OULU UNIVERSITY OF
APPLIED SCIENCES

Huong Nguyen

DESIGN AND IMPLEMENTATION OF A PET CARE SYSTEM

DESIGN AND IMPLEMENTATION OF A PET CARE SYSTEM

Huong Nguyen
Bachelor's Thesis
DIN15SP
Degree Programme in Information Technology
Oulu University of Applied Sciences

ABSTRACT

Oulu University of Applied Sciences
Degree Programme in Information Technology

Author: Huong Nguyen
Title of the bachelor's thesis: Design and implementation of a pet care system
Supervisor: Anne Keskitalo
Term and year of completion: Spring 2020 Number of pages: 49

The objective of this thesis was to build a Pet Care System using the trendiest technologies from both the Internet of Things and the Web development and to analyse different aspects of this approach.

To build a pet feeding machine, a food dispenser, a servo, and a configured Raspberry Pi were connected. The pet's food could be released from the food dispenser as a result of Raspberry Pi triggering the servo to rotate the spinner. A progressive web application for remotely manipulating the pet feeding machine was developed using ReactJS. As important as that, a web server for receiving control signals from the web app was built and deployed in Raspberry Pi using NodeJS.

As a result, a Pet Care System was made, it includes the pet feeding machine and the web application for remote control feeding activities. Core features, such as immediate feed and scheduling feed were implemented in this system. The analysis and evaluation of the project were also written to give more knowledge of the system.

Keywords: Raspberry Pi, progressive web app, React.js, Node.js, servo, pet care, smart pet feeder

PREFACE

This bachelor thesis was written at Oulu University of Applied Sciences as part of the curriculum of Degree Programme in Information Technology. This project was made under the supervision of Anne Keskitalo and Kaija Posio, who guided and supported the author a lot by providing helpful advice as well as resources.

Oulu, 30.1.2020
Huong Nguyen

CONTENTS

ABSTRACT 3
PREFACE 4
CONTENTS 5
VOCABULARY 7
1 INTRODUCTION 8
2 APPLIED TECHNOLOGY 9
2.1 Hardware technology 9
2.1.1 Raspberry Pi 9
2.1.2 Servo 11
2.1.3 Other relevant devices 12
2.2 Software technology 13
2.2.1 ReactJS 13
2.2.2 React Bootstrap 16
2.2.3 Progressive Web App 16
2.2.4 Node.js 16
2.2.5 Express.js 18
2.2.6 Node-schedule 18
2.2.7 Pigpio 19
3 DEVELOPMENT PROCEDURE 20
3.1 Project plan 20
3.2 Project requirements 20
4 ARCHITECTURE AND DESIGN 22
4.1 System architecture 22
4.2 UI design 23
5 PRODUCT DEVELOPMENT 26
5.1 Hardware installation 26
5.2 Environment establishment 27
5.2.1 Backend 27
5.2.2 Frontend 30
5.3 Feature development 31

5.3.1 Backend 31
5.3.2 Frontend 34
6 EVALUATION 45
6.1 Project evaluation 45
6.2 Evaluation of possible future development 45
7 CONCLUSION 47
REFERENCES 48

VOCABULARY

AJAX	Asynchronous JavaScript and XML
DIY	Do it yourself
DOM	Document Object Model
GPIO	General-purpose input/output
HTML	Hypertext Markup Language
JS	Javascript
JSX	JavaScript XML
LAN	Local Area Network
SSH	Secured Shell
UI	User Interface
PWM	Pulse Width Modulation

1 INTRODUCTION

Nowadays, owning a pet becomes more challenging for people due to the fact that pet health is affected by feeding schedule, while people cannot always stay at home to feed their dear friends. Although the urge to have an automatic pet feeder at home has raised significantly worldwide, it is not yet popular and affordable in many countries at the time. As a result, a DIY Pet Care System can be an excellent choice for people who are not able to purchase an automatic pet feeder.

The purpose of this thesis was to study and offer an approach to the mentioned problem: developing a pet care machine using Raspberry Pi as the main component and a web application used for remotely control the machine.

In this thesis, one of the main tasks was studying the development of the pet care system. That included different aspects, such as design, architecture, component installation and feature development.

Another main task was studying the technologies used to build a pet care system, explain them generally, as well as analyse them in relation to this specific system.

The mentioned tasks were all crucial and constructive in providing readers with an insight into popular and trending technologies in both web development and the IoT.

2 APPLIED TECHNOLOGY

2.1 Hardware technology

2.1.1 Raspberry Pi

“The Raspberry Pi is a low cost, credit-card sized computer that plugs into a computer monitor or TV, and uses a standard keyboard and mouse.” [1]

“The Raspberry Pi is a series of small single-board computers” [2]

Basically, Pi can be expected to do anything a desktop can such as browsing the Internet, playing music or acting as a game console. Pi can be used as a web server as well since it uses Debian – a Linux based operating system.

In Pi, important components which are responsible for a computer's operation, such as a microprocessor, a memory and ports, are all present in one single board. It is made not only useful but also convenient.

It is undeniable that Raspberry Pi may not be as powerful as a desktop computer; Pi, however, is still a perfect option for projects which do not require significant strong processing power. [3]

There are a number of Raspberry Pi models which can be chosen: Model A, Model B, Model Zero and Model Compute. There are some certain strengths and weaknesses in each model; for example, model Zero is the cheapest option but the Internet connection is not supported. While the price is higher than that of other models, Model B is considered to be the most powerful. [4]

In this project, Raspberry Pi 3 Model B was selected for product development. FIGURE 1 shows the structure of Raspberry Pi 3 Model B.

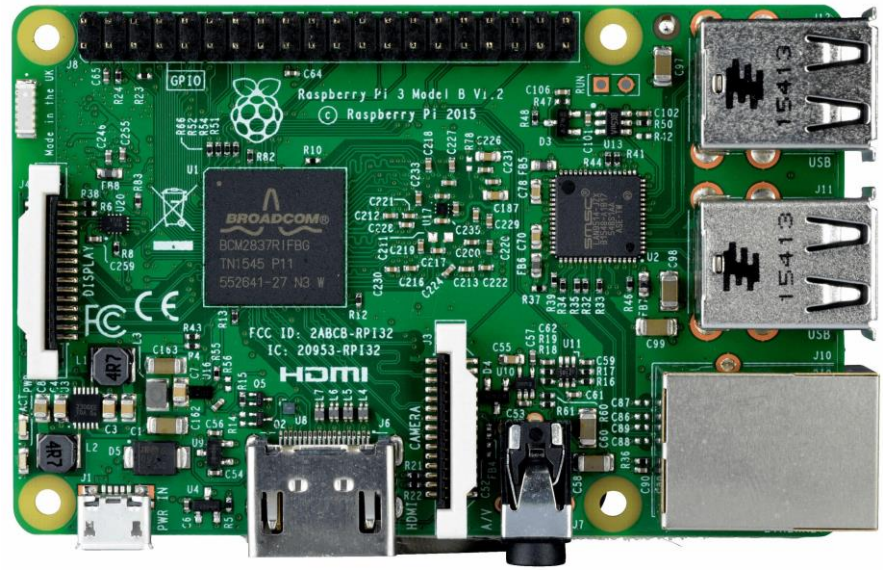

FIGURE 1. Raspberry Pi 3 Model B [5]

According to the official website [6], Raspberry Pi Model B has below specifications:

- Quad Core 1.2GHz Broadcom BCM2837 64bit CPU
- 1GB RAM
- BCM43438 wireless LAN and Bluetooth Low Energy (BLE) on board
- 100 Base Ethernet
- 40-pin extended GPIO
- 4 USB 2 ports
- 4 Pole stereo output and composite video port
- Full size HDMI
- CSI camera port for connecting a Raspberry Pi camera
- DSI display port for connecting a Raspberry Pi touchscreen display
- Micro SD port for loading your operating system and storing data

- Upgraded switched Micro USB power source up to 2.5A

2.1.2 Servo

“A Servo is a small device that incorporates a two-wire DC motor, a gear train, a potentiometer, an integrated circuit, and an output shaft.” [7] FIGURE 2 below shows a servo’s components.

The servo can be controlled by having GPIO pins turned on and off at a dramatically fast rate. The direction of the servo is controlled by a pulse width, which is the length of the pulses, while the frequency is kept constant. These signals are known as Pulse Width Modulation (PWM). [8]

Because Raspberry Pi does not have a clock system for PWM signal generation, it is a must to use software to generate PWM signals. Pigpio is a helpful library for that purpose.

Usually, a servo has a 3-pin plug connector which includes 3 wires of different colours: The black wire is for ground signal, the red one is for the power supply (at 5.5V) and the yellow one is for the control signal.

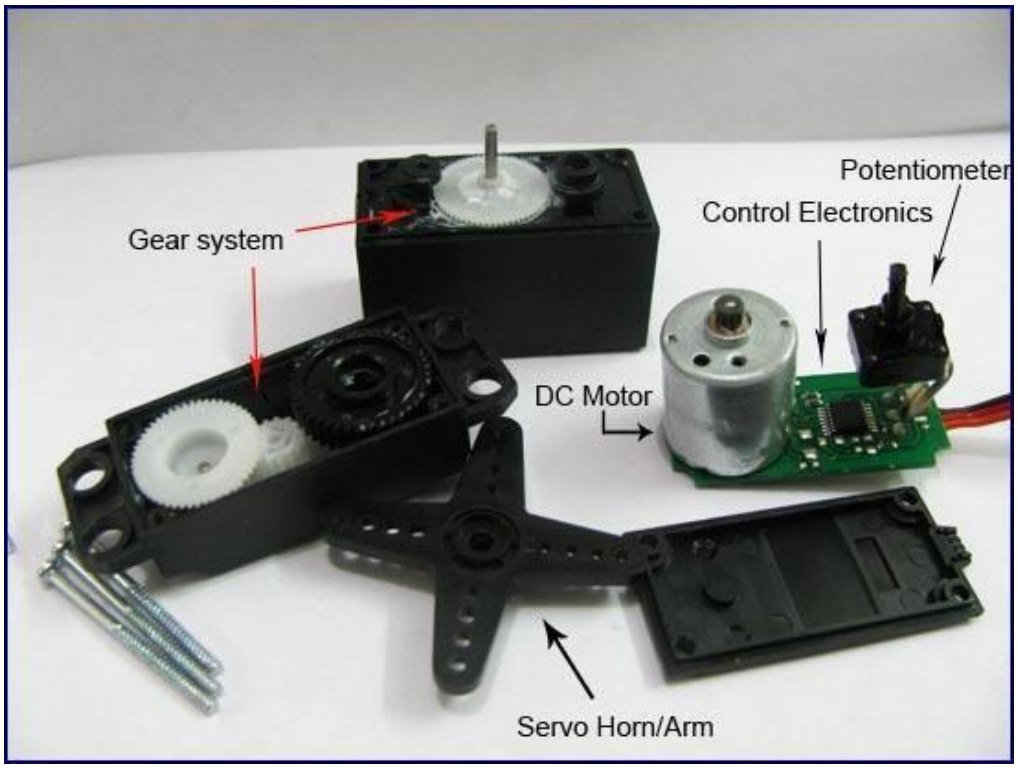

FIGURE 2. Servo's components [9]

2.1.3 Other relevant devices

A food dispenser with a silicone spinner is one of the most important items to build this pet care system. It can be made or bought easily in the local market. This food dispenser is for storing the pet's food; the spinner is made so that the food is released whenever the user rotates a button which leads the spinner to rotate. In this project, the food dispenser was bought as many important tools and materials could not be found. FIGURE 3 below shows the appearance of the food dispenser.

Tools, such as a wooden hanger and a piece of glue or tape, are required. They are used to attach the servo with the food dispenser.

FIGURE 3. Food dispenser

2.2 Software technology

2.2.1 ReactJS

ReactJS is a Javascript library which is designed for user interface development. One of the features which make React special is JSX. JSX is an XML like syntax extension for JavaScript. Traditionally, logic (JavaScript) and markup (HTML) are written separately. JSX, however, combines both logic and markup – HTML elements can be added to JavaScript and appended to the DOM without DOM methods.

Below is an example of JSX:

const element = <h1>Hello, world!</h1>;

The DOM, or the Document Object Model, has been known as a crucial part of a website; therefore, how to control it effectively is always a controversial issue in the

developer community. Nevertheless, the DOM's updating speed is usually slow in most JavaScript frameworks, as the whole DOM tree is updated every time there is a tiny change. ReactJS is known for the excellent ability to update only the DOM parts that have new changes, thus increasing the speed significantly. This feature of React is called Virtual DOM.

React creates the virtual DOM by creating a copy of the original DOM. Whenever there is a change detected, the whole virtual DOM is updated. React compares the updated virtual DOM with the original one; thus, the difference is rapidly identified and the changed part in the original DOM is updated in no time. [10]

In React, the UI is divided into multiple parts called "components". These components can work independently and they can be reused many times. Thanks to this feature, repeated codes can be avoided and DRY (Don't Repeat Yourself), which is an important principle for web developers to remember, can be made use of.

One-way data binding is another special feature of React. It means that data only flows in a single direction. Data can only be passed from the parent component to the child through props. If data needs to be passed from the child component to the parent, a callback function should be made to update the state [10].

"A single-page application is a web application or web site that interacts with the user by dynamically rewriting the current page rather than loading entire new pages from a server" [11]. Single-page applications have become popular nowadays, and React is a powerful tool for building them.

Because only the view layer of the application is handled by React, additional libraries and plugins are usually used in complex applications requiring some non-UI frontend features, such as API interactions and state management. This, at the same time, gives React more flexibility than other competitors in the market.

React and the competition

During recent years, the question about which frontend framework or library developers should use has been a controversial topic. ReactJS, AngularJS and VueJS are mentioned the most in multiple tech articles.

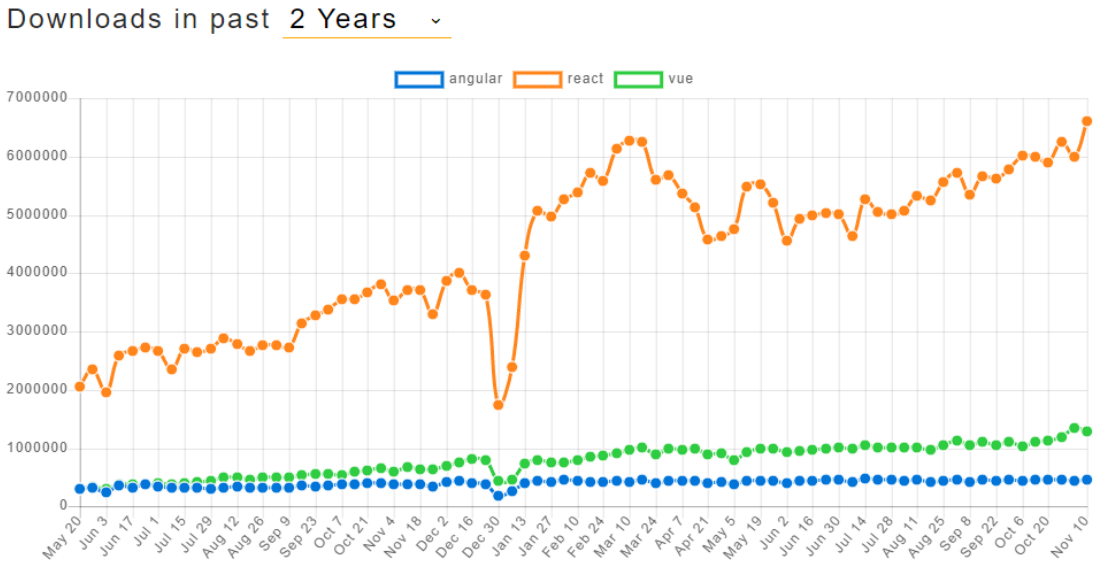

FIGURE 4 The number of downloads for popular Javascript frameworks in the past 2 years [12]

In terms of popularity, it is shown in the graph above (FIGURE 6) that ReactJS has always had the highest number of downloads, compared to AngularJS and VueJS.

Regarding the performance, the virtual DOM has made use of both React and Vue, which means that they are faster than Angular. Bidirectional data-binding is used in Angular, which is also a factor making it slower than React.

Both VueJS and ReactJS are significantly less lightweight compared to Angular. Even though Angular is heavy, more features ,such as templates and testing utilities, are offered.

As mentioned earlier, flexibility is React's strength because only the user interface part is handled by React. Additional frameworks and libraries can be used for specific demands, depending on the characteristics and requirements of each application.

In summary, if a fully equipped frontend framework is in need, Angular can be considered; in contrast, if a lightweight, flexible and fast option is prefered, React may be the best choice.

2.2.2 React Bootstrap

Nowadays, a responsive and mobile-first UI is a key factor in many modern web applications. React Bootstrap, which is a combination of React and Bootstrap components – the best UI libraries, can be an excellent option when it comes to building such web applications.

Usually, Bootstrap often comes with some dependencies such as jQuery or bootstrap.js, which affect the DOM directly. With React Bootstrap, however, Bootstrap works on the virtual DOM together with React to provide a more stable solution.

2.2.3 Progressive Web App

Progressive Web App is one of the most trendy technologies nowadays due to the fact that it offers a high performance and a mobile app-like experience without requiring the user to install it. It can be developed using HTML, CSS and JavaScript – the same languages with a normal web. The benefits PWA offers include a fast response, less data consumption, a push notification and an offline use. When it comes to building a PWA, React.js is one of the best choices as it is always easy and fast

2.2.4 Node.js

According to the Node.js official document:

“Node.js is a platform built on Chrome's JavaScript runtime for easily building fast and scalable network applications. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices.” [13]

Traditionally, servers use a thread-based networking model, which means that a new thread is created every time a request comes in. A number of troubles can be caused by this mechanism. The fact that Node.js is an asynchronous event-driven JavaScript

runtime makes it possible to maintain multiple connections at the same time. Whenever there is a request is sent to a Node.js server, a callback is fired. Thus, concurrent requests can create multiple callbacks which are able to work independently in a single thread.

One of the reasons why Node.js is popular is that it is significantly fast and scalable. The ability to handle requests based on single-thread with event looping mentioned above makes it extremely fast and lightweight. The number of loops is never limited and there is usually no blocking, Node is well-known for its scalability.

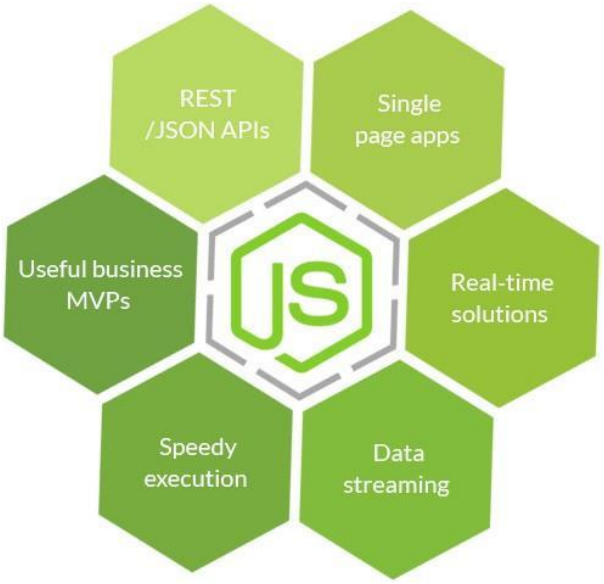

FIGURE 5. Node.js features [14]

Node.js is the solution for many kinds of applications. FIGURE 5 shows the typical applications which often make use of node. For example, in data streaming or real-time applications, it is very important that the speed is high and concurrent requests can be handled with little latency. Node.js servers perform effectively in such cases.

Node.js works consistently with a package manager called npm. Npm is the largest open source library which offters various dependencies for developing different features. Npm is helpful for package managing activities such as install or uninstall dependencies

2.2.5 Express.js

Express.js is a web framework for Node.js. A set of powerful features that are beneficial for building websites and mobile applications are offered. Middlewares being used for handling HTTP requests are also included in Express.js. Routing tables used for different actions with HTTP methods and URL can be defined. Especially, HTML rendering based on passing arguments to templates is allowed in Express.js. [15]

In this project, Express.js was necessary for the server to deal with HTTP requests. Routing also became easier with Express.js, while the middlewares it offered reduce the number of packages needed to be installed.

2.2.6 Node-schedule

Scheduling tasks in Node.js has become easier thanks to a Node module called Node-schedule. A task's execution time and the rule of its repetition can be set. There is only a single timer being made use of, and no upcoming jobs re-evaluation will be done.

It is crucial to distinguish time-based and interval-based scheduling. Interval-based scheduling should be used when the task requires repetition after a certain period. For example, running a chore every ten minutes is an interval-based scheduling task. The JavaScript function *setSchedule()* or similar functions can be applied for such a task. On the other hand, a time-based scheduling task, such as running a chore at a specific time every day, can be done easily with the help of Node-schedule.

In term of this specific project, Node schedule was applied because it was healthier for the pet to be fed at a specific time every day.

Node Schedule is fired when the *schedule()* method is called in a manually created job object. Another option is to call *scheduleJob().* This convenient function will help node automatically create a scheduling job and run it.

2.2.7 Pigpio

Pigpio is a library written in C for Raspberry Pi. It allows General Purpose Input Outputs (GPIO) manipulation. GPIO is a way Pi can communicate with the outside world by being connected to electronic circuits. [16]

In this project, the server was built using Node.js. As a result, a wrapper for Pigpio was installed for the servo control.

Two methods for controlling the servo are *servoWrite(pulseWidth)* and *getServoPulseWidth().* Among them, *servoWrite(pulseWidth)* was used in the prototype's code. This method is for adding the pulse width, which leads to the servo movement.

3 DEVELOPMENT PROCEDURE

3.1 Project plan

Generally, the product development process could be divided into these steps:

1. Environment setup. This step included Raspberry Pi, its OS setup and SSH configuration; All the required files and modules were installed and set up in the server and the client to prepare for the development step.
2. Feature development. During this step, the features mentioned in the project requirements were developed.
3. Testing. The features were tested based on the requirements aligned in the Project requirements part. The audit with Lighthouse could be performed to test the PWA. However, there was no automated testing since the scope of this project was narrow.

3.2 Project requirements

It was planned that a feature which allowed the user to run the pet feeding machine immediately would be developed in the pet feeding application. This included a “Feed now” button which sent a signal to the server to run the servo when the user clicked on it.

The feature *Scheduling the next feed* was also expected to be completed in this project. This feature included a “Schedule” button which would open a modal containing a date-time picker. This component would allow the user to pick the next feeding time. It should be possible for the user to save that time or cancel/ close the modal. When the user clicked on “Save”, a request would be sent to the server to trigger the machine to run at the specified time.

Feedbacks should be shown to allow the user to know if the action was successful. This was achieved by showing a modal or pop-up with the message received from the server.

The application must fulfil PWA requirements. This would be tested by using Lighthouse audit. Some PWA features were installability, offline use and responsiveness.

The application was expected to be a single page application. It means that no page refreshment would be required to load new content. Obviously, REST API should be taken advantage of to do that.

The UI of the application should be simple and responsive. Given that only two functionalities were developed, it would not be necessary to include eg. a menu bar or sidebar. However, it was required that the UI looks good and simple in different screen resolutions.

The servo should be strong and firm enough to manipulate the spinner of the food dispenser. It was also important that the servo would be attached to the food dispenser strong enough to make the whole machine stay stable. In reality, the food dispenser could be repaired so that the servo and Raspberry Pi were totally hidden behind it and out of the pet’s touch.

4 ARCHITECTURE AND DESIGN

4.1 System architecture

FIGURE 6 describes how this Pet Care System worked. As can be seen, running signals were sent by Raspberry Pi to control the servo and the food dispenser. Pi was a small size computer with Raspbian OS installed, so it could work as a web server. As a result, the client, which could be a web browser or a PWA, could communicate with Pi by sending HTTP requests or responses.

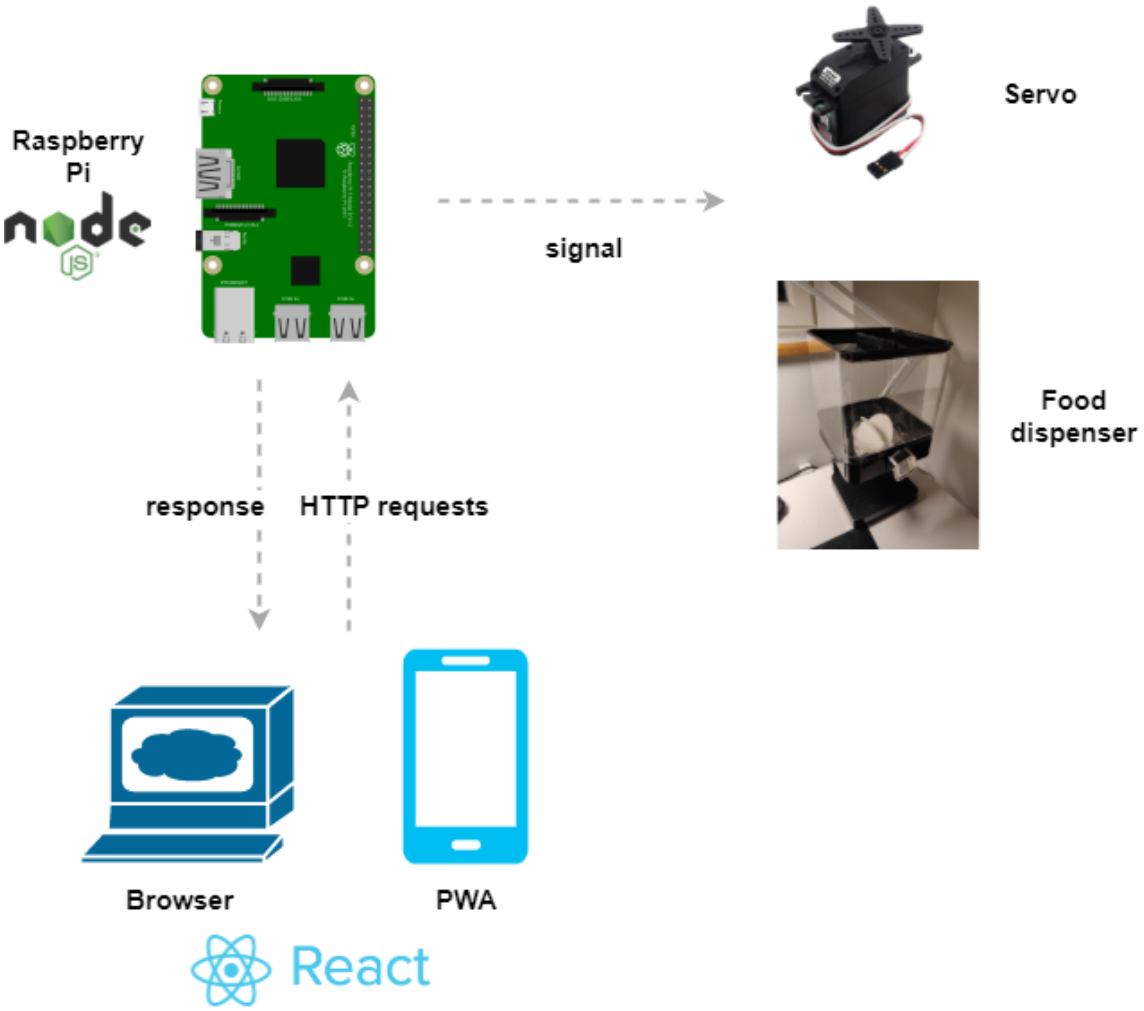

FIGURE 6. System's architecture

Client-side rendering was applied to this project and it means that the website was rendered on the client-side with a browser and JavaScript. The reason for that was the

scenario that the user would use a browser or mobile app to control the feeding machine remotely. Additionally, a single page application with client-side rendering was considered to be faster and more effective in this case. Client-side rendering made it possible for the client to only load the page once; the communication with the server after that was handled with AJAX.

In this project, the client and the server were made separately. The server was developed and saved in Pi via SSH. The logic for handling HTTP requests and controlling the servo was written in file app.js. FIGURE 7 shows app.js location in a React application project.

On the other hand, the client was a React.js application and it was developed in the author's computer before it was exported as a production version and then transferred to a host. In the React.js project, most logic was written in src/App.js.

FIGURE 7. React.js app project structure [17]

4.2 UI design

Two UI mockups were drawn to show the author's design ideas. The first one (FIGURE 8) showed the main screen of the application. Two buttons with different levels of importance were shown. "Feed now" was the primary button, while "Schedule" was the secondary button. A pet's image was shown above the buttons. There could also be a header showing the pet's name above the pet's image.

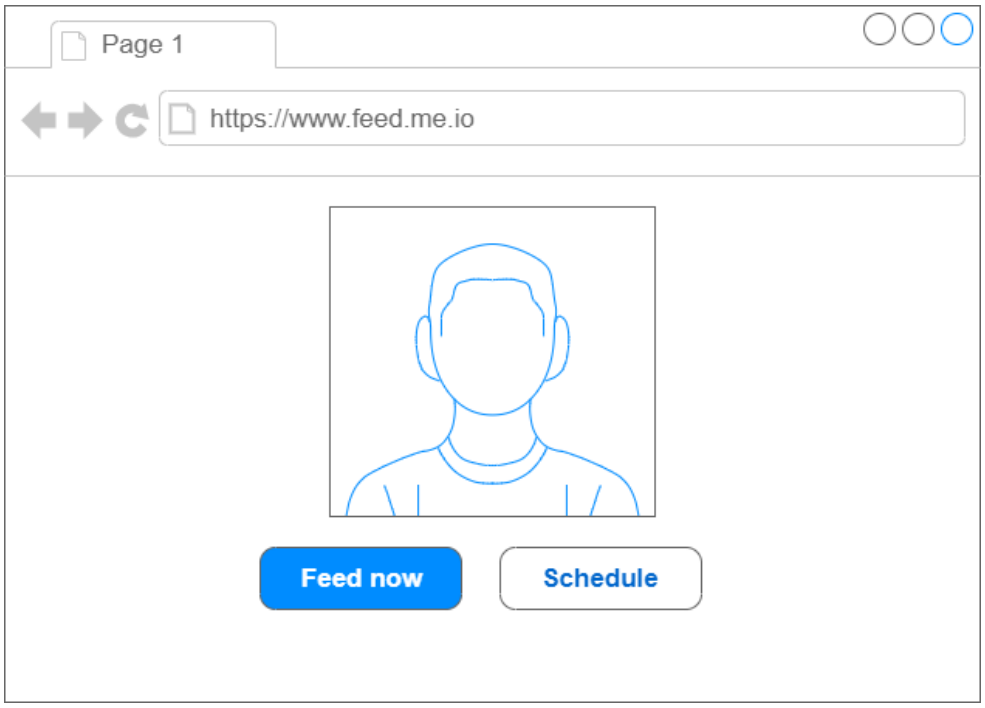

FIGURE 8. Main page mockup

The second mockup (FIGURE 9) showed what the user would see after she or he clicked on the “Schedule” button. A popup or modal would be shown and it overlayed the main screen. This popup would show a time picker and two buttons for saving and closing. Both these buttons should have colours based on the level of importance.

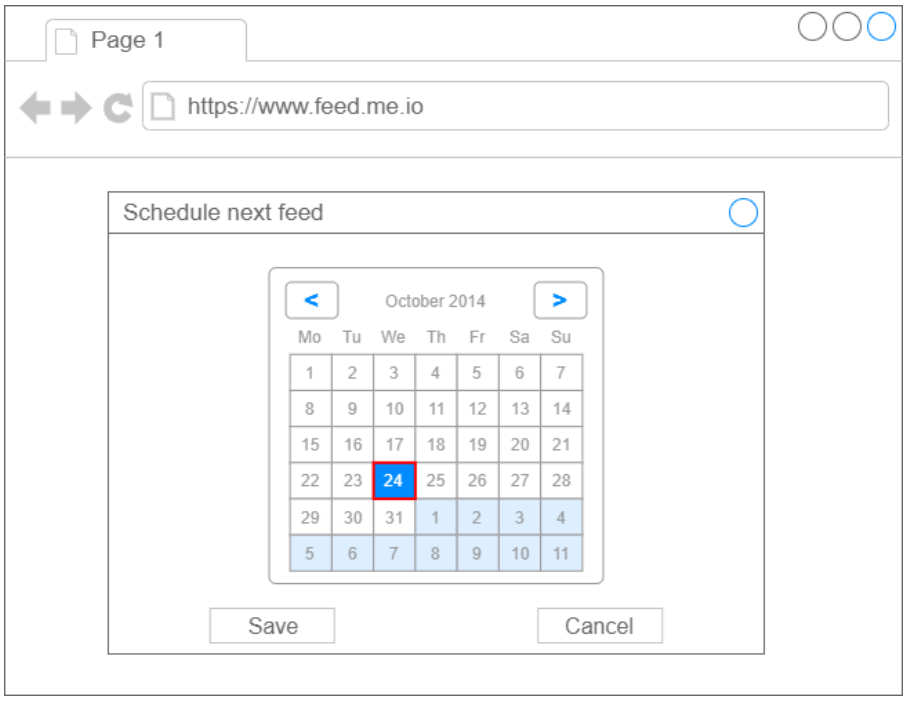

FIGURE 9. Date time picker mockup

5 PRODUCT DEVELOPMENT

5.1 Hardware installation

The first step was to power on Pi and connect it to the Internet by using an Ethernet cable. When Pi worked, a red light was shown. When Pi was connected to the Internet, a green light under the Ethernet port blinked.

As mentioned before, the servo included three leads which could be used to connect with Pi through some Male-to-female wires. It was important that these wires were plugged correctly to the specified GPIO pins on Pi. The red wire must be connected to pin 2 (5V pin), the yellow (signal wire) and black (ground) ones must be connected to pin 19 and 20. FIGURE 10 below demonstrates how the servo should be connected with Pi.

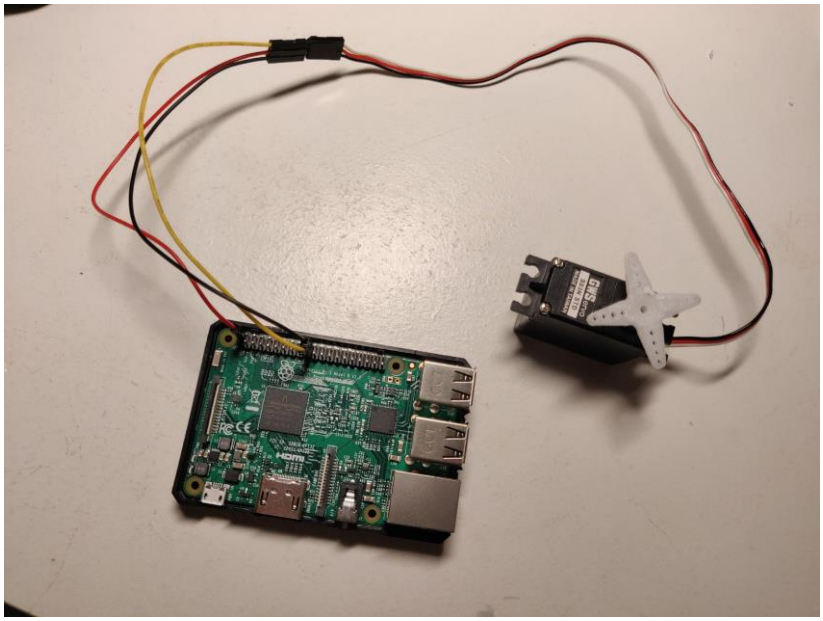

FIGURE 10. Raspberry Pi is connected with servo

The next step was to attach the servo to the food dispenser. The servo arm must be correctly attached to the food dispenser's button so that the centre of the servo arm would match the centre of the button. This is important because it made the servo stable and unmoved.

Another option for attaching the servo to the food dispenser was that the servo arm could be attached to the back of the food dispenser. As long as the centre of the servo arm and the centre of the silicone spinner were on the same axis, the movement would be consistent. FIGURE 11 shows how the servo and Pi are attached to the food dispenser.

FIGURE 11. Pet feeding device

5.2 Environment establishment

5.2.1 Backend

There are various ways of setting up a Raspberry Pi. The easiest way is to connect a monitor, a keyboard and a mouse with Pi. However, those components were not available when this prototype was developed. As a result, a headless setup was applied on this prototype. With this arrangement, Raspberry Pi was remotely controlled from the author's computer via SSH.

A prerequisite for SSH login was SSH activation in Raspbian. This was done by inserting Pi's SD card into a computer, then creating a file called "ssh" without an

extension. When the SD card was inserted back to Raspberry Pi, SSH would be enabled without difficulty.

For the environment testing purpose, a simple app.js file could be created with the sample code from the NodeJS official document. This file could be written in a PC and then transferred to Pi by WinSCP – a file transfer supporting software.

The first step of headless setup was to look up Raspberry Pi's IP address. Usually, Pi's address is decided by the router, so it could be easily found when the author logged in to the router's admin panel from the computer and checked the list of connected devices. FIGURE 12 shows the connected device list of the author's router. The second row in this FIGURE shows Pi's IP, which is what was needed in the first step.

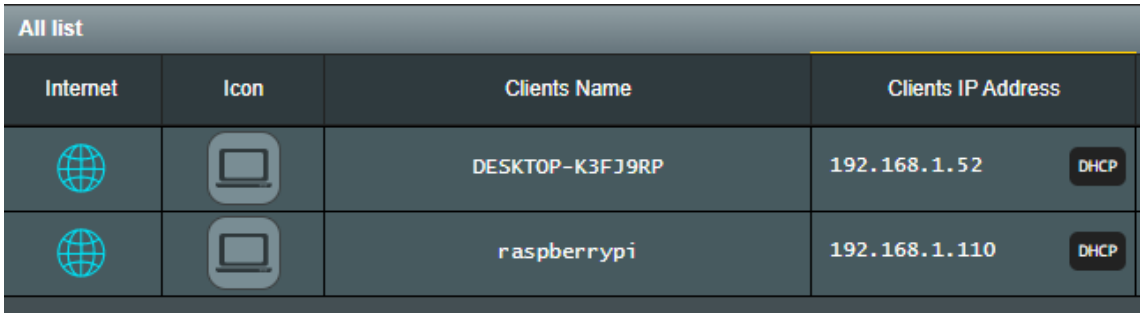

FIGURE 12. Router's connected device list

The next step was to use PuTTY – an SSH client – to log in to Pi. Pi's credential information was needed in this step. A welcoming message was shown when the author successfully logged in to Pi.

After accessing to Raspberry Pi, necessary tools for running the server and controlling the servo could be installed.

The first tool was NodeJS:

sudo apt-get install -y nodejs

In order to keep the server alive and avoid downtime, PM2 was used:

Sudo npm install pm2

Library Pigpio should be installed for servo manipulation:

Sudo npm install pigpio

All Raspberry Pi ports were closed by default; thus, it was necessary to open a port for public use. Therefore, Nginx could be used in the development as a Reverse Proxy Server:

sudo apt install nginx

After that, nginx's default configuration file was edited by using a command

sudo vim /etc/nginx/sites-available/default

This file was then saved and Nginx was restarted:

```
server {
        listen 80 default_server;
        listen [::]:80 default_server;

        root /var/www/html;

        index index.html index.htm index.nginx-debian.html;

        server_name _;

        location / {
            proxy_pass http://192.168.0.0.1:3000;
            proxy_http_version 1.1;
            proxy_set_header Upgrade $http_upgrade;
            proxy_set_header Connection 'upgrade';
```

```
            proxy_set_header Host $host;
            proxy_cache_bypass $http_upgrade;
        }
    }
```

Other utilities were installed including Express.js – Node.js web framework; cors – a Node.js module for cross-origin resource sharing and node-schedule – a Node.js scheduler:

Sudo npm install express

Sudo npm install cors

Sudo npm install node-schedule

5.2.2 Frontend

In order to quickly set up a development environment for the application, a *create-react-app* command was applied:

npx create-react-app my-app

Axios – a library which supports HTTP request creation – was installed.

Sudo npm install axios

React Bootstrap and React-datetime-picker were installed to enhance the user experience.

Sudo npm install react-bootstrap bootstrap

Sudo npm install react-datetime-picker

5.3 Feature development

5.3.1 Backend

The first step was to set up a working server. In order to do that, a js file was created for server code (app.js). All necessary modules were imported to this file via a function *require*. The basic structure of app.js is shown in FIGURE 13 below.

Whenever a client's connecting request is accepted, Express's method `app.listen()` binds the server and the client. After that, changes are detected and listened on the specified port and host. This method is the same with Node's *http.Server.listen().*

HTTP requests could also be made easily with the help of Express.js. For testing purpose, a simple HTTP request was created to show that the server was working and that it was able to respond to the client's request:

app.get('/', (req, res) => res.send('server is running'));

The author then browsed to the address http://localhost:8080/ and saw the message "server is running".

```
const cors = require('cors');
const express = require('express');
const schedule = require('node-schedule');

const http = require('http');
const hostname = 'localhost';
const port = 8080;
const app = express();

app.use(express.json());
app.use(cors());
app.get('/', (req, res) => res.send('server is running'));

app.listen(port, hostname, () => {
  console.log(`Server running at http://${hostname}:${port}/`);
});
```

FIGURE 13. Basic server code

The first two lines of FIGURE 14 shows how to use Pigpio's method Gpio to set up GPIO 10 to an output. This was important as Raspberry Pi's default mode of the GPIO was an input, which obviously could not control the servo.

The function *feedNow()* played an important role in this file because it manipulated the servo's movement. As mentioned in the theory part, a servo move is a result of the change in its pulse width. *feedNow()* made the servo work by looping the code block inside *setInterval()* every 500 milliseconds. This code block includes:

- *servoWrite(pulseWidth)* started servo pulses at 50Hz on GPIO 10 at the most anti-clockwise position (*pulseWidth* is 500).
- *if* statement increased the *pulseWidth* gradually to 2500 (most clockwise) and then decreased it.

```
const Gpio = require('pigpio').Gpio;
const motor = new Gpio(10, {mode: Gpio.OUTPUT});

let pulseWidth = 500;
let increment = 500;

feedNow = () => {
  setInterval(() => {
    motor.servoWrite(pulseWidth);

    pulseWidth += increment;
    if (pulseWidth >= 2500) {
      increment = -500;
    }
  }, 500);
}
```

FIGURE 14. Servo controlling code

Two routes were needed for two major features which were *Feed* and *Schedule*. In FIGURE 15, Express's methods *app.get()* was used for the feature *Feed* considering the context that the pet should be fed immediately. The method *feedNow()* in the callback function of this route would trigger the servo to run as soon as the route with the path "*/feed*" was called.

app.post() was used for the feature *Schedule* instead of *app.get()* because the feeding time needed to be provided by the user. This information should be stored in the HTTP request's body; therefore, the method POST was a suitable choice.

In the callback function of this POST route, a *Date* object was created from the provided time taken from the request body. This object was then used as a parameter in *scheduleJob()* . As mentioned earlier, *scheduleJob()* is a method of node-schedule which allows users to automatically create a *job* object and execute it at the specified time.

In both callback functions, *feedNow()* was called. The difference was that it was executed at a different time. The messages sent together with the response were also different.

```
app.get('/feed', (req, res) => {
  feedNow();
  res.send('Your pet has been fed!');
});
app.post('/schedule', (req,res) => {
  let time = new Date(req.body.time)
  schedule.scheduleJob(time, function(){
    feedNow();
  });
  res.send('Your pet will be fed at scheduled time!');
});
```

FIGURE 15. Server handles HTTP requests get and post

5.3.2 Frontend

First of all, all the necessary components and modules, such as React Bootstrap components *Row*, *Col*, *Button*, *Image* should be imported.

It is popular in many programming languages and frameworks that a constructor is the first part to be set up, so does React. As a rule, a constructor initializes the local state of the component; event handler binding also happens in the constructor. Below the screenshot (FIGURE 16) indicates how the constructor in App.js was made:

- *super()* called the constructor of the parent class, which was the constructor of *React.Component* in this case
- Property values of the state were set. *Feedback* was set to a string and it would be filled up with a meaningful sentence when the state changed. *isShowingFeed* and *isShowingSchedule* were used for showing or hiding the corresponding modals and they were set to false, which means that the modals were hidden by default. *Date* was a JavaScript date object, it would be used for recording the date which was selected by the user when a feeding schedule was set.

- Event handlers *handleShowFeed*, *handleCloseFeed* , *handleShowSchedule* and *handleCloseSchedule* were bound in the last four lines of the constructor.

```
15    constructor () {
16      super();
17      this.state = {
18        feedback: '',
19        isShowingFeed: false,
20        isShowingSchedule: false,
21        date: new Date()
22      };
23      this.handleShowFeed = this.handleShowFeed.bind(this);
24      this.handleCloseFeed = this.handleCloseFeed.bind(this);
25      this.handleShowSchedule = this.handleShowSchedule.bind(this);
26      this.handleCloseSchedule = this.handleCloseSchedule.bind(this);
27    };
```

FIGURE 16. app's constructor

The next step was to define the event handlers.

Regarding the first four lines of FIGURE 17, *handleShowFeed()* was called when the user clicked on “Feed now”. A promise was returned which then did two tasks if it was fulfilled: sent an HTTP request to the server so that it triggered the pet feeding device to run; then *feedback* was updated with a new message receiving from the server and *isShowingFeed* switched to a value *true*, which means that the modal containing the feedback would be shown. Axios was applied in this function to make it shorter and more effective.

handleCloseFeed() , handleShowSchedule() and *handleCloseSchedule()* are similar to each other. They triggered the modals to show or hide by changing the state. Regardless of modal visibility, *changeDate(),* on the other hand, received a parameter *date* and set a property *date* to the new value.

```
28    handleShowFeed = () => {
29        axios.get('http://192.168.1.110/feed')
30        .then(response => this.setState({feedback: response.data,
          isShowingFeed: true}));
31    }
32    handleCloseFeed = () => {
33        this.setState({
34            isShowingFeed: false
35        });
36    }
37    handleShowSchedule = () => {
38        this.setState({
39            isShowingSchedule: true
40        });
41    }
42    handleCloseSchedule = () => {
43        this.setState({
44            isShowingSchedule: false
45        });
46    }
47    changeDate = date => {
48      this.setState({ date });
49    }
```

FIGURE 17. Event handlers

As shown in FIGURE 18, the method *setSchedule()* was used to set time for the next feeding turn as well as giving feedback on the completed action. This function could send a request with the affix "*/schedule*" and parameter *time* collected from the state's property *date* to the server using the HTTP method POST.

As mentioned in the backend development part, the HTTP method POST was used for setting a schedule because the parameter *time* needed to be sent together with the request. Using POST was recommended in this case as it enhanced the security level of the server.

```
55    setSchedule = () => {
56      axios.post('http://192.168.1.110/schedule',
 •      {time : this.state.date}).then(
57        response => {
58          this.handleCloseSchedule();
59          this.setState({feedback: response.data,
 •          isShowingFeed: true})
60        }
61      );
62    }
```

FIGURE 18 Function setSchedule()

In React Bootstrap, the same grid system with Bootstrap was used. The grid system made the page responsive in different screen resolutions. It included a set of containers, rows and columns used for the content layout and alignment.

FIGURE 19 indicates how the main page of pet care application was created based on the React Bootstrap grid system. This page included a container containing three rows which were used for the page's title, pet's image and buttons. *Container*, *Row*, *Col*, *Image* and *Button* were built-in components that can be applied to develop this UI. FIGURE 20 shows the result of the codes in FIGURE 19.

As mentioned before, it is always faster and shorter to use these React Bootstrap built-in components instead of using the traditional Bootstrap, CSS, jQuery and React.js together. The reason for that is that React Bootstrap manipulates React's virtual DOM while jQuery events occur on the real DOM.

As shown in line 78 and 81 of the same FIGURE, event handlers, such as *handleShowFeed* or *handleShowSchedule* were easily called by using a built-in *onClick* event. In contrast, it would take more effort to trigger this event with jQuery since a selector, such as *$('button'),* must be chosen and a separated jQuery code block must be written.

```
66        <Container className="App mt-4">
67          <Row className="justify-content-center">
68            <h2 className="text-info">
69              My Dog
70            </h2>
71          </Row>
72          <Row className="justify-content-center">
73            <Col sm={6}>
74              <Image src={logo} thumbnail></Image>
75            </Col>
76          </Row>
77          <Row className="justify-content-center">
78            <Button variant="info" size="lg" onClick={this.handleShowFeed}
              className="my-3 mx-2">
79              Feed now
80            </Button>
81            <Button variant="outline-info" size="lg"
              onClick={this.handleShowSchedule} className="my-3 mx-2">
82              Schedule
83            </Button>
84          </Row>
85        </Container>
```

FIGURE 19. App's main page code

FIGURE 20. Application's main page

Two React Bootstrap's modals were made use of to show feedbacks and the time picker. Feedback is necessary in web applications as it tells the user whether the action is successful or failed. The time picker is for the user to pick the next feeding time. The

reason for using modals for these functionalities was the fact that the modal helped the user to focus on just the content inside them instead of other components on the application's main page. There could be buttons on the modal for closing or saving, which could be convenient and logical. It was also clearer that the modal had header – body – footer structure. FIGURE 21 shows how a React Bootstrap modal looks like.

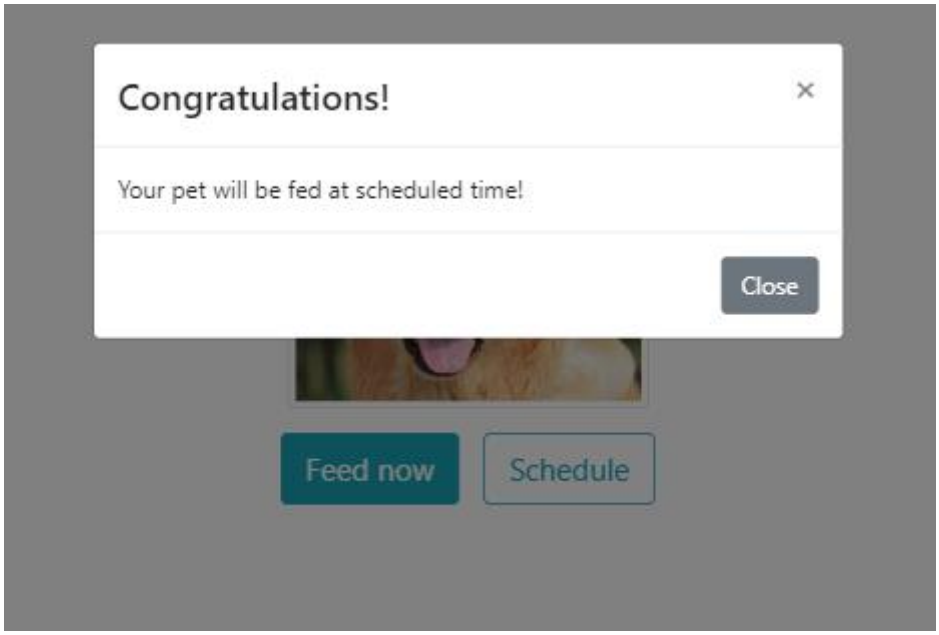

FIGURE 21. Feedback popup

The React Bootstrap modal has some built-in attributes for showing and hiding itself ,such as *show* and *onHide*. *show* can be set to either *true* or *false*, which corresponds to showing or hiding the modal. *onHide*, on the other hand, accepts a function and runs it when the event occurs. As shown in line 87 and line 93 of FIGURE 22, *handleCloseFeed()* was used with *onHide* and *onClick* events to change the value of the state property *isShowingFeed*, which made the modal disappear.

```
86          {/* Show success action */}
87          <Modal show={this.state.isShowingFeed}
            onHide={this.handleCloseFeed}>
88            <Modal.Header closeButton>
89              <Modal.Title>Congratulations!</Modal.Title>
90            </Modal.Header>
91            <Modal.Body>{this.state.feedback}</Modal.Body>
92            <Modal.Footer>
93              <Button variant="secondary" onClick={this.handleCloseFeed}>
94                Close
95              </Button>
96            </Modal.Footer>
97          </Modal>
```

FIGURE 22. Feedback modal codes

Similarly, the modal of feature *Schedule next feed* was shown and hidden in the same way with the feedback modal. As shown in line 105 to 108 of FIGURE 23, a simple *DateTimePicker* component from the React-datetime-picker could be applied in the Modal's body to build a time picker. This component was developed with a variety of attributes that could be beneficial for different puposes. In FIGURE 23, *DateTimePicker* detected changes in the input value and called a function *changeDate* to update the state. The HTTP request for setting a schedule was sent to the server only when the user clicked on the button "Save", which triggered the function *setSchedule*. FIGURE 24 shows the result of the codes in FIGURE 23.

```
99          {/* Show time picker */}
100         <Modal show={this.state.isShowingSchedule}
            onHide={this.handleCloseSchedule}>
101           <Modal.Header closeButton>
102             <Modal.Title>Schedule next feed</Modal.Title>
103           </Modal.Header>
104           <Modal.Body>
105             <DateTimePicker
106               onChange={this.changeDate}
107               value={this.state.date}
108             />
109           </Modal.Body>
110           <Modal.Footer>
111             <Button variant="secondary"
              onClick={this.handleCloseSchedule}>
112               Close
113             </Button>
114             <Button variant="primary" onClick={this.setSchedule}>
115               Save
116             </Button>
117           </Modal.Footer>
118         </Modal>
```

FIGURE 23. Time picker codes

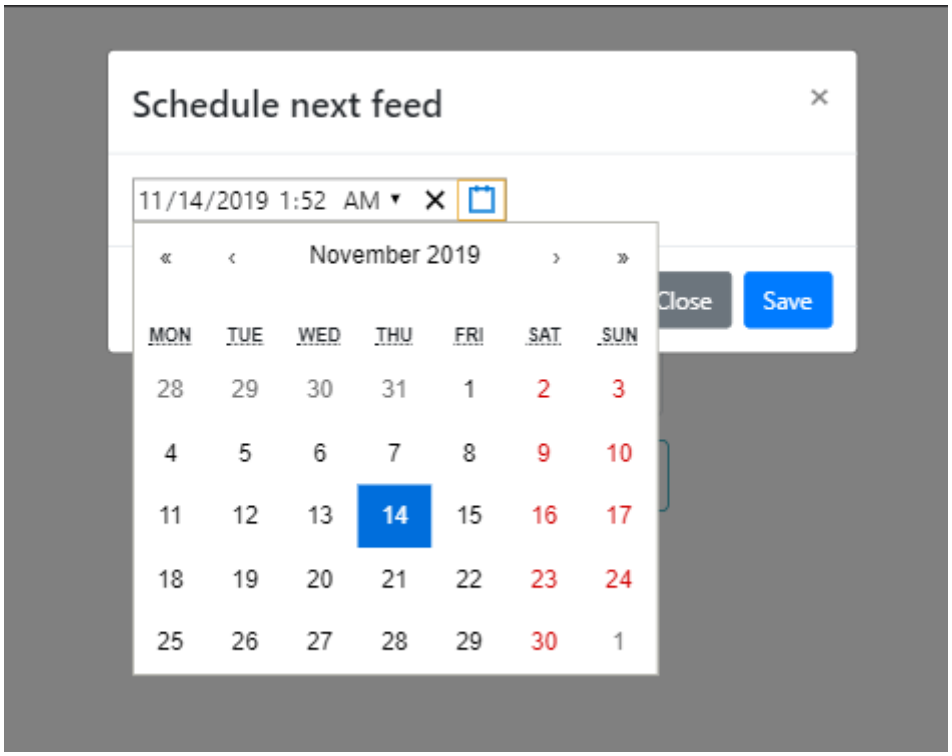

FIGURE 24. Time picker

The final task before exporting a production version for use was to make the application a progressive web app. Manifest.json should be modified as below to define the app's appearance (FIGURE 25).

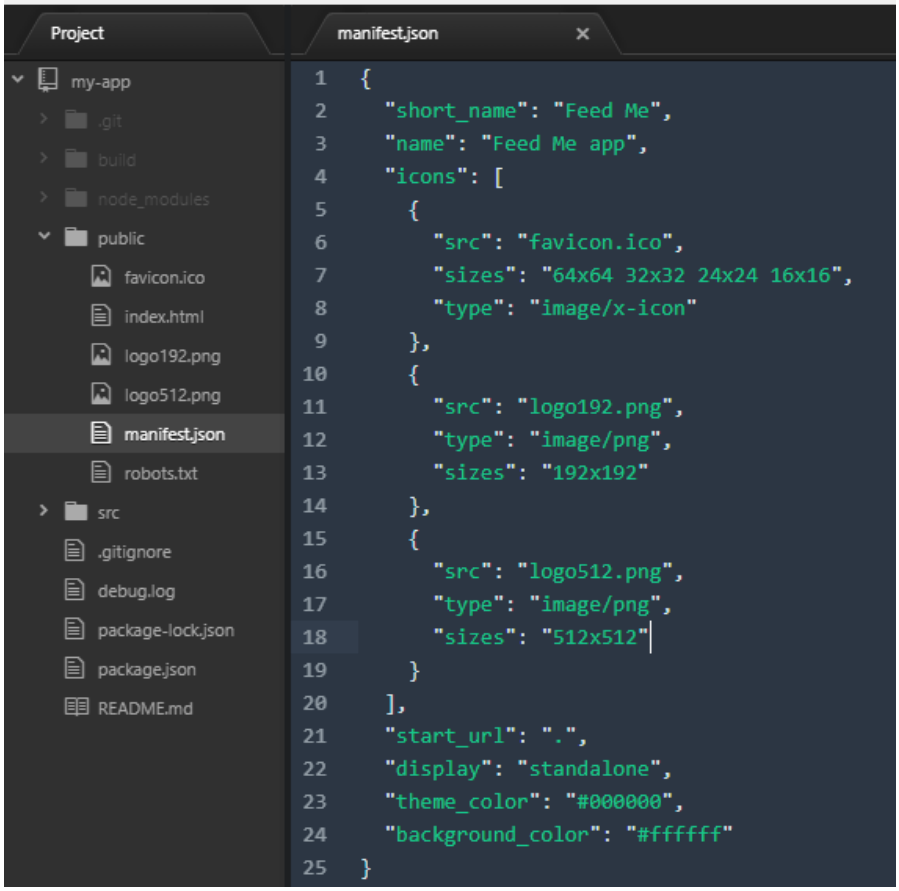

FIGURE 25. manifest.json

The browser would show an "Add to home screen" option if a manifest file existed. However, it should be linked in the public/index.html file with this first:

<link rel="manifest" href="%PUBLIC_URL%/manifest.json" />

A service worker is a script running in the browser's background and supports features that do not need a web page or user interaction. Those features may include push notification and caching, for example. React.js provides a file called serviceWorker.js in the folder *src/* which has preloaded code for handling service worker's lifecycle. It must be registered in index.js:

serviceWorker.register();

Finally, a command *sudo npm run build* could be used to export the final product.

Google's Lighthouse is a popular choice for performing a PWA audit. FIGURE 26 shows how the Lighthouse audit performed on the application. This was done by opening the browser's DevTools and running the audits with a "Progressive Web App" option. FIGURE 27 indicates that the web app could be installed.

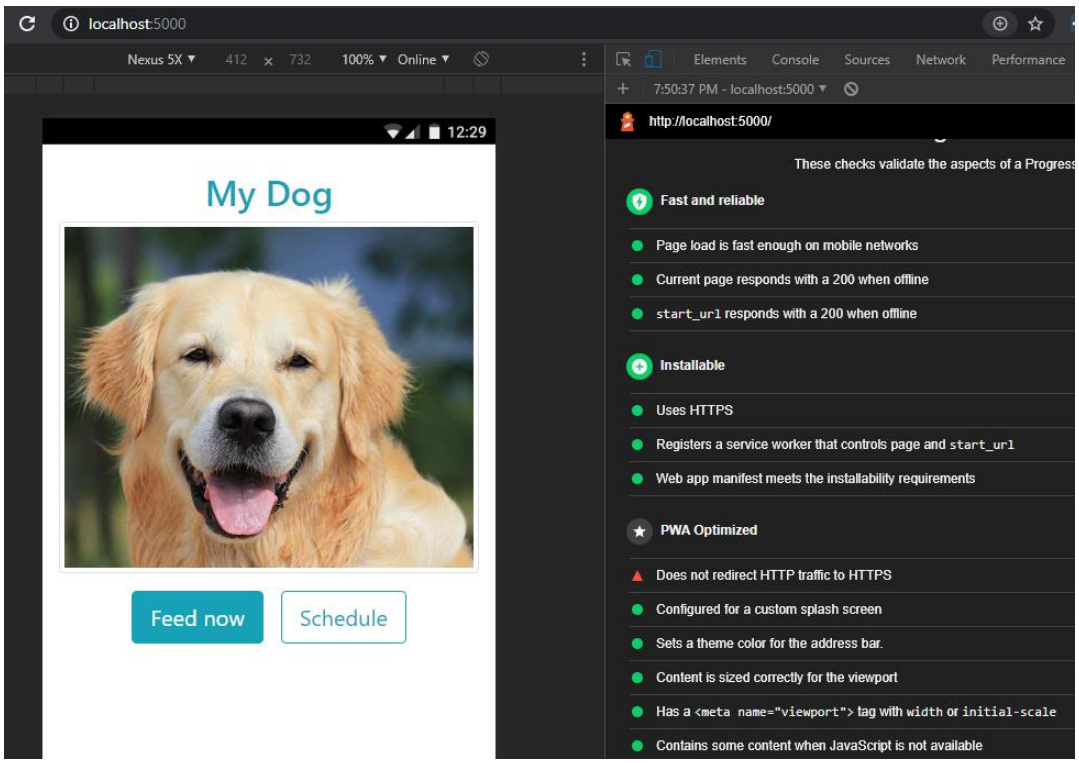

FIGURE 26. Lighthouse audit

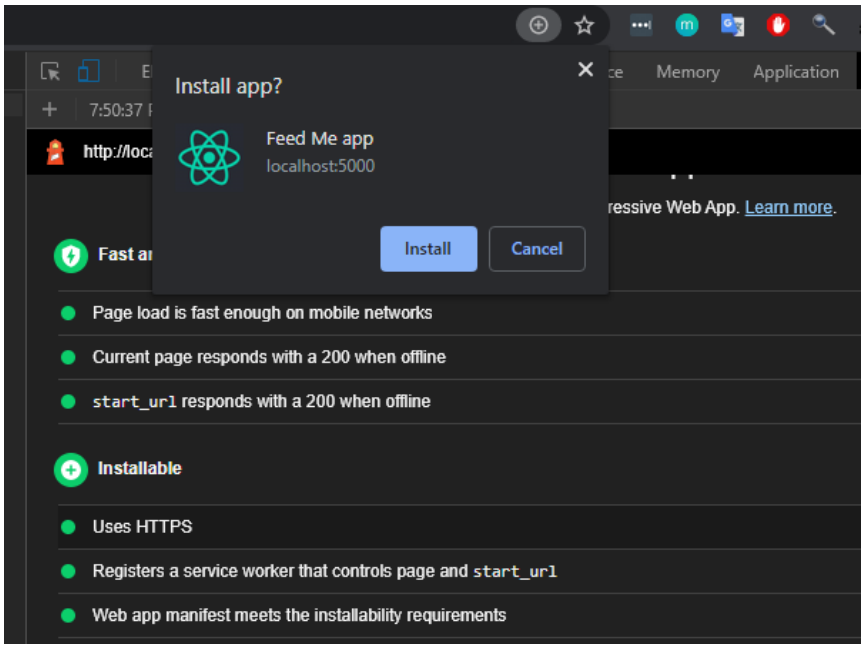

FIGURE 27. Web app meets installability requirement

6 EVALUATION

6.1 Project evaluation

When this project started, several challenges were highlighted. However, the author faced more difficulties than expected during the development phase.

For example, it was planned at the beginning that the food dispenser would be a hand-made product. Nevertheless, some components and materials could not be found. In addition, it was not easy to use tools to cut perfect pieces as more skills and machines were required.

It should not be forgotten that the project scope is limited, and the prototype therefore cannot be developed as complete as a production version. It means that, for instance, much attention to error handling was not paid. Features such as scheduling repeated feed were not developed.

Although there were a few difficulties and limitations, most project requirements were fulfilled and the final result is satisfying. The prototype was made and it worked perfectly in the local network.

An experience of trendy technologies such as React.js, Node.js and Raspberry Pi was gained. Non-IT knowledge was also acquired including mechanical and electronic skills.

Some errors happened during the development phase, but once they were solved, lessons were learned and knowledge was improved.

6.2 Evaluation of possible future development

First of all, a feature which allows the user to set a schedule for repeated feeds can be implemented. It should not be a heavy work because node-schedule has some built-in methods for recurrence rule scheduling which can be applied. For example, a method *RecurrenceRule()* can be used with options, such as *hour, minute*, to set the recurrence rule for the schedule job.

Another feature which can be developed is a live streaming camera system. This will be a helpful tool for pet owners to observe their pets and assure that their pets are healthy and eat well. In order to do this, a camera should be connected to Pi and a Raspberry Pi Camera module should be enabled. A Node module, such as *raspivid-stream,* can be applied to effectively achieve the purpose.

It is also possible to include a notification mechanism. For example, user can be reminded that the pet's vaccine appointment is coming soon. If this is the case, a database, such as MongoDB, may be necessary to save information. It is also essential to set up a connection between it and the NodeJS server.

It is even better if the user interface is re-designed as well as improved so that it becomes more beautiful and has more widgets: A customized date-time picker for the recurrence schedule, a separated thread for video streaming and a menu which includes user's account information, notification and settings.

The pet feeding device should be improved also. It is a fact that pet can be playful sometimes and he or she can destroy the device accidentally, so it is better to take this into account when installing the feeding device. The device can be put on a shelf out of the pet's touch and the food can go through a long pipe to reach the pet's bowl, for instance. A stronger glue and nails can also be applied for a better attachment.

7 CONCLUSION

The purpose of this project was to create a working Pet Care System with the most trendy technologies and analyze that approach. In addition, different skills could be practiced and improved.

In this thesis, the author's study on the applied technologies was revealed; the system design and implementation were described in detail and further development was also discussed.

During the project development, several problems arised out of expectation. A few of them could not be solved due to some objective factors. Although there were still some drawbacks in the prototype, most requirements were fulfiled and the final product worked perfectly.

The author encountered many difficulties, mostly because the technical stack was considerably new and the author did not have experience on these technologies. Nevertheless, valuable lessons were learned from those problems and difficulties; new knowledge was gained.

After the project, the author's interest with the IoT increased. It is also promising that more projects which combine the IoT and the web development will be smoothly developed in the future.

REFERENCES

[1] Raspberry Pi official website. What is a Raspberry Pi. 2019. Date of retrieval 10.01.2020 https://www.raspberrypi.org/help/what-%20is-a-raspberry-pi/.

[2] Wikipedia. Raspberry Pi. Date of retrieval 12.01.2020 https://en.wikipedia.org/wiki/Raspberry_Pi.

[3] Raspberry Insider. How Raspberry Pi is different from a desktop computer. 2019. https://raspberryinsider.com/how-raspberry-pi-is-different-from-a-desktop-computer/.

[4] Maker.io staff. How to pick the right Raspberry Pi. 05.02.2018. https://www.digikey.com/en/maker/blogs/2018/how-to-pick-the-right-raspberry-pi.

[5] Newark.com. Date of retrieval 15.01.2020 https://www.newark.com/raspberry-pi/raspberrypi3-modb-1gb/sbc-raspberry-pi-3-mod-b-1gb-ram/dp/77Y6520.

[6] R. P. official document R. P 3 model B. Date of retrieval 18.01.2020 https://www.raspberrypi.org/products/raspberry-pi-3-model-b/.

[7] Servocity. What is a servo. 2019. Date of retrieval 20.01.2020 https://www.servocity.com/what-is-a-servo.

[8] R. P. o. website. Grandpa scarer. 2019. Date of retrieval 22.01.2020 https://projects.raspberrypi.org/en/projects/grandpa-scarer/4.

[9] Pinterest. Data of retrieval 22.01.2020 https://fi.pinterest.com/pin/350506783496828638/.

[10] Jamsheer K, "ReactJS features," 22.08.2019. Date of retrieval 22.01.2020 https://acowebs.com/react-js-features/.

[11] Wikipedia. Single page application. 2019. Date of retrieval: 23.01.2020 https://en.wikipedia.org/wiki/Single-page_application.

[12] Npm trends. Date of retrieval 25.01.2020 https://www.npmtrends.com/angular-vs-react-vs-vue.

[13] NodeJS official document. 2018.Date of retrieval 25.01.2020 https://NodeJS.org/en/.

[14] Codebust.io. Date of retrieval 28.01.2020. https://codeburst.io/all-about-node-js-you-wanted-to-know-25f3374e0be7.

[15] Tutorialspoint. Node.js - Express Framework. 2019. Date of retrieval 29.01.2020 https://www.tutorialspoint.com/NodeJS/NodeJS_express_framework.htm.

[16] The Pi Hut. Turning on an led with your Raspberry Pi gpio pins. 2019. Date of retrieval 29.01.2020. https://thepihut.com/blogs/raspberry-pi-tutorials/27968772-turning-on-an-led-with-your-raspberry-pis-gpio-pins.

[17] Jpolanco. Hello world with ReactJS. 2018. https://steemit.com/utopian-io/@jpolanco/hello-world-with-react-js.

[18] Milton Abe. Demystifying React. Date of retrieval 30.01.2020 https://www.codemag.com/Article/1809041/Demystifying-React.