The value of the sensor is measured to three decimal places, although

it is not necessary to have such high accuracy. The module guarantees measurement accuracy in temperature conditions from -30 °C to +80 °C. The sensor is supplied with power from a 3.3 V output on the microchip and grounded to the microchip. The SDA and SCL outputs are wired to the D1 and D2 pins of the microchip to provide I2C communication. The wiring diagram of the TSL2591 sensor is shown in Fig. 4.6

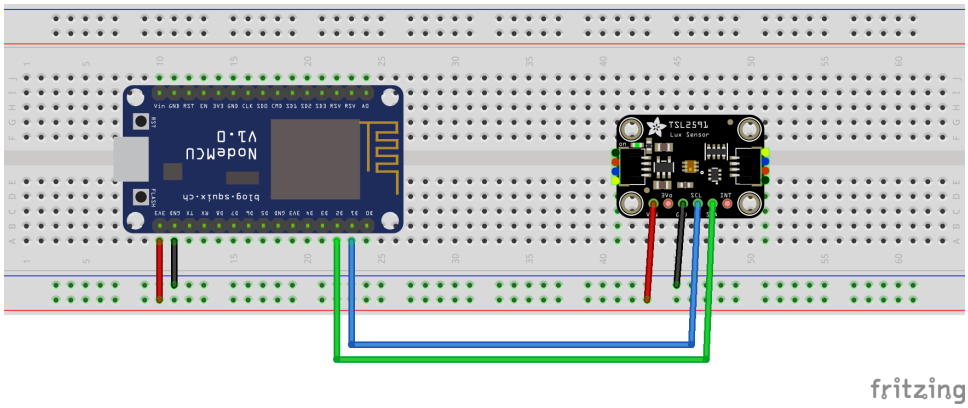


Figure 4.6: TSL2591 wiring diagram

4.3 Time

This module provides commands for manipulation with the time, such as asking for time, date, set timer. The module does not communicate with other devices. The module's functions exploit system information and information available on the Internet. The module uses technique calls web scraping to reach the web page information. The technique runs the webpage and sucks desired pieces of information from it.

The diagram (see Fig. 4.7) shows how the function of the timer work in a simplified way. Other different functions of the time module work on the principle of downloading data from the Internet, and it is not necessary to explain them further.

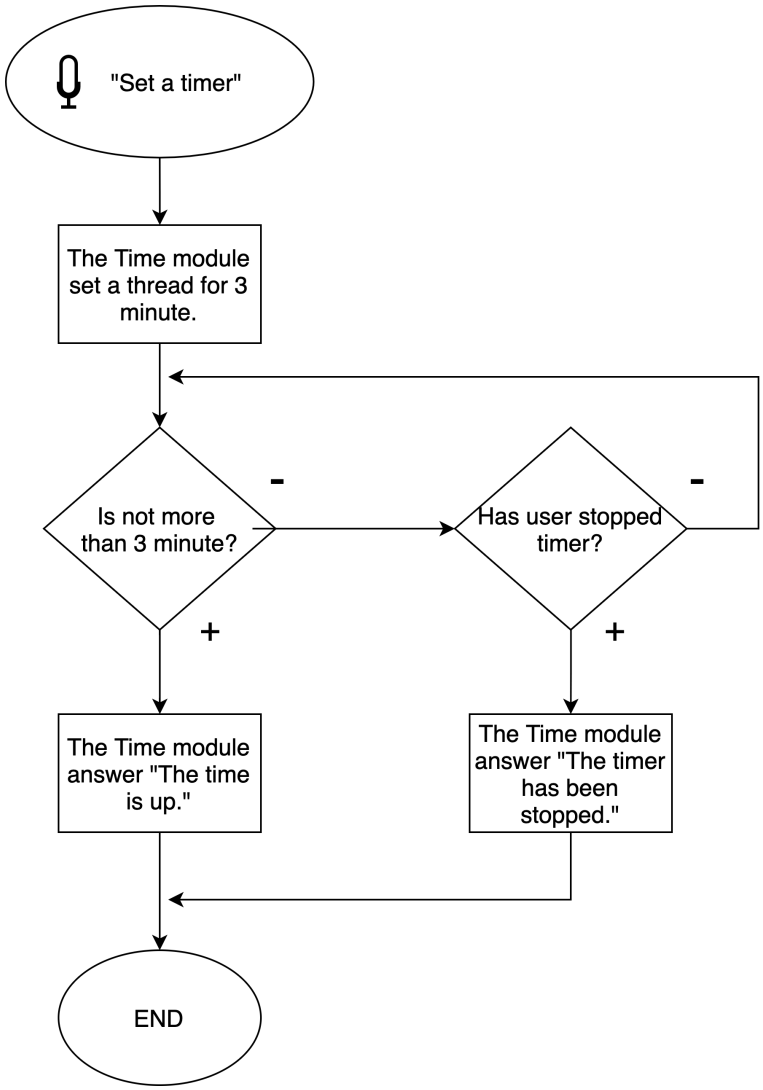


Figure 4.7: Diagram of a timer function

4.3.1 Implemented functions

This subsection show table of 8 functions implemented in the *Time* module. The number of calls (commands) registered in the language model is summed up in Table 4.3. A complete list of all possible commands is provided in appendix A2.3.

ID	Description of the function	# calls
1	Send command to ask the server for the current time	2
2	Send command to ask the server for the current day of year	2
3	Send a command to the server to start timer on 3 minute	1
4	Send a command to the server to stop timer	2
5	Ask the server for today's day of the week	2
6	Ask the server for today's sunrise time	2
7	Ask the server for today's sunset time	2
8	Ask the server for today's nameday	2

Table 4.3: Implemented functions of the Time module (for detail see appendix A2.3)

4.4 System

The system module provides the user commands to test the functionality and adjust some settings of the engine. The module communicates primarily with the engine, but it is possible to establish this communication with other devices. Like other technology, the module uses the MongoDB library from python to test the engine's database.

4.4.1 Implemented functions

This subsection show table of 8 functions implemented in the *System* module. The number of calls (commands) registered in the language model is summed up in Table 4.4. A complete list of all possible commands is provided in appendix A2.4.

ID	Description of the function	# calls
1	Reloads all the modules again. A fresh refresh.	3
2	Makes a testing write and read with the database.	3
3	Makes a testing MQTT publish to voicehome/system/test, which this module is also subscribing	3
4	Sends a testing websocket message with passport system/test.	3

Table 4.4: Implemented functions of the System module (for detail see appendix A2.4)

4.5 Weather

The weather module provides the user commands to answer questions about the weather. The module's functions exploit the information available on the Internet. By preprocessing the information and replacing characters like "°C" to "stupnů" or "-" to "mínus" from the Internet, we can then send fully synthesizable text to SpeechCloud and answer the question to the user. The module uses the same technique as the Time module calls web scraping to reach the webpage information. The technique runs the webpage and sucks desired pieces of information from

it. Preprocessing the information uses the technique regex and essential functions such as finding text and selecting text — a simple example of how regex is used shown in code part 4.1.

The diagram (see Fig. 4.8) shows the steps for answer forecast questions by VoiceKit. The forecast for other days is very similar. The module only searches for a different day of the week on the web page.

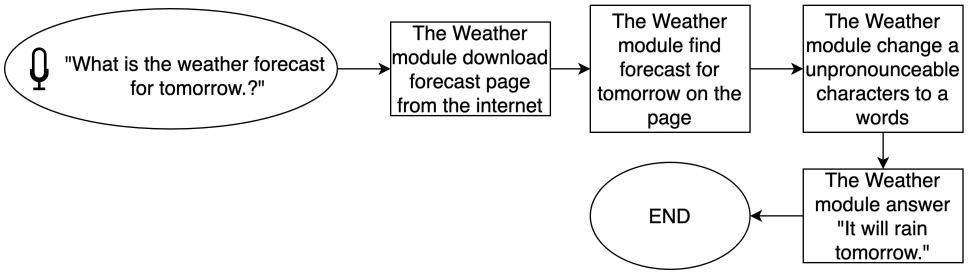


Figure 4.8: Diagram of a process VoiceKit answering forecast

4.5.1 Implemented functions

This subsection show table of 9 functions implemented in the *Weather* module. The number of calls (commands) registered in the language model is summed up in Table 4.5. A complete list of all possible commands is provided in appendix A2.5.

ID	Description of the function	# calls
1	Getting forecast for today from www.chmi.cz.	2
2	Getting forecast for tomorrow from www.chmi.cz.	2
3	Getting forecast for monday from www.chmi.cz if it is up to four days and not today.	2
4	Getting forecast for tuesday from www.chmi.cz if it is up to four days and not today.	2
5	Getting forecast for wednesday from www.chmi.cz if it is up to four days and not today.	2
6	Getting forecast for thursday from www.chmi.cz if it is up to four days and not today.	2
7	Getting forecast for friday from www.chmi.cz if it is up to four days and not today.	2
8	Getting forecast for saturday from www.chmi.cz if it is up to four days and not today.	2
9	Getting forecast for sunday from www.chmi.cz if it is up to four days and not today.	2

Table 4.5: Implemented functions of the Weather module (for detail see appendix A2.5)

Chapter 5

Graphical User Interface

The web was chosen as the visualization environment for its simplicity and the number of possible tools to use. The website is available at the public IP address http://147.228.124.230:8881/. The pages are written in separate HTML files and share the same JavaScript and CSS file. JavaScript is used to provide the necessary dynamics, such as new sensor's data, which display on the page or changes the sensor's state icon. The web design is defined in the CSS file. Styles for HTML elements are stored here, such as font letters, font family, the layout of elements on the page. Combining these three languages was created an interactive web visualization for the presentation of measured data and control of a smart home. Libraries and frameworks used to build the website are Bootstrap, JQuery and Dygraphs.

The structure of the website is divided into subpages Home, Analytics, Modules. The user can move between these subpages using the horizontal menu (see Fig. 5.1), which consists of three buttons with subpages names. The style of this navigation menu is designed using the Bootstrap framework. This menu is changed to a slider from above for more straightforward operation using the touch screen when using a mobile device.

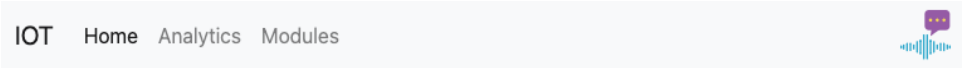


Figure 5.1: The creenshot of the horizontal menu on the web page

When a user opens a web page, all modules, voice commands and currently connected controller are loaded using the REST service. This information is transmitted in JSON format. The connected controller always shows in the navigation menu next to links of subpages.

The Home page allows the user to interact with the light control, view the current status of the sensors, current data from the sensors, the voice commands currently in use, and display the fundamental weather forecast for today and tomorrow. The Analytics page offers the user detailed historical data from sensors after selecting sensors using the switch at the sensors filter section and sending a query using the submit button. The data from the database are displayed on the page below

the sensors filter. The Modules page shows the user all the programmed modules user can use with all its voice commands. It is possible to switch each of these modules on/off using the switch next to the module name.

5.1 Home

The Home page is the default page when the website is visited. At the top of the page there is a menu with links to other pages. The main container with all the information on the page is below the menu. This container is divided into smaller boxes, which will be described below.

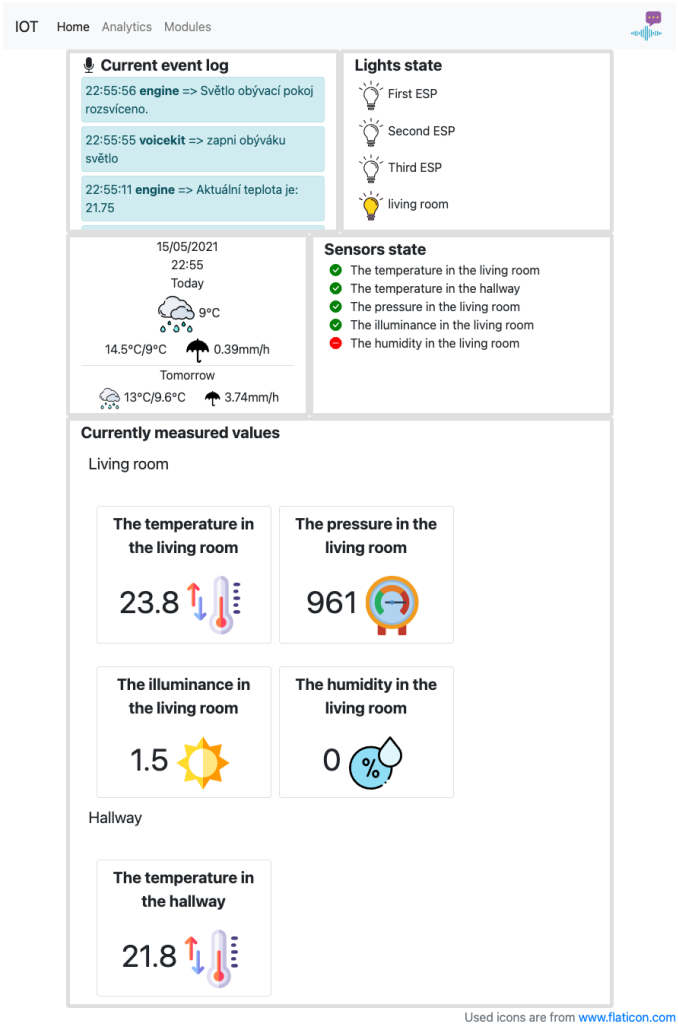


Figure 5.2: The screenshot of the web page on the Home page

The first box here is the Current event log (see Fig. 5.3), the currently spoken voice commands are displayed here. It shows the time of the message and from whom the message is.

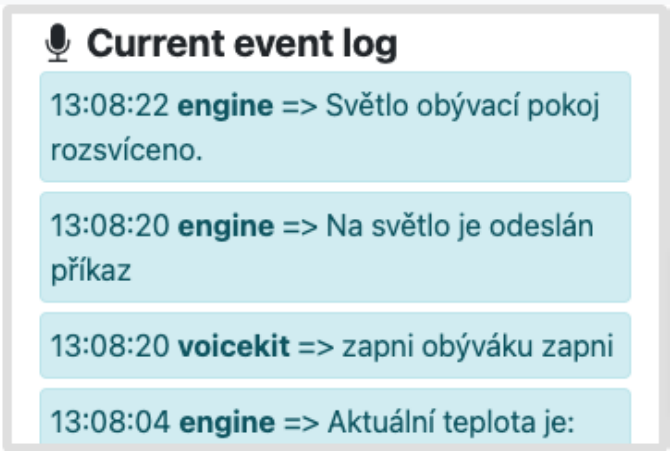


Figure 5.3: The screenshot of the Current event log box on the Home page

The box next to the Current event log box is Lights state box (see Fig. 5.4). All connected lights with an image of a bulb are listed in this box. The bulb indicates whether the light is turned on, off or not connected. When the bulb image is pressed, the light turns on or off, depending on the initial state.

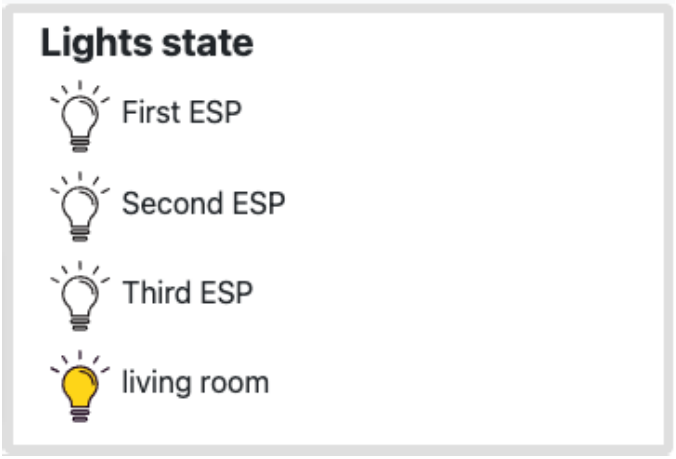


Figure 5.4: The screenshot of the Lights state box on the Home page

The box below the Current event log is Weather (see Fig. 5.5). This box shows the fundamental weather forecast for today and tomorrow. At the top is placed current date and time. Below time is displayed forecast for today, which contains icons that specify weather type (such as rain, sun, fog, snowfall). Next to the icon is today's forecast temperature. The row below shows a forecast temperature for a day temperature and night temperature with a rain volume displayed next to that. The last row in this box display day temperature, night temperature and rain volume for tomorrow.

Figure 5.5: The screenshot of the Weather box on the Home page

The box on the left of the Weather box is the Sensors state box (see Fig. 5.6). This box shows the indicator and description of each sensor. The indicator is red when the sensor

- is not working correctly;
- it is an error in data;
- the sensor has a delay in communication for more than 5 minutes.

Otherwise, the indicator is green.

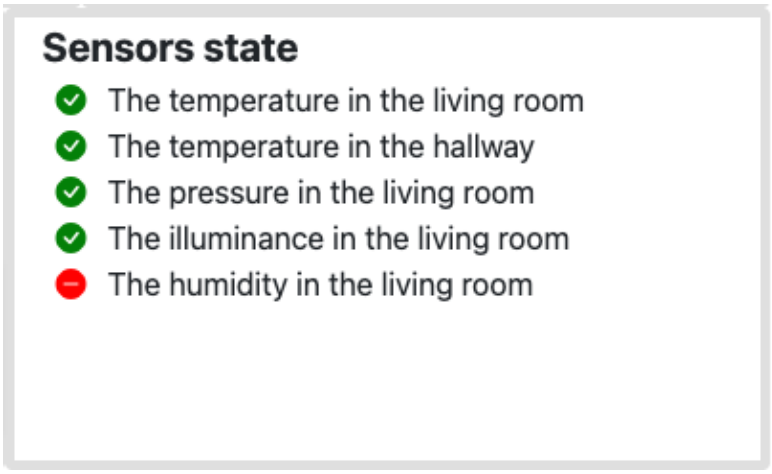


Figure 5.6: The creenshot of the Sensors state box on the Home page

The last box on the page is the Currently measured values box (see Fig. 5.7). This box shows current data from sensors that arrange by rooms.

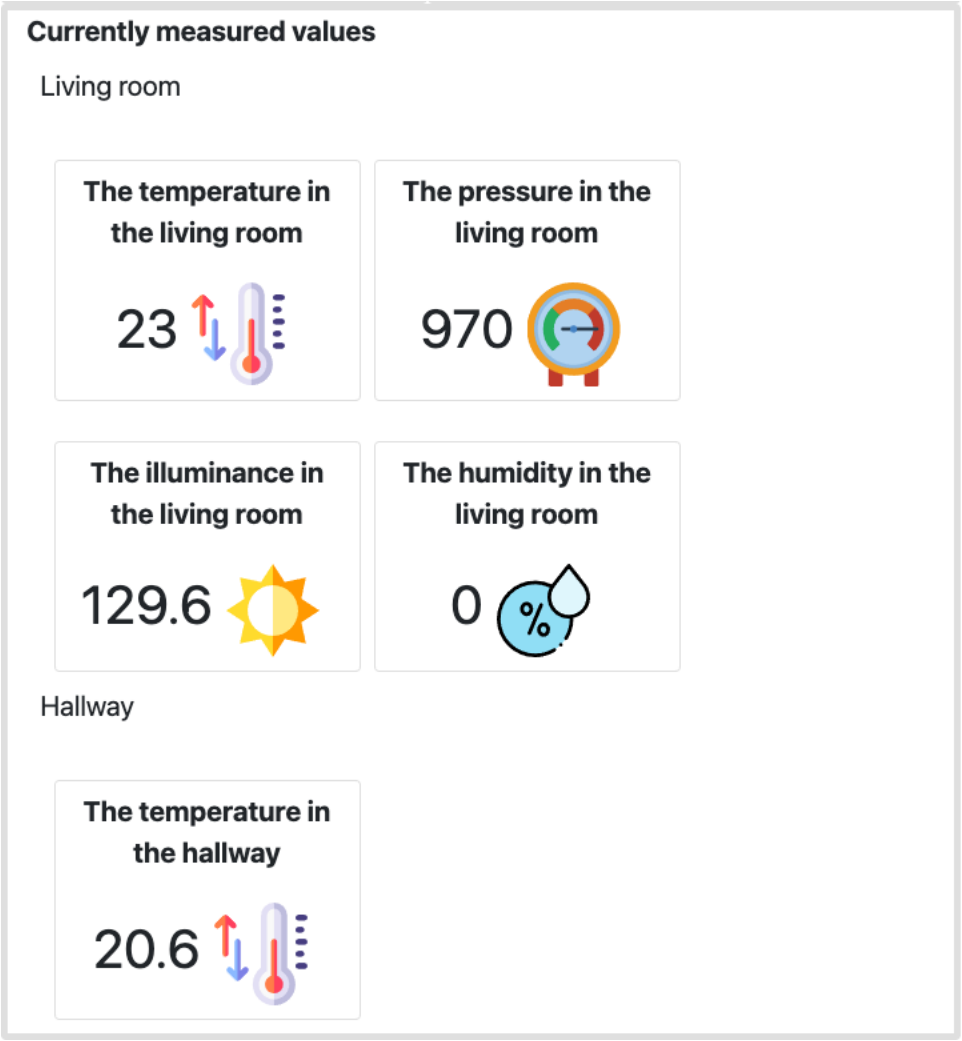


Figure 5.7: The screenshot of the Currently measured values on the Home page

All values from ESP on this page are changed asynchronously according to the new data arriving at the server using the *WebSocket* protocol (see Section 3.3.2).

5.2 Analytics

The Analytics page (see Fig. 5.8) allows user to select specific sensors and visualize their historical values stored in the MongoDB database. When the user opens the page, all data from the engine are immediately sent to the page using WebSocket (see Section 3.3.2). The user's first step in entering the page is to select the sensors in the list of sensors he wants to plot below. After filling in these preferences, the user clicks the submit button `"Vykreslit data"`. At this point, the page begins to transform the received data into the desired Dygraph library format. The Dygraph library was chosen based on the benefits of an open-source license and easy selection of the time user want to display data. The time is selected using the banner below the graph by dragging the wick.

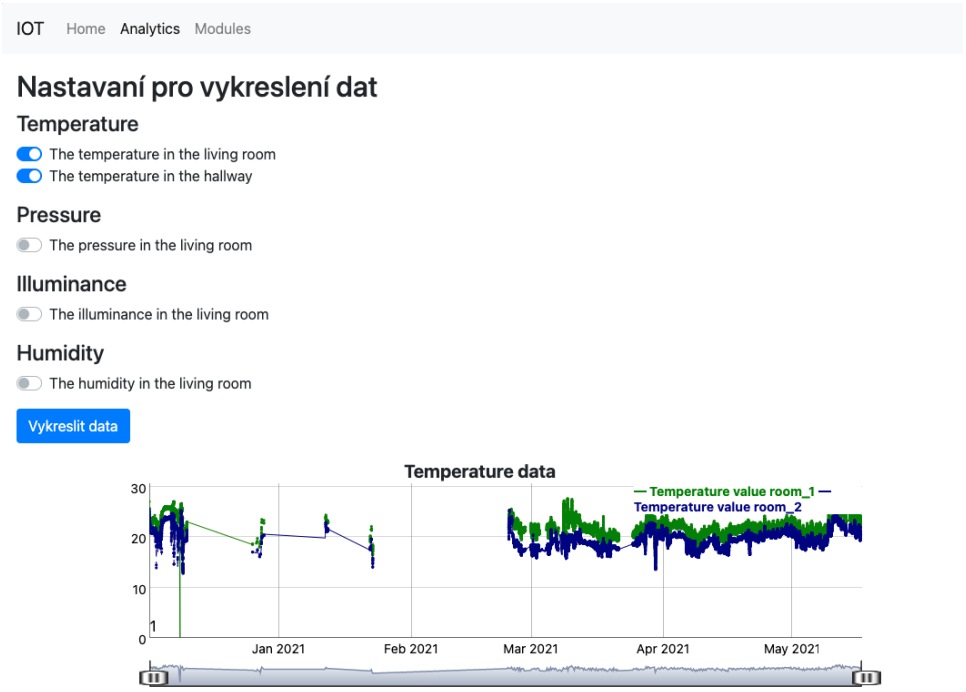


Figure 5.8: The Screenshot of the Analytics page

The Fig. 5.9 shows measured illuminance data from sensor TSL2591 visualized on the Analytics page. The sensor has been running non-stop from 23.2. to 22.5. except for a few stages when the system had to be shut down due to program fixes. In this graph is select time range from `'2021/02/24 11:05:31'` to `'2021/02/26 09:44:31'`. It is noticeable that the sensor was placed in a room with a window. Each day is separated from the other by night (about 0 lx). The graph also shows the phenomenon of lit lamp in the evening (about 20 or 60 lx depending on the lamp).

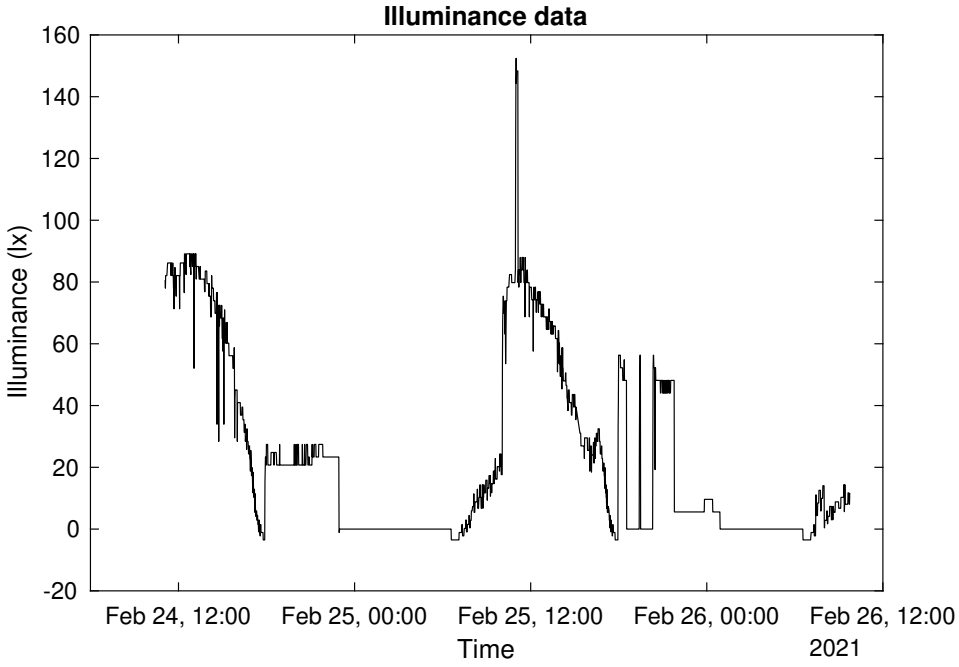


Figure 5.9: The plot of measured illuminance data by the sensor TSL2591

5.3 Modules

The Modules page allows the user to view all possible voice commands for each module. On the left, there is a list of all modules (see Fig. 5.10) that connect to the engine. After clicking on the user-selected module, the module name, the module description and a list of blocks of all possible voice commands show on the left side of the page (see Fig. 5.11). The function's name to execute, the description of the command and all possible calls display in the block of voice command.

The following functionality is turning on or off the module. The module turns by clicking the switch next to the module's name in the list of modules. This click deactivates the module in the system, and then its functions are no longer available. The system stops responding to voice commands in this module.

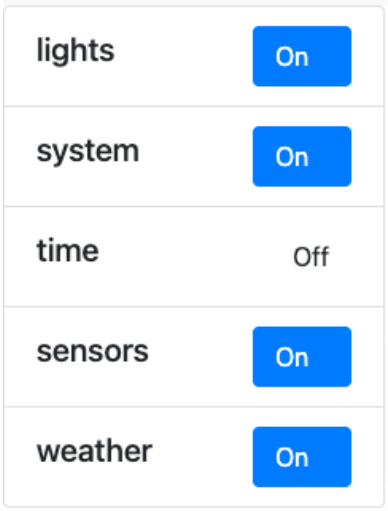


Figure 5.10: The screenshot of the list of all modules on the Modules page

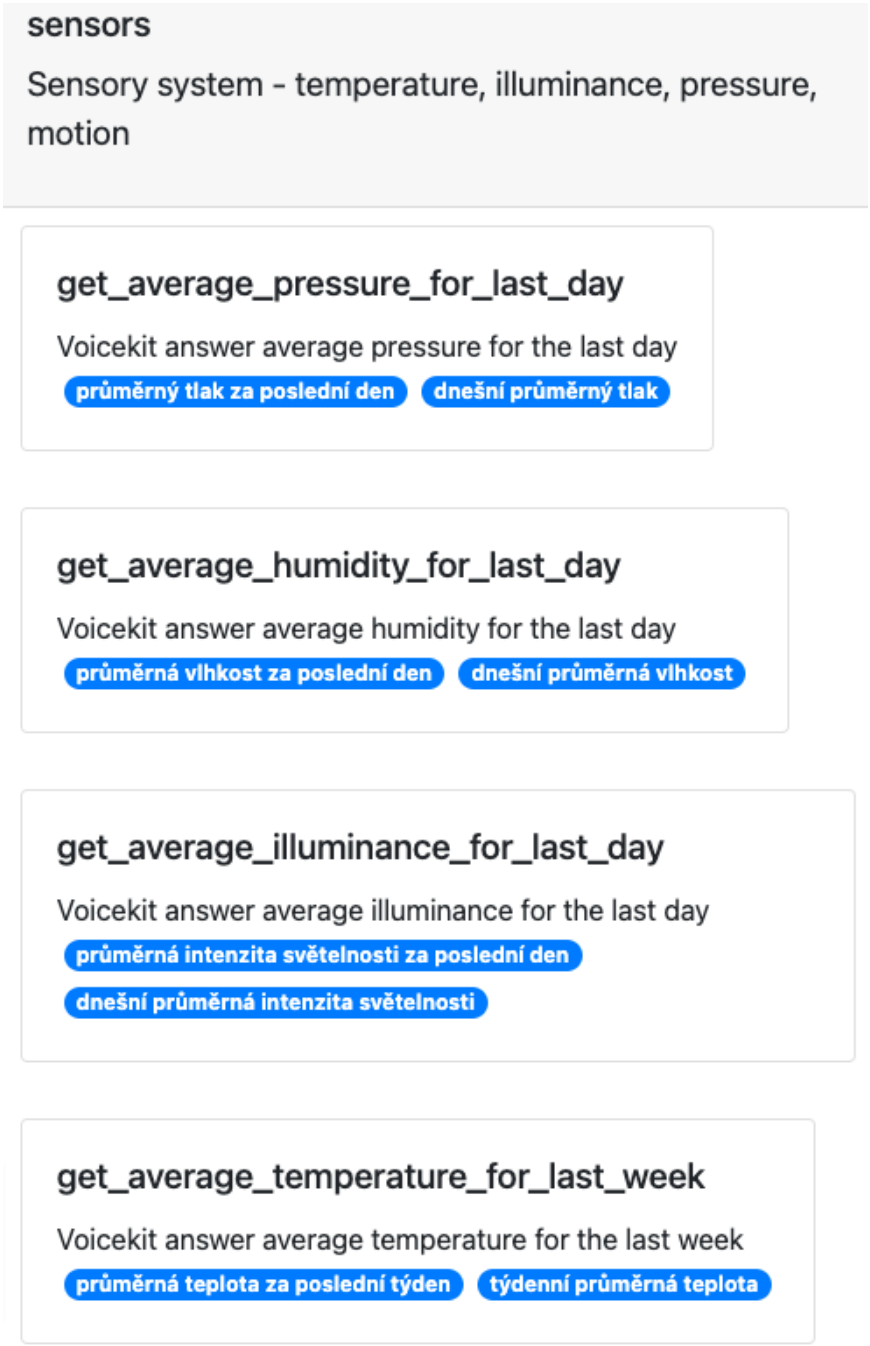


Figure 5.11: The screenshot of the list of calls for the module Sensors on the Modules page

Chapter 6

Discussion

The overall objective of the thesis consisted of five subtasks:

1. to test selected modules for use in the project;
2. to implement hardware-dependent modules physically;
3. to design an interface for communication between the user and selected modules;
4. to add an interface for voice interaction;
5. to enrich the project with other modules according to the time possibilities.

The thesis output indicates that with the help of now widely available technologies, building a smart home at a low cost is possible. As used in this project, everyone can use development boards, cheap sensors and motors, and free web hosting with this open-source project to make a home more like a smart home. A general open-source engine has been built to handle any module that complies (following the primary standard for connection to the engine). The user can communicate with modules in Czech, unlike Siri and other assistants. Thus, we can say that the output met our expectations to create an open-source, modularly functional system with voice-enabled modules for the smart home.

The thesis provides new insight into the relationship between cheap developing technology and available speech synthesis and speech recognition. When creating a module, it does not matter the principle it works, it can be based purely on Internet data, but it can also be linked to hardware. The module developer has several tools at his disposal, such as a database, a web interface and a communication channel that he can use. The thesis output provides an example of a practical implication.

It is beyond the scope of this study to address the question of security and the impossibility of leaking personal data. Due to the complexity of this work, there is undoubtedly much room for improvement, such as adding more modules. However, the thesis gives a basic idea and direction for the future, where it should go.

Further research is required to establish whether MQTT is set correctly for communication security. Further future research is essential to create a more efficient database and data storage, both for search speed and to save space.

Chapter 7

Conclusion

This thesis is about creating a voice-enabled smart home modules system. The purpose is to allow the user to create various modules for the smart home and a fully customised solution to his needs. The thesis consists of connecting and programming a VoiceKit, an engine running on a Raspberry Pi and three ESP development boards controlling three sensors and four lights. The sensors together measure four physical quantities such as temperature, illuminance, pressure and humidity. The project additionally includes an automatic speech recognition and speech synthesis system.

The first part of the thesis was designing a hardware solution and the physical implementation of individual sensors and lights. Subsequently, the ESP development board with the engine for data transfer from sensors and light control was programmed. Then a database for storing the data has been created. Next, we built the language processor VoiceKit and connected it to a SpeechCloud and the engine. In the next phase, modules were created in the engine with essential functions. At the end of the project, a comprehensive website was created to present the project's output.

To sum up the thesis, it is a good start and basis for future work. Many tasks in terms of communication and architecture of code have been solved. During this thesis, three ESP development boards were connected with three sensors and four lights. The general system has been created for connecting modules. Subsequently, the primary five modules were created both on a hardware basis and the Internet. Furthermore, a web page was created to present data, states, and voice commands. However, I think we have opened some new questions, so there is still much space to work on in this field.

7.1 Future Work

Some of the ideas for the future work are listed here.

- *Building a robust database.* The project's database is as simple as possible because it is not the topic of the thesis. Therefore, is there plenty of room for improvement and streamlining.

- *Building a web application*. The thesis's website is made without any model-view-ViewModel framework for the front end, so it would be better to create a single-page application using, for example, Vue.js.
- *Making a project security*. As already mentioned in the thesis field, great emphasis is placed on security and the impossibility of leaking personal data, which could not be explored due to time reasons and complexity of work.
- *Programming an ESP in the C language*. Python was used in the thesis to increase productivity and prototyping. The ESP developing boards would be better to program in C language for acceleration and less memory consumption of the algorithm in the long run.
- *Extending with complex functions and modules*. It is crucial for the future of the project that the development of other modules and functions continues. Many users would certainly appreciate being able to control, for example, music, radio, windows or home security in a smart home.

Bibliography

[1] Heiga Zen, Keiichi Tokuda, and Alan W. Black. “Statistical parametric speech synthesis”. In: *Speech Communication* 51.11 (2009), pp. 1039–1064. issn: 0167-6393. doi: https://doi.org/10.1016/j.specom.2009.04.004. url: https://www.sciencedirect.com/science/article/pii/S0167639309000648.

[2] *TSL2591 Datasheet*. Version ams163.5. ams AG, Austria-Europe. 2013. url: https://cdn-shop.adafruit.com/datasheets/TSL25911_Datasheet_EN_v1.pdf.

[3] 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.

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

[5] *BME280 Combined humidity and pressure sensor*. Version 1.6092018. BST-BME280-DS002-15, 0 273 141 185. Bosch Sensortec GmbH. 2018. url: https://ae-bst.resource.bosch.com/media/_tech/media/datasheets/BST-BME280-DS002.pdf.

[6] Matthew B. Hoy. “Alexa, Siri, Cortana, and more: An introduction to voice assistants”. In: *Medical Reference Services Quarterly* 37.1 (2018). doi: 10.1080/02763869.2018.1404391.

[7] Gene Munster. *Annual Digital Assistant IQ Test*. 2019. url: https://loupventures.com/annual-digital-assistant-iq-test/.

[8] Maxim Integrated Products. *DS18B20 Programmable Resolution 1-Wire Digital Thermometer*. Version 6. Maxim Integrated Products, Inc. 2019. url: https://datasheets.maximintegrated.com/en/ds/DS18B20.pdf.

[9] Wikimedia Commons. *File:Google AIY Voice Kit (34225487980).png — Wikimedia Commons, the free media repository*. [Online; accessed 22-May-2021]. 2020. url: https://commons.wikimedia.org/w/index.php?title=File:Google_AIY_Voice_Kit_(34225487980).png&oldid=500774098.

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

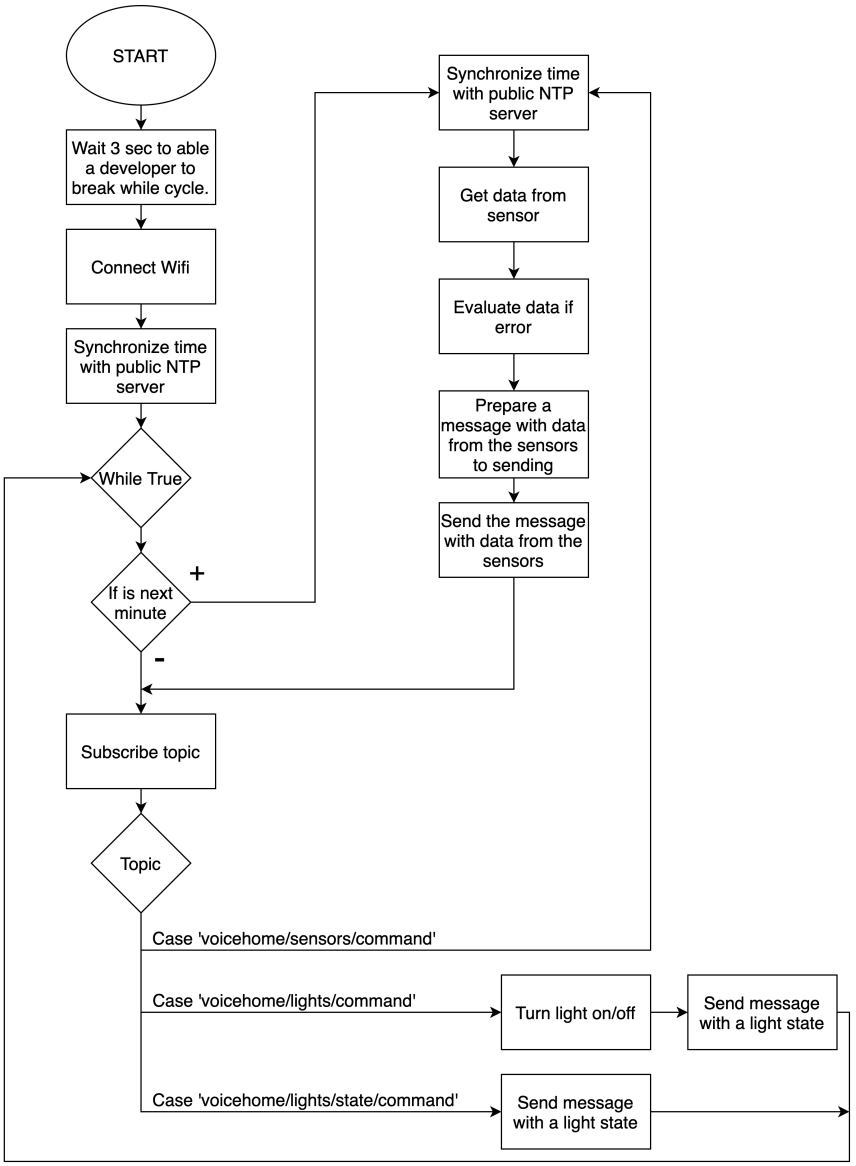
[11] Petr Stanislav. “Speech recognition of patients after total laryngectomy communicating by electrolarynx”. PhD dissertation. Západočeská univerzita v Plzni, 2020.

[12] Google. *VoiceKit*. url: https://aiyprojects.withgoogle.com/voice/.

Appendix A1

Diagram of an Algorithm Running on the ESP

The following figure shows a diagram of an algorithm running on the ESP development board.



Appendix A2

Modules Calls

A2.1 Lights

The module responds to the following questions:

- Turn on all the onboard LEDs
  Voice commands
  - "Rozsviť všechny vývojové desky."
  - "Rozsviť všechny vestavěné ledky."

  Reply
  - Module confirm each light separately - "Vývojová deska jedna je rozsvícena.", "Vývojová deska dva je rozsvícena.", etc.
- Turn off all the onboard LEDs
  Voice commands
  - "Zhasni všechny vývojové desky."
  - "Zhasni všechny vestavěné ledky."

  Reply
  - Module confirm each light separately - "Vývojová deska jedna je zhasnuta.", "Vývojová deska dva je zhasnuta.", etc.
- Turn on the light 1
  Voice commands
  - "Zapni obýváku světlo."
  - "Rozsviť obýváku světlo."
  - "Zapni obývacím pokoji světlo."
  - "Rozsviť obývacím pokoji světlo."

  Reply
  - "Světlo v obývacím pokoji rozsvíceno."
- Turn off the light 1
  Voice commands
  - "Vypni obýváku světlo."

  - "Zhasni obýváku světlo."
  - "Vypni obývacím pokoji světlo."
  - "Zhasni obývacím pokoji světlo."

  Reply

  - "Světlo v obývacím pokoji zhasnuto."

- Turn on the onboard LED number 1
  Voice commands

  - "Rozsviť vestavěnou ledku vývojové desky číslo jedna."
  - "Rozsviť vývojovou desku číslo jedna."

  Reply

  - "Vývojová deska číslo jedna rozsvícena."

- Turn off the onboard LED number 1
  Voice commands

  - "Zhasni vestavěnou ledku vývojové desky číslo jedna."
  - "Zhasni vývojovou desku číslo jedna."

  Reply

  - "Vývojová deska číslo jedna zhasnuta."

- Turn on the onboard LED number 2
  Voice commands

  - "Rozsviť vestavěnou ledku vývojové desky číslo dva."
  - "Rozsviť vývojovou desku číslo dva."

  Reply

  - "Vývojová deska číslo dva rozsvícena."

- Turn off the onboard LED number 2
  Voice commands

  - "Zhasni vestavěnou ledku vývojové desky číslo dva."
  - "Zhasni vývojovou desku číslo dva."

  Reply

  - "Vývojová deska číslo dva zhasnuta."

- Turn on the onboard LED number 3
  Voice commands

  - "Rozsviť vestavěnou ledku vývojové desky číslo tři."
  - "Rozsviť vývojovou desku číslo tři."

  Reply

  - "Vývojová deska číslo tři rozsvícena."

- Turn off the onboard LED number 3
  Voice commands
  - "Zhasni vestavěnou ledku vývojové desky číslo tři."
  - "Zhasni vývojovou desku číslo tři."

  Reply
  - "Vývojová deska číslo tři zhasnuta."

- Voicekit answer which lights are turned on
  Voice commands
  - "Která světla svítí."

  Reply
  - "Aktuálně nejsou rozsvícena žádná světla."
  - "Aktuálně jsou rozsvícena tyto světla první ESP, druhé ESP..."

A2.2 Sensors

- Sends a command via MQTT to measure current temperature
  Voice commands
  - "Kolik je stupňů?"
  - "Jaká je teplota?"
  - "Změř teplotu."

  Reply
  - "Na senzor je odeslán dotaz. Aktuální teplota je dvacet."

- Sends a command via MQTT to measure current pressure
  Voice commands
  - "Kolik je tlak?"
  - "Jaký je tlak?"
  - "Změř tlak."

  Reply
  - "Na senzor je odeslán dotaz. Aktuální tlak je devět set devadesát devět."

- Sends a command via MQTT to measure current humidity
  Voice commands
  - "Kolik je vlhkost?"
  - "Jaká je vlhkost?"
  - "Změř vlhkost."

Reply

- "Na senzor je odeslán dotaz. Aktuální vlhkost je deset."

- Sends a command via MQTT to measure current illuminance
  Voice commands
  - "Kolik je intenzity světla?"
  - "Jaká je intenzita světla?"
  - "Změř světlo."

  Reply
  - "Na senzor je odeslán dotaz. Aktuální intenzita světla je třicet."

- Voicekit answer average temperature for the last day
  Voice commands
  - "Průměrná teplota za poslední den."
  - "Dnešní průměrná teplota."

  Reply
  - "Teplotu nebylo možné vypočíst."
  - "Průměrná teplota za poslední den je dvacet."

- Voicekit answer average pressure for the last day
  Voice commands
  - "Průměrný tlak za poslední den."
  - "Dnešní průměrný tlak."

  Reply
  - "Tlak nebylo možné vypočíst."
  - "Průměrný tlak za poslední den je devět set devadesát."

- Voicekit answer average humidity for the last day
  Voice commands
  - "Průměrná vlhkost za poslední den."
  - "Dnešní průměrná vlhkost."

  Reply
  - "Vlhkost nebylo možné vypočíst."
  - "Průměrná vlhkost za poslední den je devět set devadesát."

- Voicekit answer average illuminance for the last day
  Voice commands
  - "Průměrná intenzita světelnosti za poslední den."
  - "Dnešní průměrná intenzita světelnosti."

Reply

- "Světelnost nebylo možné vypočíst."
- "Průměrná intenzita světelnosti za poslední den je dvanáct."

- Voicekit answer average temperature for the last week
  Voice commands
  - "Průměrná teplota za poslední týden."
  - "Týdenní průměrná teplota."

  Reply
  - "Teplotu nebylo možné vypočíst."
  - "Průměrná teplota za poslední týden je dvacet."
- Voicekit answer average pressure for the last week
  Voice commands
  - "Průměrný tlak za poslední týden."
  - "Týdenní průměrný tlak."

  Reply
  - "Tlak nebylo možné vypočíst."
  - "Průměrný tlak za poslední týden je devět set devadesát."
- Voicekit answer average humidity for the last week
  Voice commands
  - "Průměrná vlhkost za poslední týden."
  - "Týdenní průměrná vlhkost."

  Reply
  - "Vlhkost nebylo možné vypočíst."
  - "Průměrná vlhkost za poslední týden je devět set devadesát."
- Voicekit answer average illuminance for the last week
  Voice commands
  - "Průměrná intenzita světelnosti za poslední týden."
  - "Týdenní průměrná intenzita světelnosti."

  Reply
  - "Světelnost nebylo možné vypočíst."
  - "Průměrná intenzita světelnosti za poslední týden je dvanáct."

A2.3 Time

- Send command to ask the server for the current time
  Voice commands

  - Kolik je hodin?
  - Čas

  Reply

  - Právě je pět hodin dvacet minut a pět sekund.

- Send command to ask the server for the current day of year
  Voice commands

  - Kolikátého dnes je?
  - Datum

  Reply

  - Dnes je 4. 5. 2021

- Send a command to the server to start timer on 3 minute
  Voice commands

  - Zapni časovač

  Reply

  - Časovač je nastaven na 3 minuty

- Send a command to the server to stop timer
  Voice commands

  - Vypni časovač
  - Zastav časovač

  Reply

  - Časovač je vypnut

- Ask the server for today's day of the week
  Voice commands

  - Co je za den v týdnu?
  - Co je za den?

  Reply

  - Dnes je pondělí.

- Ask the server for today's sunrise time
  Voice commands

  - Kdy vychází slunce?
  - Východ slunce

  Reply

  - Nebylo možno získat data ze serveru meteogram.cz
  - Slunce vychází v šest hodin a třicet minut.

- Ask the server for today's sunset time
  Voice commands
  - Kdy zapadá slunce?
  - Západ slunce

  Reply
  - Nebylo možno získat data ze serveru meteogram.cz
  - Slunce zapadá v osmnáct hodin a třicet minut.
- Ask the server for today's nameday
  Voice commands
  - Kdo má dnes svátek?
  - Svátek

  Reply
  - Nebylo možno získat data ze serveru svatky.centrum.cz
  - Podle serveru svatky.centrum.cz Renata.

A2.4 System

- Reloads all the modules again. A fresh refresh.
  Voice commands
  - Načti moduly
  - Aktualizuj moduly
  - Přenačti moduly

  Reply
  - Moduly byly znovu načteny.
- Makes a testing write and read with the database.
  Voice commands
  - Otestuj databáze
  - Test databáze
  - Otestuj databázi

  Reply
  - Modul System: Databáze otestována. Vyhledáno dat jeden.
  - Modul System: Chyba při testování databáze
- Makes a testing MQTT publish to voicehome/system/test, which this module is also subscribing
  Voice commands
  - Otestuj MQTT

  - Test MQTT
  - Vyzkoušej MQTT

  Reply

  - Na mqtt nebylo možné odesla zprávu
  - Zpráva na mqtt odeslána

- Sends a testing websocket message with passport system/test.
  Voice commands

  - Otestuj WebSocket
  - Test WebSocket
  - Vyzkoušej WebSockety

  Reply

  - Na websoket nebylo možné odesla zprávu
  - Zpráva na websoket odeslána

A2.5 Weather

- Getting forecast for today from www.chmi.cz.
  Voice commands

  - Dnešní předpověď
  - Jak dnes bude?

  Reply

  - Nebylo možno získat data ze serveru chmi.cz
  - Server chmi.cz předpovídá pro dnešek. Polojasno až oblačno, místy přeháňky...

- Getting forecast for tomorrow from www.chmi.cz.
  Voice commands

  - Předpověď zítra
  - Jak zítra bude?

  Reply

  - Nebylo možno získat data ze serveru chmi.cz
  - Server chmi.cz předpovídá pro zítřek. Polojasno až oblačno, místy přeháňky...

- Getting forecast for monday from www.chmi.cz if it is up to four days and not today.
  Voice commands

  - Předpověď na pondělí

  - Jak bude pondělí?

  Reply

  - Nebylo možno získat data ze serveru chmi.cz
  - Server chmi.cz předpovídá na pondělí. Polojasno až oblačno, místy přeháňky...

- Getting forecast for tuesday from www.chmi.cz if it is up to four days and not today.
  Voice commands

  - Předpověď na úterý
  - Jak bude úterý?

  Reply

  - Nebylo možno získat data ze serveru chmi.cz
  - Server chmi.cz předpovídá na úterý. Polojasno až oblačno, místy přeháňky...

- Getting forecast for wednesday from www.chmi.cz if it is up to four days and not today.
  Voice commands

  - Předpověď na středu
  - Jak bude středu?

  Reply

  - Nebylo možno získat data ze serveru chmi.cz
  - Server chmi.cz předpovídá na středu. Polojasno až oblačno, místy přeháňky...

- Getting forecast for thursday from www.chmi.cz if it is up to four days and not today.
  Voice commands

  - Předpověď na čtvrtek
  - Jak bude čtvrtek?

  Reply

  - Nebylo možno získat data ze serveru chmi.cz
  - Server chmi.cz předpovídá na čtvrtek. Polojasno až oblačno, místy přeháňky...

- Getting forecast for friday from www.chmi.cz if it is up to four days and not today.
  Voice commands

  - Předpověď na pátek
  - Jak bude pátek?

Reply

- Nebylo možno získat data ze serveru chmi.cz
- Server chmi.cz předpovídá na pátek. Polojasno až oblačno, místy přeháňky...

- Getting forecast for saturday from www.chmi.cz if it is up to four days and not today.
  Voice commands
  - Předpověď na sobotu
  - Jak bude sobotu?

  Reply
  - Nebylo možno získat data ze serveru chmi.cz
  - Server chmi.cz předpovídá na sobotu. Polojasno až oblačno, místy přeháňky...

- Getting forecast for sunday from www.chmi.cz if it is up to four days and not today.
  Voice commands
  - Předpověď na neděli
  - Jak bude neděli?

  Reply
  - Nebylo možno získat data ze serveru chmi.cz
  - Server chmi.cz předpovídá na neděli. Polojasno až oblačno, místy přeháňky...