Silesian
University
of Technology

Faculty of Automatic Control,
Electronics and Computer Science

Internet Technologies – project work

Plantie™

Authors:

Oskar Stabla

Kamil Choiński

Year, semester, group, section: 2020, semester 5, group 1, section 1

Project supervisor: dr inż. Stanisław Wrona

Gliwice 2020

Contents

Chapter 1

Introduction and scope

We are living in times where everything is being automatized, people have less time for everyday tasks and became lazier. Having that in mind, we decided to create something useful, engaging and relatively complex for us to create, something that everyone with just a little knowledge of electronics would be able to implement having only the hardware and the source code. Also, we wanted to open up for some new technologies, which may be helpful in the future as well. This resulted in creating a remote plant-management system connected through WI-FI to hardware stuffed with sensors mainly based on Arduino IDE and Node.js server. With our product, people will be able to save their precious time.

Chapter 2

Schedule

In almost every project there is a schedule in which we assume, that certain things should be done, but most of the time (it happened there too) the primal schedule needs correction. We did not really know how much time it takes to prepare certain things so we've made a very "safe" schedule not to be late with any of the parts.

2.1 Schedule approved at the beginning

Week 1:
Creating site sketch in HTML
Week 2:
Creating CSS styles
Week 3:
Getting acquainted with Node.js technology
Week 4:
Setting up page on a server
Week 5, 6
Implementing our 2-way communication system
Week 7, 8
Final technical and visual modifications. Tests

2.2 Schedule reflecting actual work

During the first week we were getting to know how everything works and what are the possible solutions and technologies for our project. Having the base, we decided to create a simple website in HTML on which we could learn the basics of CSS as well.

Second week was quite more resultful. Old-fashioned front-end was replaced with a new one written from scratch once again, but with a help of Bootstrap. Some icons were added using Fontawesome, everything started looking way better than we expected. Also we decided to work a little bit on back-end. Simple local server was started using Node.js, for now it was sufficient.

When third week began, there was a need to work on JavaScript together with requests to the server and a database, which was created using MySQL set on separate server from Oracle and a Sequelize framework simplifying database operations to working on JS objects. Unfortunately, MySQL appeared to be too complex and resource-consuming for a simple usage, so we changed it to SQLite on a single file. Also, long script code was replaced with a shorter one using jQuery.

During the rest of the time, we were working on two-way communication and minor (which appeared to be major) updates. Separate socket server was set using node (instead of ESP as we thought) and simple commands could be sent to ESP and Arduino like turning LED light on and off for a test purpose. With that solution, many other ESPs can be connected simultaneously if needed in the future. Everything was constantly being improved to work with current setup and sensors during the Christmas Holidays and database.

Chapter 3

Software and hardware implementation

3.1 Defining the problem

The main problem was that we were not in touch with any of the internet technologies. It took a while to do some research, which, to our surprise, is not that easy when you do not even know what to look for. Second one was to spare some time for tutorials, but eventually it was like a bottomless well, the more we knew, the more we needed to learn and with every idea came another technology.

3.2 Analysis of possible solutions

Thankfully, in XXI age, the knowledge is widely available and we were overwhelmed by the possibilities to accomplish our project. The first one was to create a mobile application for android to operate the Arduino and read sensor values. Then we had an idea to create a website, which does not actually need a phone, but can also be used on it. Second option was more suitable for us, especially when we wanted to have everything on our computer and learn more about creating a fully functional one.

3.3 Implementation and problems during project development

Everything was written using Visual Studio Code with a Prettier extention making the overall look of the code "prettier". Visual made writing the code way faster with his hints under the Tab button, colors and even descriptions of used functions.

3.3.1 Front-end

As beginners, we encountered a lot of implementational and visual problems. We did not really know what "good looking website" means. First tries of doing the main page were a failure written using only CSS. Learning from mistakes and getting some more information about possible solutions for this problem, we decided to use a Bootstrap, which is an open source toolkit allowing us to insert already done objects in a class field, which appeared to be very helpful in almost everything like buttons, panels, indicators and navbar presented below.

```
<nav class="navbar navbar-expand-md bg-dark navbar-dark">
   <a class="navbar-brand nav-item" href="index.html"><img src="img/
       logo_plantie2_green_2.png" alt="logo" style="width:160px"></a>
   <!-- logo display and go back to home  -->
   <button class="navbar-toggler" type="button" data-toggle="collapse" data-target="
       #collapsibleNavbar">
      <span class="navbar-toggler-icon"></span>
   </button>
      <div class="collapse navbar-collapse" id="collapsibleNavbar">
            <ul class="navbar-nav ml-auto">
      <!-- Navigation buttons -->
         <li class="nav-item ">
         <a class="nav-link" href="index.html">Home</a>
      </li>
         <li class="nav-item ">
         <a class="nav-link" href="about.html">About</a>
      </li>
         <li class="nav-item ">
         <a class="nav-link" href="control.html">Control</a>
      </li>
         <li class="nav-item active">
         <a class="nav-link" href="readings.html">Readings</a>
      </li>
   </ul>
   </div>
</nav>
```

Our front-end is quite simple and that is what was our priority. Basically just a main page describing used technologies and a control panel on the "readings" endpoint with a chart underneath. We also spent a while for the website to be responsive. On the other hand, compared to front, back-end is more interesting to look at.

3.3.2 Back-end

Server was set using Node.js, which is a multi-platform environment used in creating server-side applications, and an Express.js framework providing a myriad of HTTP utility methods.

```
var express = require("express");
var app = express();
//create a server
const server = http.createServer(app);
//Getting html file
app.get("/", function(req, resp) {
  resp.sendFile("index.html", { root: path.join(__dirname, "./views") });
});
//Server listens for requests comming from localhost
app.listen(3000, "localhost", function() {
  console.log("Application worker " + process.pid + " started...");
});
```

An example showing the simplicity of used solution

Previously, we had an idea to use MySQL database which we already had on our computer as a result of education purposes. Unfortunately, there was no point in setting another server for a single database, single table and several attributes, it would be just triumph of form over content. Also, project was being tested on at least 3 notebooks depending on the time and some organisation issues, it just needed to be a simple as that. Eventually, database was made in SQLite and operated on with promise-based Node.js ORM - Sequelize, allowing to work on objects instead of writing raw queries. We used it to both fetch from and insert into database our readings. Here is a fragment of code containing important declarations and creating Sensor_Readings object:

```
const sequelize = new Sequelize({
   dialect: "sqlite",
   storage: "database/baza.sqlite",
   define: {
    timestamps: false,
    id: false
   }
});
const Model = Sequelize.Model;
class Sensor_readings extends Model {}
Sensor_readings.init(OUR TABLES);
```

For fully functioning communication, we needed to make a connection between Hardware and Software. Keeping everything on a computer, we decided to make a socket server. It is used to manage connection requests from clients and is able to receive and send data. In our project, we send data to every client, decision if the message is for certain board is made on Arduino and/or ESP software. The implementation in site's javascript is as follows.

```
var sock = new WebSocket("ws://localhost:3000");
//data is being sent to all clients currently connected to the server
async function send_command(data) {
  sock.send(data);
}
```

Most of the socket source code lies in Node.js file

```
const WebSocket = require("ws");
const wsServer = new WebSocket.Server({ server });

wsServer.on("connection", function(ws, req) {
  /******* when server receives messsage from client trigger function with argument
      message *****/
   ws.on("message", function(message) {})
   ws.on("close", function() {
         console.log("lost one client");
    });
   wsServer.clients.forEach(function(client) {
   //broadcast incoming message to all clients (s.clients)
   if (client != ws && client.readyState) {
   //except to the same client (ws) that sent this message
   client.send("broadcast: " + message);
       }
       });
    });
   console.log("new client connected");
});
```

In simplification, when socket server gets a message, it triggers a function called ws.on with "message" argument. Then, after checking if message is complete (starts and ends within brackets) and contains readings (is other than any of commands), we divide it into separate values and write into the database object using Sequelize methods. If the whole process is a success, we update the database.

Being able to collect and operate on given data, we created asynchronous function in javascript responsible for making "get" request to the server, which in response provided whole database data as an array consisting of three elements: first containing last record of the database, second one with every element in the table and third one with the amount of data. A single javascript function could update readings on the control panel and make a chart underneath. A chart was done using Charts.js, an open source JavaScript library that allowed us to draw different types of charts by using the HTML5 canvas element. We decided to keep it in a function of Soil Moisture in a certain Time, which caused a lot of trouble. For example. time must be saved in certain ISO format, also the scale must have been changed into "series", so every day has his own space now and looks way better than a linear one. A website has a regulable timer performing "Update" and "Get from database" actions periodically every certain amount of time.The asynchronous function is shown below:

```
async function get_from_database() {
  try {
    let response = await fetch("/readings", {
      method: "get",
      headers: {
        "Content-Type": "application/json"
      }
    });
    response = await response.json();
    console.log("Got response!", response);
    $("#Soil_moisture").text(response[0].Soil_Moisture + "%");
    $("#Air_temperature").text(response[0].Air_Temperature + "o");
    $("#Brightness_level").text(response[0].Brightness_Level + "%");
    $("#Water_tank_level").text(response[0].Water_Tank_Level + "%");
    $("#Air_Humidity").text(response[0].Air_Humidity + "%");
    makeChart(response[1], response[2]); // response 1 is data for chart, response 2
         is amount of data
  } catch (err) {
    console.error(`Error: \${err}`);
    $("#Soil_moisture").text("err");
    $("#Air_temperature").text("err");
    $("#Brightness_level").text("err");
    $("#Water_tank_level").text("err");
    $("#Air_Humidity").text("err");
  }
}
```

Describing the whole process of connecting and sending, we should take a look at how the ESP, which is actually a microprocessor board with build-in WiFi controller, works in this project. There are, to be precise, three parameters which it needs: SSID and password for the Hotspot or router connecton and socket server's IP address being also IPv4 of the computer with local server. It tries to join the network and restart till it reached. After creating a successful connection with both router/hotspot and server, it waits for the message which is then processed. Currently there are two possible messages: only asking Arduino for updated sensor readings or turning on the water pump. Every action ends with response to the server.

In a real-time system, there is always a question about synchronization and delays. We needed to create an optimal solution, in which for example data sent to the database, server and ESP is always complete. We decided to keep data in angle brackets (<>) as it allowed us to do certain actions under condition, that the message starts and ends within a bracket, also the readings in the response message were separated with semicolons. ESP before sending given data, concatenates string values with brackets and semicolons.

```
//When there is something in serial buffer
 while (NodeMCU.available())
 {
 //Serial.println("NodeMCU available !@");
   char serialChar = NodeMCU.read();  //gets one byte from serial buffer
   if (serialChar == '<') //if < is the beginning of the received char
   reset the buffer
   {
     serialMessage = "";
     serialMessage += serialChar;
   }
   else if (serialChar == '>')
   {
     stringComplete = true;
     serialMessage += serialChar;
   }
   else
   {
   //Append the message within specified ascii section
     if ((serialChar >= 48 && serialChar <= 122) || serialChar == ',')
     serialMessage += serialChar;
   }
 }
```

Chapter 4

Summary

To our surprise, the project in general was more time consuming that we could ever expect. The lack of knowledge makes it hard to even get one, because we did not really know what to look for. Thankfully, the Internet nowadays provides lots of resources and tutorials that helped us in comming up with solutions and finding out why they do not work. In conclusion we are proud of the outcome, not everything went as expected but we managed to combine software and hardware solutions and make them work together. The project forced us to think outside of the box as there were neither imposed technologies nor the way to write our code. It was good practise to come up with something of our own, develop that one idea and bring it to the end.

Project was based on tutorials and documentation from the given websites: [**?**] [**?**] [**?**] [**?**] [**?**] [**?**] [**?**] [**?**] [**?**] [**?**]