

Bachelor's

Degree programme in Information Technology

2022

Shanker Bhattarai

# A SMART PETFEEDER

Powered by PWA and Raspberry PI



BACHELOR'S | ABSTRACT

TURKU UNIVERSITY OF APPLIED SCIENCES

Degree programme In Information Technology

2022 | 39 pages

Shanker Bhattarai

## A SMART PETFEEDER

- Powered by PWA and Raspberry Pi

IoT devices are increasing massively in industrial areas and the general household. In addition, automation of the devices is making the lives of homeowners easier.

The main objective of this thesis is to combine the power of PWA and embedded systems and create a smart pet feeder. This petfeeder is an example of an IoT system in a home. The thesis covers the main aspects of creating PWA and goes through all the technology used to make it. The thesis also explains the use of raspberry pi and its importance in IoT for the project. The process of building the product is also described in this thesis, and that process can be divided into the following phases: planning, development, and testing. These phases are thoroughly presented in this thesis concerning the product.

The result of this thesis was a smart pet feeder which dispenses dry foods remotely through an application on the phone.

KEYWORDS:

Raspberry Pi, PWA, Flask, Petfeeder, IoT, Service Worker

# CONTENTS

|   |           |
|---|-----------|
| <b>LIST OF ABBREVIATIONS (OR) SYMBOLS</b> | <b>5</b>  |
| <b>1 INTRODUCTION</b>                     | <b>1</b>  |
| 1.1 Background                            | 1         |
| 1.2 Scope of project                      | 2         |
| 1.3 Objective of the project              | 2         |
| 1.4 Thesis Structure                      | 3         |
| <b>2 TECHNOLOGICAL OVERVIEW</b>           | <b>4</b>  |
| 2.1. PWA                                  | 4         |
| 2.1.1 Web App Manifest                    | 4         |
| 2.1.2. Service worker                     | 5         |
| 2.2 Embedded system                       | 6         |
| 2.2.1 Embedded System Hardware            | 7         |
| 2.2.2 Embedded System Software            | 7         |
| 2.3 Secure Web Hosting                    | 7         |
| 2.4 IoT                                   | 8         |
| 2.5 Flask                                 | 9         |
| <b>3 PRODUCT DEVELOPMENT</b>              | <b>11</b> |
| 3.1 Planning                              | 12        |
| 3.2 Development                           | 18        |
| 3.2.1 Development of PWA                  | 18        |
| 3.2.2 Python Script to run the motor      | 22        |
| 3.2.3 Creating Flask Server               | 23        |
| 3.2.4 Integration of all the processes    | 25        |
| 3.2.5 Testing                             | 30        |
| <b>4 CONCLUSION</b>                       | <b>31</b> |
| <b>REFERENCES</b>                         | <b>33</b> |

## FIGURES

|  |    |
|--|----|
| Figure 1. An example of the Manifest file.....                     | 5  |
| Figure 2. Working process of service worker .....                  | 6  |
| Figure 3. An example of Simple flask app.....                      | 9  |
| Figure 4. Gpio Pins in raspberry pi.....                           | 13 |
| Figure 5. A working principle of standard DC motor.....            | 14 |
| Figure 6. L298N Motor Driver.....                                  | 15 |
| Figure 7. Raspberry Pi schematic.....                              | 16 |
| Figure 8. Picture of the Petfeeder .....                           | 17 |
| Figure 9. Code for service worker registration .....               | 20 |
| Figure 10. Code for installation of service worker .....           | 20 |
| Figure 11. Code for caching the files. ....                        | 21 |
| Figure 12. Code for activation of installed software .....         | 21 |
| Figure 13. Code Snippet to run motor.....                          | 22 |
| Figure 14. Flask server .....                                      | 24 |
| Figure 15. App structure for flask server .....                    | 25 |
| Figure 16. The main function app.py.....                           | 27 |
| Figure 17. A full-screen view of index page .....                  | 28 |
| Figure 18. View of the index page on a small-screened device ..... | 29 |

## **LIST OF ABBREVIATIONS (OR) SYMBOLS**

| Abbreviation | Explanation of abbreviation (Source) |
|--------------|--------------------------------------|
| CPU          | Central Processing Unit              |
| CSS          | Cascading Styles Sheets              |
| DC           | Direct Current                       |
| GPIO         | General Purpose Input/Output         |
| HTML         | Hyper Text Markup Language           |
| HTTPS        | Hyper Text Transfer Protocols        |
| IoT          | Internet of Things                   |
| JSON         | JavaScript Object Notation           |
| LED          | Light Emitting Diode                 |
| PWA          | Progressive Web App                  |
| WSGI         | Web Server Gateway Interface         |

# 1 INTRODUCTION

## 1.1 Background

Owning a pet is a widespread thing in present society. Some people often keep a pet for fun or style, but for some people, there are more serious reasons, such as emotional support.

According to a survey by Euromonitor and Nestle Purina Petcare(Sivewright, Americas and Krueger, n.d.), around 471 million dogs and 373 million cats were kept as pets worldwide in 2018.

The trend of keeping pets is increasing worldwide. However, due to a lack of knowledge or ignorance, pets suffer from a high risk of health problems. One of those main problems is obesity. Obesity not only takes a heavy toll on the pets' health but also brings financial troubles for the owners in the form of medical costs. According to Association for Pet Obesity Prevention (APOP) research in 2018, 60% of cats and around 56% of dogs in the United States are classified as obese. (2018 — Association for Pet Obesity Prevention, 2018).

Because of the excessive fat in pets, they suffer from various health conditions. Obesity is the leading cause of most common health conditions such as Asthma, Diabetes, or even many forms of cancer. They also bring forth high blood pressure and liver failure and may result in heart failure.

There could be multiple factors that cause obesity in pets. However, among those factors, feeding habits such as unsupervised feeding, overfeeding, high-calorie food, and feeding too much as a treat can be the leading cause. So, by default, if we can solve this problem, we do not just prolong the life of our friends but also

save ourselves from unnecessary medical bills. Thus, the product from this project aims to find a solution by adding convenience.

This project is dedicated to making a smart pet feeder. Smart PetFeeder is a bright solution for any household aiming to solve the presented problem. The problems are tackled using the embedded system and IoT's help.

## 1.2 Scope of project

Having a pet is a hassle, mainly when the owner stays away from home most of the day. A smart petfeeder is an ideal product for owners with hectic schedules. With the help of this product, the owner can pour the desired amount of dry food at a specific time remotely. It can prevent the pet from getting extra dry foods and thus helps in improving health conditions such as diabetes, obesity, and renal diseases. It also allows the owner to check the live feed from anywhere at any given time, allowing the owner the facility to monitor them. Business aspects of the project are not included in this thesis.

## 1.3 Objective of the project

The smart petfeeder is a food dispenser that controls the amount of food poured using any smartphone. The Pet feeder is powered by Raspberry Pi and is controlled remotely using PWA. It includes a feeding container, which contains dry food. The container holds around 1 kg of average dry food. The project targets small pets such as cats, so a small container is used. Raspberry pi is connected to the internet. The owner then controls the container remotely using a smartphone with a dedicated PWA. It also contains a camera showing the activity of the pet. The live feed can also be viewed from the same app.

## 1.4 Thesis Structure

In this thesis, chapter 1 deals with the introduction to the project, its motivation, and the details of the thesis itself. Chapter 2 focuses mainly on explaining the technological concepts used to complete the project, followed by chapter 3, which is dedicated to product development. This chapter clearly explains the development process from planning to testing.

Chapter 4 covers the result of the project. It also documents all the discussions that were obtained. Moreover lastly, Chapter 5 covers the conclusion of the thesis, product development, and the results.



## 2 TECHNOLOGICAL OVERVIEW

### 2.1. PWA

As the name suggests, PWA or Progressive webapp combines web application and native application. Meaning that PWA utilizes the functions of web technologies with a combination of native features such as camera, locations, and even push technologies. Unlike traditional web apps, these PWA can load when there is no network connection and give the users a native-like experience. PWA should fulfill the following technical requests to function correctly:

- Valid secure HTTPS connection
- Valid Installed JSON manifest
- Installed Service Worker

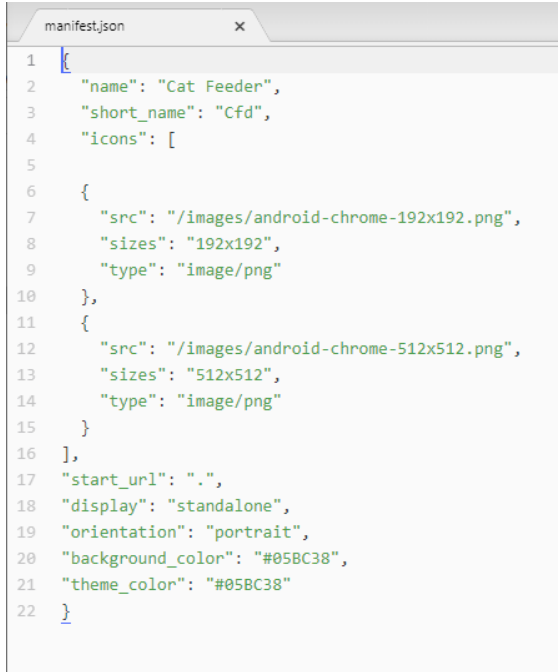
In addition to the points mentioned, PWA requires various tools to function correctly. These tools are technologies supported by browsers.

So, if the browser does not support these tools, they still function correctly as a standard webapp. Such technologies are listed below:

#### 2.1.1 Web App Manifest

A web app manifest is a JSON file responsible for making a PWA installable on a user's device. It also allows the apps to launch in full-screen mode and provides control of screen orientation. It gives information about PWA to the browser about how it should behave when installed on any desktop or mobile device by providing a near-native application experience.

The manifest file usually consists of information such as name, short name, icons, display, background color, theme color, etc. A fine example of a web app manifest file is shown in Figure 1.

A screenshot of a code editor window titled 'manifest.json'. The code is a JSON object with the following structure: an opening curly brace, a 'name' property with value 'Cat Feeder', a 'short\_name' property with value 'Cfd', an 'icons' array containing two objects. The first object has 'src' as '/images/android-chrome-192x192.png', 'sizes' as '192x192', and 'type' as 'image/png'. The second object has 'src' as '/images/android-chrome-512x512.png', 'sizes' as '512x512', and 'type' as 'image/png'. After the icons array, there is a comma, a 'start\_url' property with value '.', a 'display' property with value 'standalone', an 'orientation' property with value 'portrait', a 'background\_color' property with value '#05BC38', and a 'theme\_color' property with value '#05BC38'. The file ends with a closing curly brace. Line numbers 1 through 22 are visible on the left side of the editor.

```
1  {
2    "name": "Cat Feeder",
3    "short_name": "Cfd",
4    "icons": [
5
6      {
7        "src": "/images/android-chrome-192x192.png",
8        "sizes": "192x192",
9        "type": "image/png"
10     },
11     {
12       "src": "/images/android-chrome-512x512.png",
13       "sizes": "512x512",
14       "type": "image/png"
15     }
16   ],
17   "start_url": ".",
18   "display": "standalone",
19   "orientation": "portrait",
20   "background_color": "#05BC38",
21   "theme_color": "#05BC38"
22 }
```

Figure 1. An example of a Manifest file.

### 2.1.2. Service worker

Service Worker is also an essential building block while creating a PWA. It is a JavaScript file that the browser runs in the background, separate from a webpage, opening the door to features that do not need a web page or user interactions. (Google Developers, 2016). Different parts of PWA can be selectively cached using a service worker to provide an offline experience. Even if the browser does not support the service worker, it works as a regular webapp. Service workers give the user total control of each request made from the site, which opens the possibility for many different use cases. (Dean Alan Hume and Addy Osmani, 2018)

Usually, JavaScript files are attached to an HTML page and executed in a single thread. A single HTML file may include various JavaScript files, but all files run in a single thread. These Javascript files can access DOM(Document Object Model) to bring changes to the websites and execute various actions using the DOM. DOM can be defined as the platform or interface that allows scripts to update a document's content, structure, and style. (www.w3schools.com, n.d.).

On the other hand, the service worker is a JavaScript file that does not execute on the same thread. It runs separately from the regular thread and can intercept network requests. Service workers are not tied to any webpage and cannot modify any elements in a webpage. Figure 2 illustrates the work of the service worker.

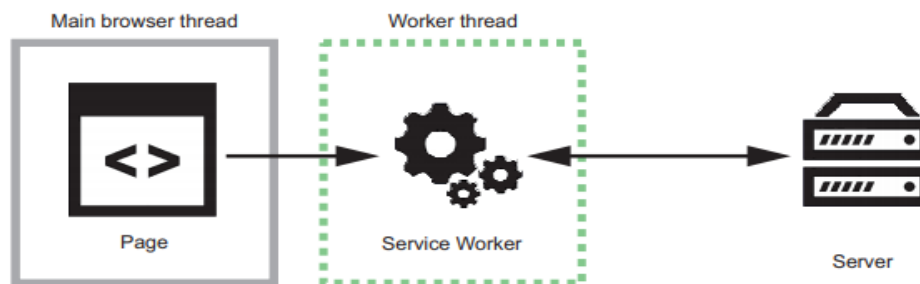


Figure 2. Working process of the service worker

Source: Dean Alan Hume and Addy Osmani (2018). *Progressive web apps*. Shelter Island, NY: Manning Publications.

## 2.2 Embedded system

A combination of hardware and software developed for a specific function is known as an embedded system. These systems can either be stand-alone or part of a more extensive system that includes various hardware, such as electrical and mechanical components. Although a part of a more comprehensive system, they are dedicated to operating one or two specific functions. An automatic car window mechanism is an example of an embedded system belonging to the more extensive procedure.

Since the embedded system is designed to perform a specific task only, the design engineers can optimize the size, cost, power consumption, reliability, and performance. (Techopedia.com, n.d.).

### 2.2.1 Embedded System Hardware

Embedded system hardware is built around microcontrollers or microprocessors. A microcontroller is a processor with integrated memory, whereas microprocessors are CPUs that use external chips for memory and peripheral interfaces. (Gregory, 2020). Embedded system hardware includes components such as User Interface, Memory, and Power supply.

Raspberry pi is used as a microprocessor for this specific project, and various other hardware such as DC motor, Motor Driver, and Resistors.

### 2.2.2 Embedded System Software

Embedded System Software is a set of codes used to control the microcontrollers or microprocessors. These codes are dedicated to operating simple calculations and basic functionalities. Embedded software is less memory-intensive because the hardware has limited computing capabilities.

This project uses raspberry pi as hardware, so Linux os is used as software, and python is used as a language.

## 2.3 Secure Web Hosting

The website comprises various content such as HTML, CSS, and images. Putting these contents online so that other users can access them is called web hosting. In general terms, Web hosting refers to deploying websites on the internet.

A server is needed to host a website. A server is a computer that connects other users to our website from the whole world via the internet. The server contains the files responsible for creating a website and making it viewable online.

Various things should be considered while hosting a website. They depend on multiple factors, such as the host's requirements, website type, and price ranges. However, security is the most common thing to be cautious about while hosting

a website. Various website hosting company gives reliable security service for a reasonable price, but it would be a hassle to serve a host personally. Furthermore, various security measures should be followed to host a website from home, as it exposes the whole network system.

The network must be secured with an HTTPS connection to run a PWA from its server. It provides security and also creates a sense of trust among the users.

One must get the proper certification and a public IP to get an HTTPS connection. This part is not included in this thesis, as an HTTPS connection is not required for PWA to work in the same network.

## 2.4 IoT

IoT, or the internet of things, is the network of physical objects that contain embedded technology to communicate, sense or interact with the internal states or the external environment. (Gartner, n.d.). Embedded devices have sensors that collect data from the environment. These devices can capture and share data by getting connected through the internet. Depending on the system, either the software processes and sends the alert or adjusts the system, or the user makes adjustments required in the user interface. A fine example of IoT is an innovative home system. The system might include an automatic burglar system that alerts the owner through the internet or a simple, smart coffee maker that uses voice commands and sends the user notifications when coffee is ready.

## 2.5 Flask

A framework can be defined as a standard platform to build and deploy the application. It provides a foundation for the developers to build programs, as they include the predetermined function and classes. Flask is a python-based lightweight WSGI web application framework. Flask provides the tools and technologies to build various web applications. Flask is a micro-framework, which means they are light; there are few dependencies to update and watch for security bugs (pymbook.readthedocs.io, n.d.a). On the other hand, it means we might have to increase the dependencies by adding plugins.(pymbook.readthedocs.io, n.d.b)The dependencies of Flask are:

Werkzeug,a WSGI utility library

Jinja2, a template engine

Figure 3 below shows an excellent example of a FLASK app that shows Hello World once the user runs the server.

```
1  from flask import Flask
2  from flask import render_template
3
4  app = Flask(__name__)
5  @app.route('/')
6  @app.route('/index')
7  def index():
8      return "Hello, World!"
9
```

Figure 3. An example of the Simple flask app

The detailed explanation of the codes in Figure 3 will be given in the later part of the thesis under the topic Development of Topic.

For this project, Plain Flask Server is used. Plain Flask server is an excellent way to demonstrate the project for prototyping. But a secure proxy (such as Nginx) should be used if the product is launched for commercial purposes.

### 3 PRODUCT DEVELOPMENT

Developing a product is a challenging process as it goes through many stages. Having an idea of a product is one thing, but implementing it with all the features required might create various difficulties. Product development is not a single process but a life cycle of different small stages. The stages of product development can be defined in these terms: Planning, Developing, Testing, Launching, Assessing, Repeating, or Killing. Any product development starts with the planning phase. It begins with recognizing a problem and finding ways to tackle it. Depending on the scale of the product, planning may be activated with a wide range of market research. It includes researching the competitors, understanding the market, and finding the key players. (Yoav Farbey, 2016a). After the study, the work is then distributed in small features. These features are defined using the user stories and their specifications, ranked by their difficulty and priority. Once all this is done, the second phase is commenced. This is the most crucial stage as it includes developing the product itself. Without this, the whole process would be some theory on paper. The product's features are designed according to the user stories and customer satisfaction. (Yoav Farbey, 2016b). This phase is then followed by testing. The testing of the product is done on various basis, which may range from simple scale to large one. Once testing is done, and the product is finalized, the tested product is launched in the market. Then the production is assessed by users' satisfaction and the product's reliability. Once the assessment is done, decisions are made on which features should be removed or upgraded. If the product did not perform satisfactorily in the market, it might also be discontinued. All these processes are compulsory for smooth product development product.



### 3.1 Planning

The essential part of product development is planning. It creates a blueprint for the products and lays out structural paperwork with milestones to achieve. The plan is designed to keep customer satisfaction in mind. A detailed, thorough plan helps the developer understand each feature and work accordingly.

The main idea of the thesis project is to create a raspberry pi powered petfeeder controlled using a PWA. The whole project is divided into five main parts. They can be vaguely divided as follows:

- Setting up hardware
- Development of PWA
- Software (Python Scripts for camera and GPIO pins)
- Create A Flask server to host the PWA
- Integration of all the process

The following materials are used to set up the device: Raspberry pi, 12V DC motor, A dry food container, shaft coupler, motor driver, DC power source, and various cables.

## Raspberry Pi

Raspberry Pi is a low-cost, small-sized computer that connects to a monitor or TV and uses a standard keyboard and mouse. (Raspberry Pi, 2013). It can do anything a traditional desktop can do and much more. A much different attachment can be put in this small computer, making it a powerful one. It was initially designed to make kids and adults more intrigued by the computer system and use various computer languages such as Python. A data storage device, such as a hard disk, is not included with Raspberry Pi. An external SD card is required to run the machine. This SD card should have a capacity of 4GB and higher.

In addition to traditional peripheral devices such as cameras or USB devices, they have additional GPIO pins along every device's edge. All GPIO pins can be configured as input or output and used for various applications. These pins can be configured either to receive or send signals. They operate by changing the voltage set too high(3V3) or low (0V). The voltage can be modified easily with the help of internal Pull-up or pull-down resistors. However, some pins, such as Ground pins(0V) and 5V pins, cannot be reconfigured and are fixed. Figure 4 shows the general layout of GPIO pins in any standard raspberry pi.

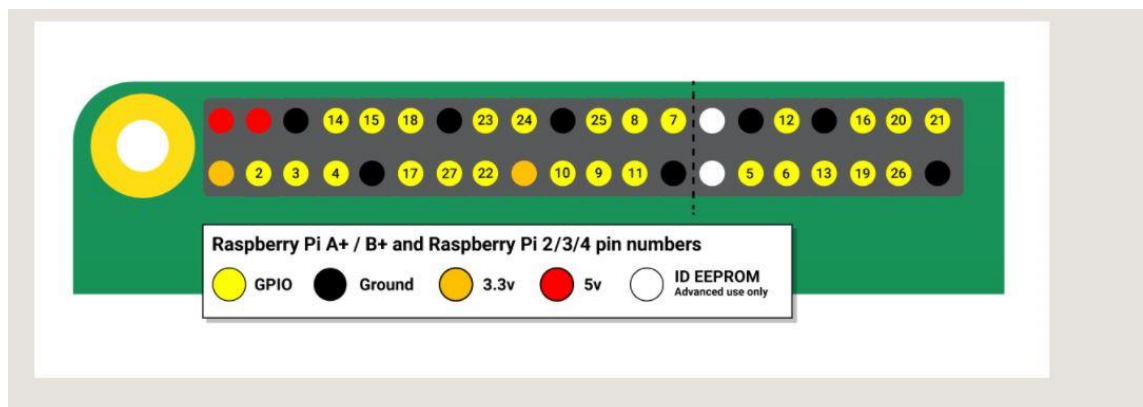


Figure 4. Gpio Pins in raspberry pi

Source: <https://www.raspberrypi.org/documentation/usage/gpio/>

For this specific project, the USB module for the camera of Pi is used. And the GPIO pins are used as output pins to send a signal to a DC motor.

### DC Motor:

In simple terms, a DC motor is an electric device that takes dc electricity and converts it into mechanical energy in the form of rotation. The motor has a fixed magnet called a stator, and a coil of wire called the armature. The armature is located right in the middle part of the stator. When the electricity is passed through the armature, it changes into an electromagnet and generates its magnetic field. However, there is a difference between the stator's magnetic field and armature. It results in torque and thus causes the armature to rotate.

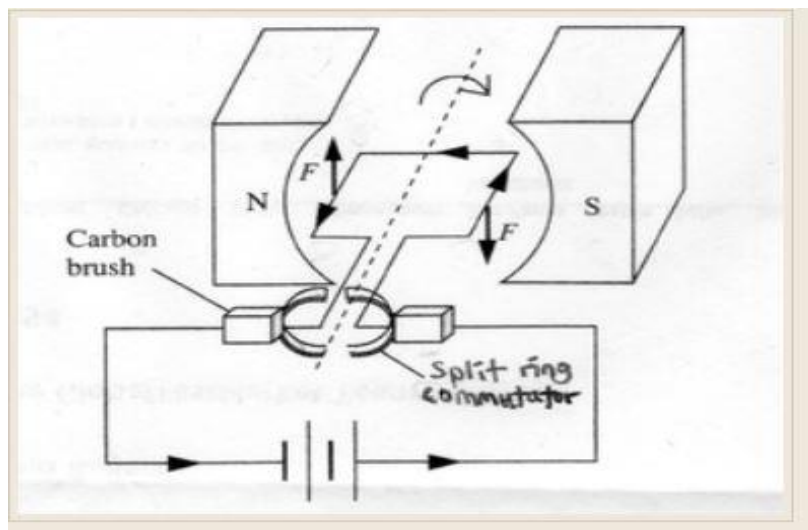


Figure 5. A working principle of standard DC motor

Source:<http://physicsf45spm.blogspot.com/2012/08/force-on-current-carrying-conductor-in.html>

A DC motor of 12 Volt with 35 Rotation per minute (RPM) is used for this project.

**L298N Dual H-Bridge Motor Driver:**

The motor driver is used for driving DC and stepper motor. The motor driver can control two or more motors in speed and directions. This feature is ideally suited to the Robotics field and can be easily controlled using microprocessors. The board uses relays, logic gates, etc., and LED to detect connections.



Figure 6. L298N Motor Driver

Source: <http://www.handsontec.com/dataspecs/L298N%20Motor%20Driver.pdf>

After this, the Development of the PWA would start. The PWA will contain two buttons. One will pour the food, and the other will turn on the camera and display the live feed on the user's phone. These buttons run a specific python script in raspberry pi, which acts accordingly. The PWA is hosted in the Raspberry pi using the Flask server. Finally, all the processes will be integrated to create a final product.

## Device assembly:

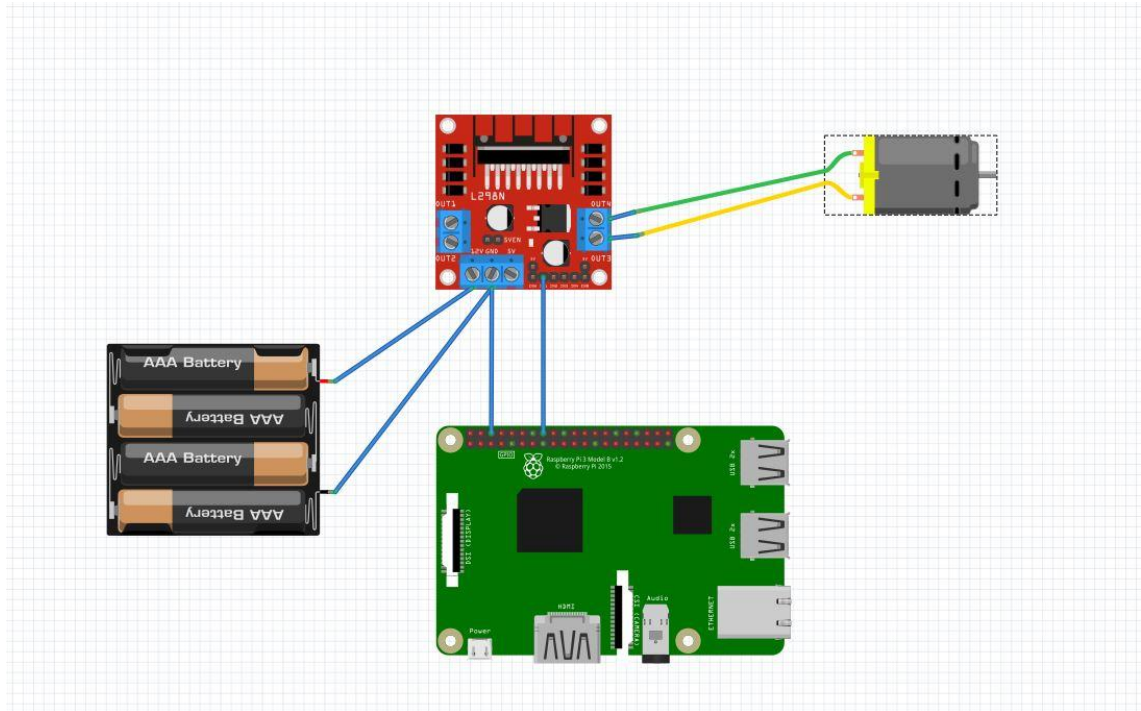


Figure 7. Raspberry Pi schematic

In figure 7, the assembly of the device is shown. First, pin 23 from raspberry pi is connected to input 1 of the motor driver. This pin 23 is marked as the output pin, responsible for sending a signal to the motor driver. Next, the two terminals of the DC motor are connected to the motor driver's output ports. 12-volt batteries are used as the power source for the motor as raspberry pi cannot provide enough power to run the engine. The positive terminal of the battery source is connected to a 12-volt input port, and the negative is connected to the ground of the motor driver. The ground of raspberry pi is connected to the same ground terminal too.



Figure 8. Picture of the Petfeeder

Figure 8 shows the final assembly of the hardware.

## 3.2 Development

### 3.2.1 Development of PWA

This chapter elaborates on the central concept of PWA. It also shows the tools used to create the PWA used for the project. The technologies used to implement the PWA for this project can be divided into sub-steps as follows:

#### **Adding Web app manifest file**

A web manifest file is added to the existing root folder of the project, where there is an index.html file. The manifest file always has JSON as its extension.

The app manifest file is linked to all HTML pages to ensure the presence of manifest properties in the entire application. It is linked to every page using an HTML link tag in the head segment.

```
<link rel="manifest" href="/manifest.json">
```

Manifest contains various elements. These elements are explained adequately with the example code used in this project as below:

- **Name:** It is the name given to the PWA and is visible on the splash screen
- **Short name:** It refers to the short name given to the app, which appears below the icon in homes.

```
"name": "Cat Feeder"
```

```
"short_name": "Cfd"
```

- **Icons:** Icons are a series of images of different scales used. They are stored in the form of JSON objects. The browser chooses the images compatible with the given device.

```
"icons": [  
  {  
    "src": "/images/android-chrome-192x192.png",  
    "sizes": "192x192",
```

```

    "type": "image/png"
  },
  {
    "src": "/images/android-chrome-512x512.png",
    "sizes": "512x512",
    "type": "image/png"
  }

```

- **Start URL:** It gives the pathname or URL to the first page when a user opens the application. "Index.html" is the starting page for this project.

```
"start_url": "."
```

- **Display:** It specifies the style of presentation for the application. There can be four display modes: standalone, Fullscreen, Minimal-UI, and Browser. The type of display varies from product to product. Standalone mode is chosen for the project.

```
"display": "standalone"
```

- **Orientation:** It shows the orientation of the application that is either portrait or landscape.

```
"orientation": "portrait"
```

- **Background color:** It is visible on a splash screen, providing a smooth transition from opening the application to starting the main page.

```
"background_color": "#05BC38"
```

- **Theme color:** It shows the central theme of the app.

```
"theme_color": "#05BC38"
```

- **Description:** It provides a short detail about the app later used by the Browser.

```
"description": "A pet feeding app."
```

- **Lang:** Lang is a string containing essential language for name, the short\_name, and description.

```
"Lang": "en-US"
```



## Adding Service Workers

The service worker file is a JavaScript file located in the root folder. Any service worker must go through three basic steps to function correctly. These steps are:

The service worker file is a JavaScript file located in the root folder. Any service worker must go through three basic steps to function correctly. These steps are:

- Registration:

The lifecycle of a service worker starts with its registration. The code below shows the process of registration of service workers. The code is saved as `app.js` in the root folder.

```

1  if('serviceWorker' in navigator){
2      navigator.serviceWorker.register('/sw.js')
3      .then(function(){
4          console.log('SW registered');
5      });
6
7  }
8

```

Figure 9. Code for service worker registration

In figure 9, the code checks if the service worker API is available or not. If the code checks out and the service worker API exists, the browser registers the service worker file. The file `sw.js` is always in the root folder.

- Installation

Once the service worker file is registered, it goes into the next phase, installation.

```

self.addEventListener('install', function(event) {
    console.log('SW Installed');

```

Figure 10. Code for installation of the service worker

We also cache our static or dynamic files inside the install callback function. Caching the files gives PWA an offline feature.

```
    caches.open('static').then(function(cache){
      cache.addAll([
        '/',
        '/index.html',
        '/app.js',
        '/manifest.json',
        '/styles.css',
        '/images/logo.png',
        '/Video.html'
      ]);
    })
  });
});
```

Figure 11. Code for caching the files.

In the Code snippet in Figure 11, we name our cache 'static' and cache all the static files in an array. `caches.open('static')` and `cache.addAll()` are chain of promises.

The service worker will be installed when all the files are appropriately cached. If there is an already installed service worker, it does not get through the installation process. However, even if there is a slight change in the service worker, the file is reinstalled when the page is reloaded.

- Activation

After the installation, the service needs to be activated. The activation is done using the code in Figure 12.

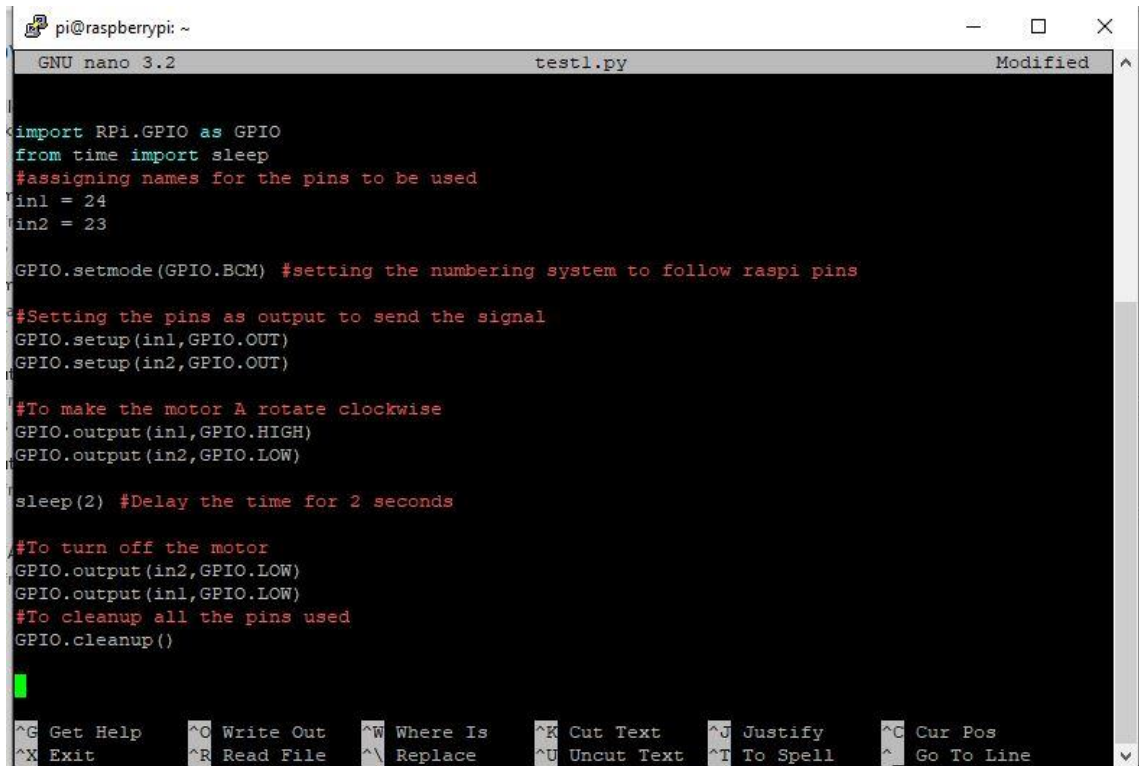
```
self.addEventListener('activate', function() {
  console.log('SW activated');
});
```

Figure 12. Code for activation of installed software

In the code in Figure 12, the function 'activate' activates the service worker. After this process, the service worker stays idle. A service worker is not activated until the opened tab is closed or reloaded.

### 3.2.2 Python Script to run the motor

Figure 13 shows the code that runs the 12 Volt motor for this project.



```

pi@raspberrypi: ~
GNU nano 3.2 test1.py Modified
import RPi.GPIO as GPIO
from time import sleep
#assigning names for the pins to be used
in1 = 24
in2 = 23

GPIO.setmode(GPIO.BCM) #setting the numbering system to follow raspi pins

#Setting the pins as output to send the signal
GPIO.setup(in1,GPIO.OUT)
GPIO.setup(in2,GPIO.OUT)

#To make the motor A rotate clockwise
GPIO.output (in1,GPIO.HIGH)
GPIO.output (in2,GPIO.LOW)

sleep(2) #Delay the time for 2 seconds

#To turn off the motor
GPIO.output (in2,GPIO.LOW)
GPIO.output (in1,GPIO.LOW)
#To cleanup all the pins used
GPIO.cleanup ()

^G Get Help      ^O Write Out    ^W Where Is     ^K Cut Text     ^J Justify      ^C Cur Pos
^X Exit          ^R Read File   ^\ Replace      ^U Uncut Text   ^T To Spell     ^_ Go To Line

```

Figure 13.Code Snippet to run motor

The code in Figure 13 utilizes the power of the motor board to control the direction and time of the rotation of the motor used. Once this code runs independently, it causes the motor to run for 2 seconds in the clockwise direction.

First, the pins in raspberry pi are chosen and connected to the motor driver's In1 and In2 ports. In the code in Figure 13, it is integers 23 and 24. Then the GPIO pins are set as an output to send the signal to the motor. After that, we set the output value of pin in1 to be high and in2 to be low. This setup makes the motor run forward or clockwise. The part of the code that says `sleep(2)` stops the motor running for 2 seconds. Then we set the value of in1 and in2 to be low, indicating the motor is at complete rest. Then we use the code `GPIO.cleanup` to clear all the values stored in the pins and make them ready for new values. The whole

function of the motor driver is not used in this project as it involves only one motor, and that motor is needed to rotate in only one direction. The integrating part of the project shows the proper placement and use of the code. In this thesis, chapter 3.2.4, integration of process shows the appropriate placing of the code.

### 3.2.3 Creating Flask Server

To create a flask server, first, it needs to be installed using the code:

```
pip install flask
```

A file named app.py is created after this command. The file contains the essential app for the flask server. Each line of code used in making the flask app for the project is explained below.

```
from flask import Flask
```

The code imports all the flask library and its underlying objects

```
from flask import render_template
```

render\_tremplate is a Flask function used to generate output from a template-based file based in the application's templates folder.

```
app = Flask(__name__)  
  
@app.route('/')  
  
@app.route('/index')
```

The code gives the URL to the main page called index.

```
def index():  
  
    return render_template('index.html',title='catfeeder')
```

It defines the function index, which renders the index.html page and returns to the URL/index. Here, the snippet of code above creates a simple flask server.

We must run the following command. The command only allows us to open the web page on the same computer.

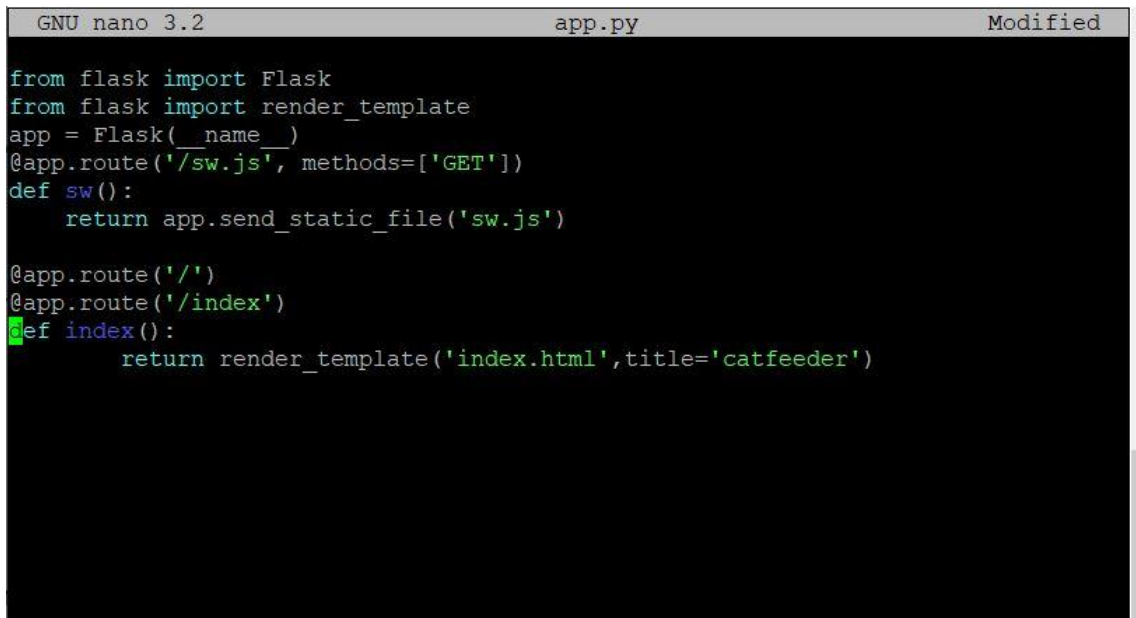
```
Export FLASK_APP=app.py
```

```
Flask run
```

These commands return the IP address, which is used to access the website served by the server. However, a user can only access this website on the raspberry pi it was built.

We must use a different command to run the flask on the external server. This command lets any user in the same network access the page created using the flask.

```
Flask run - host=0.0.0.0
```



```
GNU nano 3.2                                app.py                                Modified  
from flask import Flask  
from flask import render_template  
app = Flask(__name__)  
@app.route('/sw.js', methods=['GET'])  
def sw():  
    return app.send_static_file('sw.js')  
  
@app.route('/')  
@app.route('/index')  
def index():  
    return render_template('index.html', title='catfeeder')
```

Figure 14. Flask server

### 3.2.4 Integration of all the processes

This phase is the last phase of the development phase. In this phase, all the processes combine, resulting in the final product. First, the hardware is mounted as per the requirement.

This assembled hardware needs software to run. First, a flask server was created using the commands in point 3.2.3. The server started was simple, so we made some changes to make it PWA-friendly. Finally, the files and folders are arranged according to the structure mentioned in Figure 15.

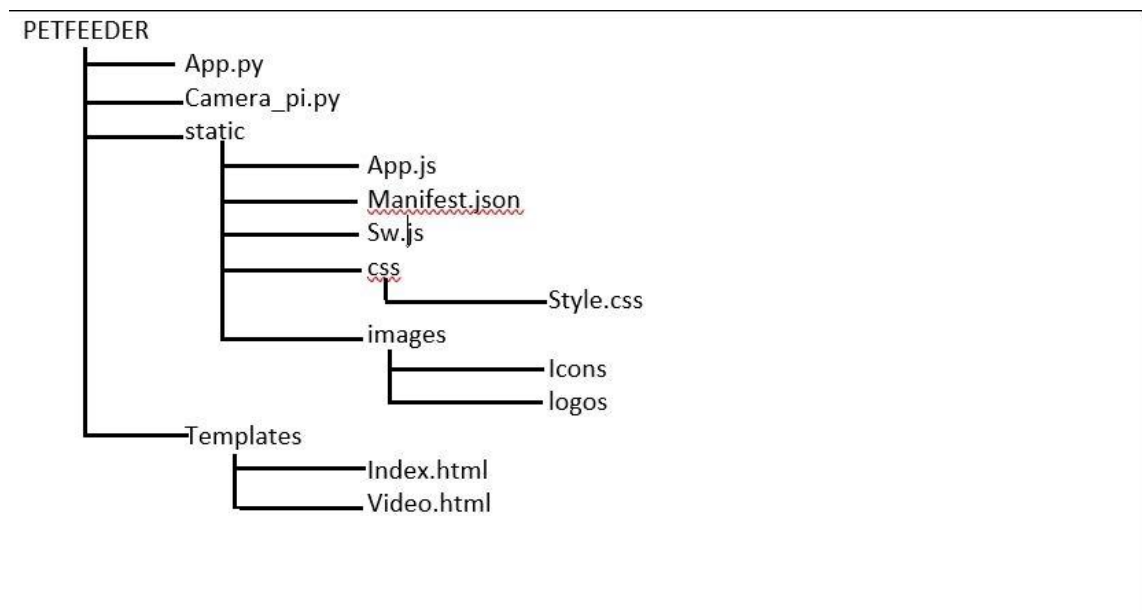


Figure 15. App structure for flask server

Figure 15 explains the folder structure of the PWA itself. Petfeeder is the main folder and holds all the files and subfolders. App.py is the main file that is responsible for running the flask server. It contains the commands and routes. The next one is camera\_pi.py which is the python program to control the camera of the raspberry pi. There is also a folder called static in petfeeder. This folder contains all the static files, such as JavaScript and CSS files. Sw.js, manifest and app.js files, essential to creating any PWA, are stored under the static folder. It also holds the images and other JavaScript files which do not need a python backend. This folder also has the subfolder images, where the logos and icons are situated.

The folder named "templates" holds all templates, one of the flask's main folders. Templates are files containing static data that are rendered to create the final document. For example, our template folder holds two HTML files, index.html and video the HTML.

After the structure was created, the function app.py was changed. This file, app.py, is the central server that directs to the homepage and video page.

As mentioned in the development of the PWA chapter, A PWA requires a service worker. Therefore, in the app.py, the service worker is added as code below:

```
@app.route('/sw.js', methods=['GET'])
```

This bit of code shows the root to the service worker using the get method.

```
def sw():  
  
    return app.send_static_file('sw.js')
```

This defines a function sw which returns the value of sw.js.

Then the necessary code to run the motor is added.

After adding the codes to the main function, It looks as in the code in Figure 16:

```

#Author Shanker Bhattarai

#Importing all the libraries
from flask import Flask, Response
from flask import render_template
from camera_pi import Camera
import RPi.GPIO as GPIO
from time import sleep

#Stating name for GPIO pin
in2 = 23
app = Flask(__name__)

@app.route('/sw.js', methods=['GET'])
def sw():
    return app.send_static_file('sw.js')

@app.route('/')
@app.route('/index')
def index():
    return render_template('index.html',title='catfeeder')

@app.route('/video')
def video():
    return render_template('video.html',title='watch your cat')

@app.route('/feed')
def feed():
    GPIO.setmode(GPIO.BCM)
    GPIO.setwarnings(False)
    GPIO.setup(in2,GPIO.OUT) #setting up the pin for output
    GPIO.output(in2,GPIO.LOW) # Pin is set to 0 so that it turns clockwise
    sleep(2) # delay of 2 seconds
    GPIO.cleanup()
    return render_template('index.html', title='catfeeder')

```

<sup>^</sup>G Get Help    <sup>^</sup>O Write Out    <sup>^</sup>W Where Is    <sup>^</sup>K Cut Text    <sup>^</sup>J Justify    <sup>^</sup>C Cur Pos  
<sup>^</sup>X Exit    <sup>^</sup>R Read File    <sup>^</sup>\ Replace    <sup>^</sup>U Uncut Text    <sup>^</sup>T To Spell    <sup>^</sup> Go To Line

Figure 16. The main function app.py

The code snippet in Figure 16 shows the full version of app.py used for this application. All the libraries are downloaded at the very beginning of the code. Then, the routes to the service worker and different pages and functions are created.

The flask server created in chapter 3.2.3 is modified by adding the code for GPIO pins responsible for running the motor. A PIN 23 of raspberry pi is named in2, then used in the `@app.route('/feed')`. This function is responsible for making the motor run, which is essential for this project.



The independent code of the running motor created in chapter 3.2.2 is used in this part of the file `app.py`. The function `feed` is activated when a button marked as “feed the cat” on the main page is pressed. After pressing the button, the function is activated, making the motor run for 2 seconds and stop.

The `index.html` page redirected by the app looks like this:

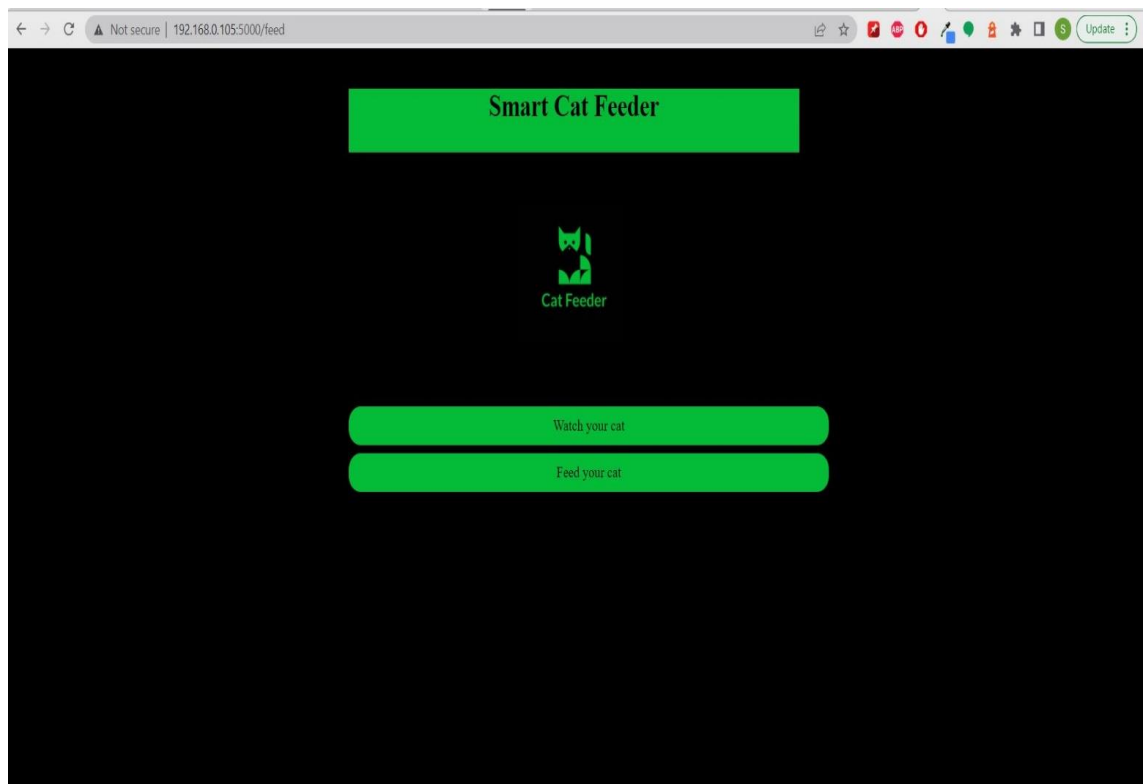


Figure 17. A full-screen view of the index page

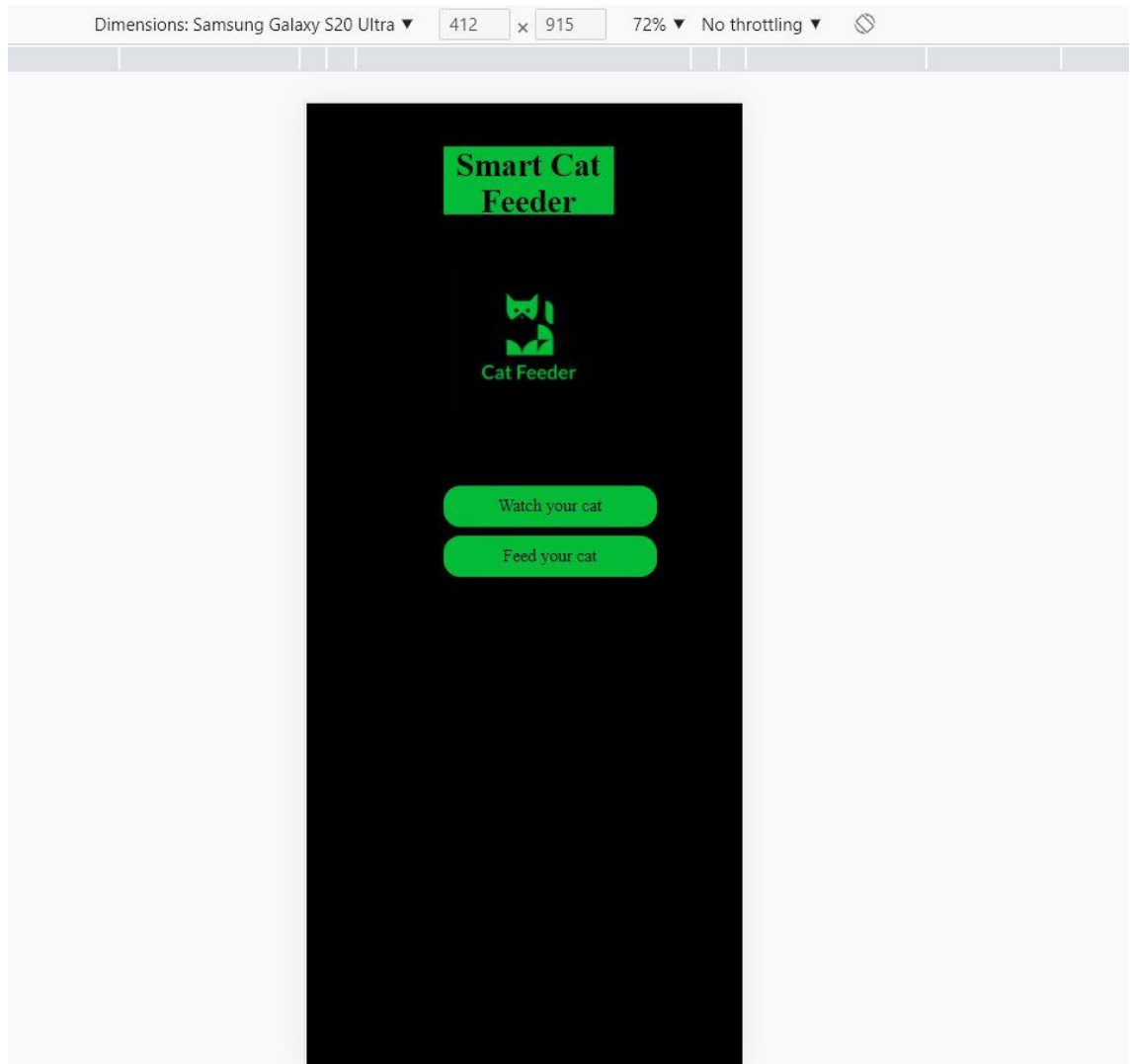


Figure 18. View of the index page in the small-screened device

### 3.2.5 Testing

Testing is the last phase of the development process. In this phase, the functions and durability of the product are tested. Testing determines whether the quality of the product was ensured during the development. In addition, it checks if there are any software bugs and hardware compatibility. This phase also answers whether the product created was able to satisfy the customer's needs and demands.

The initial plans made for the product are not always successful. All the plans rarely get completed by the end of the project. Lack of developers' skills in a specific field and hardware malfunctions are some of the reasons for this failure. So, during the developing phase, the developers must always have a mindset of dropping some ideas in product planning.

Initially, this project plan included a camera to monitor the pet. During the development process, the camera did not work. After much debugging, the problem remained unsolved. So, the idea of having a camera was dropped at some point in the developmental phase.

PWAwise, the aim projected in planning was achieved. Of course, there is still much room to improve. But for this specific project, the goal of creating a certain PWA was achieved.

## 4 CONCLUSION

This chapter concludes the thesis by summarizing this project's results and findings. It will also go through the limitation of the project and provide recommendations for future projects.

The main goal of this project was to create a smart petfeeder using raspberry pi. The raspberry is connected to the internet and is controlled by PWA. The project for the thesis was mainly divided into two parts: hardware assembly and the development of PWA.

Assembly of hardware was simple, and components were readily available on the market. The product consists the parts like a dc motor and motor driver. The motor was connected to a rotating shaft on the bottom of the container. The motor is connected to raspberry pi's GPIO pin through the motor driver. Raspberry Pi also has a camera as a peripheral.

The development of PWA began by creating a simple responsive website. Additional features such as manifest and service workers were added to the given website to convert into a simple PWA. The PWA was hosted in the raspberry pi using the flask server and was responsible for controlling the GPIO pin of the raspberry pi. During the research, It was clear that any developer with knowledge of HTML, CSS, and Javascript could create a PWA and quickly read through the code. The study also concluded that any website could be converted into PWA and give the user a closer native-like app experience if it meets the requirements. As this project was more focused on controlling the hardware, It did not explore all the potential of PWA. Functions such as push notifications are out of the scope of this project.

One can pursue further development of this project as it holds much potential. The project has not gone in-depth with the development of PWA, which can be a topic to follow in the future. The addition of the hardware can improve the product itself. All in all, the project was partly successful. Although the project could not fulfill all the functions mentioned in the planning phase, the outcome was satisfactory.

## REFERENCES

Sivewright, J., Americas, P. and Krueger, N. (n.d.). Winning in PetCare. [online] Available at: [https://www.nestle.com/sites/default/files/asset-library/documents/library/presentations/investors\\_events/investor-seminar-2019/petcare.pdf](https://www.nestle.com/sites/default/files/asset-library/documents/library/presentations/investors_events/investor-seminar-2019/petcare.pdf)[Accessed 5 Jan. 2022].

2018 — Association for Pet Obesity Prevention (2018). Association for Pet Obesity Prevention. [online] Association for Pet Obesity Prevention. Available at: <https://petobesityprevention.org/2018>. [Accessed 5 Jan. 2022].

Google Developers. (2016). Service Workers: an Introduction | Web Fundamentals. [online] Available at: <https://developers.google.com/web/fundamentals/primers/service-workers>. [Accessed 12 Feb. 2022].

Dean Alan Hume and Addy Osmani (2018). Progressive web apps. Shelter Island, NY: Manning Publications. [Accessed 18 Feb. 2022].

www.w3schools.com. (n.d.). JavaScript HTML DOM. [online] Available at: [https://www.w3schools.com/js/js\\_htmlDOM.asp#:~:text=The%20DOM%20is%20a%20W3C](https://www.w3schools.com/js/js_htmlDOM.asp#:~:text=The%20DOM%20is%20a%20W3C) [Accessed 24 Mar. 2022].

Techopedia.com. (n.d.). What is an Embedded System? - Definition from Techopedia. [online] Available at: <https://www.techopedia.com/definition/3636/embedded-system#:~:text=An%20embedded%20system%20is%20a> [Accessed 28 Mar. 2022].

Gartner. (n.d.). Definition of Internet Of Things (IoT) - Gartner Information Technology Glossary. [online] Available at: [https://www.gartner.com/en/information-technology/glossary/internet-of-things#:~:text=The%20Internet%20of%20Things%20\(IoT\)](https://www.gartner.com/en/information-technology/glossary/internet-of-things#:~:text=The%20Internet%20of%20Things%20(IoT)). [Accessed 5 Apr. 2022].

pymbook.readthedocs.io. (n.d.). Introduction to Flask — Python for you and me 0.4.alpha1 documentation. [online] Available at:

<https://pymbook.readthedocs.io/en/latest/flask.html>. [Accessed 18 Apr. 2022].

Yoav Farbey (2016). Product Life Cycle & Product Development Cycle. [online] Medium. Available at: <https://defineproducts.com/product-life-cycle-product-development-cycle-bccb9c5aabf2>[Accessed 28 Jul. 2022].

Raspberry Pi (2013). What is a Raspberry Pi? [online] Raspberry Pi. Available at: <https://www.raspberrypi.org/help/what-%20is-a-raspberry-pi/>. [Accessed 28 Jul. 2022]