**EECE 281 Section 202**

**Project 2: The Automatic Pet Feeder**

Lab section: L2C Group #: 12 Group’s Lab-Bench #s: 8C and 1C

Ryan Birtch 25563140 1/6

Angela Cho 37107133 1/6

Timothy Choy 18998120 1/6

John Deppe 49910110 1/6

Kimia Nikazm 33195124 1/6

Justin Wong 35922137 1/6

**TABLE OF CONTENTS**

**1.0 CONTRIBUTIONS**  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3

1.1 Ryan Birtch . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3

1.2 Angela Cho . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3

1.3 Timothy Choy . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3

1.4 John Deppe . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3

1.5 Kimia Nikazm . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3

1.6 Justin Wong . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3

**2.0 INTRODUCTION AND MOTIVATIONS** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4

2.1 Files Submitted . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4

**3.0 PROJECT DESCRIPTION** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4

3.1 Project Components . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5

3.2 Setting Up the CC3000 Wifi Shield . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5

3.3 Setting Up the Food Dispenser . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5

3.4 Choice of Processor . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6

3.5 Setting Up the Spark Core . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 7

3.6 The Casing . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 7

3.7 IFTTT . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 8

3.8 Android Application . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 10

3.9 THE Website (pawzles.com) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 10

**4.0 CONCLUSIONS** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 12

**5.0 REFERENCES AND BIBLIOGRAPHY** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 12

**Appendix A: Spark Core Code** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 13

**Appendix B: Android App/Website** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 17

1. **CONTRIBUTIONS**

**1.1 Ryan Birtch**

Ryan helped with the original coding and troubleshooting of the Spark Core. He also contributed in the procurement of the cereal dispenser and various other components. Ryan spent a significant portion of his time trying to find appropriate APIs, interfaces, and services that would be best for product implementation. Ryan also helped write the lab report for this project.

**1.2 Angela Cho**

Angela has contributed to the project with the initial idea and concept. She has also worked in setting up the Adafruit CC3000, as well as moderate coding for the Spark Core and calibrating the force sensor. Using the Spark Core, she has also worked on setting up IFTTT "recipes" for use across multiple platforms (ie. Twitter, Sms, Android OS) to control and receive alerts from the dispensing device. Angela has also worked on the design and construction of the casing/housing of the project and the project report.

**1.3 Timothy Choy**

Tim incepted the idea for the Dog Feeder alongside Angela. He contributed to both hardware and software for this project. He aided in the wiring of components as well as much of the coding and IFTTT recipes to get things running. He also helped in the design of the dispenser as well as troubleshooting for the force sensor to check if there was food in the bowl. Tim worked on code for both the force sensor and servo.

**1.4 John Deppe**

John identified the Spark Core platform as suitable for our application. He also sourced the Spark Core, the cereal dispenser, and a suitable servo to turn the handle in order to dispense our food. He mounted servo on cereal dispenser as well as writing an android app to interact with the Spark API. John also helped write parts of the report.

**1.5 Kimia Nikazm**

Kimia contributed to the initial coding and setting up the CC3000 wifi shield and the spark. She also contributed in troubleshooting the initial connection problems with the Spark Core. Additionally, she contributed in the testing of the final project and painting the dog feeder stand. She also finished the fritzing schematic and had a major contribution in writing the report.

**1.6 Justin Wong**

Justin has contributed to the project with the creation of the website and with debugging the Spark Core and android app. He primarily created and populated the website with content, set up the website server and backend, customized the php script for sending the HTTP requests from the website and he assisted with the prototype of the PetFeeder build and android app. Furthermore, he helped debug issues with IFTTT interfacing, the Spark Core code and also setup the ability to interface with the Spark Core microcontroller via HTTP requests. Justin also completed the flowchart for the website creation process.

1. **INTRODUCTIONS AND MOTIVATIONS**

The report encompasses the second project for EECE 281 section 202, Winter 2014. Each group chose project 2 uniquely. However, the main objective of this project was to implement a so-called “Internet of Things” (IoT) project. The IoT describes a potential network of physical objects, all of which have the ability to transfer data over a network without requiring human-to-human or a human-to-computer interaction. Our group decided to implement a smart dog feeder that is controlled over wifi through several different platforms. This project was born out of the need to feed their dog remotely due to forgetfulness of owners or simply those on vacation or at the office late at night. This system would allow the dog's owner to refill the dog's bowl in their absence by either sending a text, pressing a button on our app, or accessing the website. This dog feeder is comprised of a base, a dispenser, and a feeding bowl. The system is also capable of sending iOS or Android notifications to the owner to inform them when the bowl is empty or full upon request or on a timed interval.

**2.1 Files Submitted**

The list and descriptions of the files submitted for this project include the following:

• Project2.fzz: This is the fritzing breadboard schematic which shows the layout of the components used for this project. This schematic includes the Spark Core microcontroller which the force sensor (FSR 406) and the Servo are attached to the core. Additionally, the servo is mounted on the dispenser to allow it to open without direct human intervention.

• Project2\_Report.docx: This is the final report on the Dog Feeder project for EECE 281. It includes the contribution percentage and description of each team member, as well as an overview of the overall design approach and the implementation procedure of this project.

• dogfeederwithscheduling.ino: This is the final Spark core sketch, containing the complete code for the dog feeder system. The code monitor the force being applied to the sensor via the weight of the bowl, and depending on the weight of the bowl, it sends a text message to the owner's phone. Additionally, it controls the servo to dispense the food whenever it receives a certain text message.

• Dog Feeder Android application: See Appendix B.

• Pawzles.com: See Appendix B.

**3.0 PROJECT DESCRIPTION**

**3.1 Project Components**

* Adafruit CC3000 WiFi Shield
* uFL to rp-sma adapter
* WiFi antenna
* Servo Square Force Sensitive Resistor 406
* Food Dispenser
* Spark Core
* 10 ohm resistor
* AC adapter

**3.2 Setting Up the Adafruit CC3000 WiFi Shield**

Description:

The Adafruit CC3000 is a WiFi shield made for use on an specific Arduino models such as the Arduino Uno. The shield incorporates the TI CC3000 chip, a wireless network processor. The model we bought requires an antenna for use.

Design:

The shield was packaged with pins that were to be soldered on.

**3.3 Setting up the Food Dispenser (Servo & Force Sensor)**

For the project’s mechanical requirements, we briefly considered constructing a food dispensing mechanism, before deciding that was beyond our reasonable capabilities. We searched for preexisting home-built dog-feeder systems and found https://www.youtube.com/watch?v=YejpfCDh4Lc. We were unable to source a similar silicone-wheel Zevro cereal dispenser in Canada, and settled for an OXO door-type cereal dispenser from Homesense to build around.

Next, we had to find a way to drive the door on our food dispenser. We decided to use a servo, since they make positioning easy. We elected to drive the servo against the dispenser’s door closing spring, reasoning that this would let the door close faster, and enable us to create power-saving functionality by deactivating the servo when not actively opening the door. This left us with a challenge of designing a system capable of providing enough torque to turn the door against the spring.

Being computer engineering students, we elected to not design a gearset, and to instead directly drive the door-opening mechanism with our servo. Our remaining challenge was to source a servo with sufficient torque to turn the spring. We tested the spring’s torque by attaching an arm and pressing the arm against a scale, finding an approximate torque of 11 kg-cm was required. Accordingly, we sourced the PowerHD 1501 MG servo, rated for 17 kg-cm of torque, which proved able to do the job. We mounted the servo by trimming one of the servo horns to fit the door mechanism’s keyway and then bolting the servo body to the cereal dispenser.

Another challenge was sourcing a suitable power supply for the system. The Spark Core draws approximately 0.5 amps at 5V. The servo has a stall current of 2.5A at 6V. Lucky, RP Electronics had a reasonably priced 6V, 3A power supply for us to use.

Overall, this step of the project was to integrate the servo with pre-assembled cereal dispenser. And implement a simple code to power the servo depending on the force being read from the force sensor connected to the Arduino. We used Arduino for testing the force sensor. We first needed to find a servo with enough torque to overcome the tension of the spring, which holds the door of the dispenser closed. We needed less than 17 kg per centimeter of torque to overcome the tension. The servo is a Analog Servo- HD-1501MG by HUIDA. It was chosen because it provided sufficient torque and it's small angle of action was perfect for our needs.

Design

We had two ideas about how to open the cereal dispenser door. The first was to mount the servo behind the door and wind a string to pull the door open. Another choice was to mount the servo directly on the side of the dispenser where the lever, which would usually be operated by the user, would reside. The latter was chosen because we predicted that the string could be cut by or caught on the pet that would be using the device. Mounting it directly in the side also came with its own problems – mainly, the plastic key that we had to sculpt so that it fit into the notches used by the lever.

Also, mounting the motor on to the side was another problem we encountered. We needed to make sure the mounting was sturdy enough so that the torque provided by the motor would not break it off from the side of the dispenser, nor damage the mechanical components within. The first key that we made which fit into the notches was made of a softer plastic than was suitable for our implementation and therefore broke after a day of testing. We then constructed a second key made from a far stronger plastic that suited our needs perfectly.

We decided to bolt the motor onto the side using the servo's built-in frame for mounting. After setting up the servo, we connected the Arduino to the servo for troubleshooting and it would dispense the food based on the force threshold we found during the testing.

Testing and Evaluation

There was some trial and error involving the configuration of the servo (e.g. how much power we needed, how much we wanted to turn). We also had to troubleshoot the threshold for when we wanted alerts from our program and also estimate the initial weight on the force sensor.

**3.4 Choice of Microcontroller Platform**

Description

One of the requirements was for the device to be connected to the Internet. Our project needed to interface with many different APIs and libraries for correct and robust implementation. The use of an Arduino and the CC3000 WiFi shield limited us in certain aspects with regards to our functionality. This problem was remedied by using another and more capable processor. The Spark Core that we chose has a WiFi chip on board and extensive documentation.

Design

We first tried to use a CC3000 WiFi Shield mounted on the Arduino. This component came disassembled, and after soldering we started on implementation. The shield that was purchased needed a specific library in order to work. The library integrated poorly with the majority of libraries needed for our project. In order to work with the non-standard WiFi library, we would need to spend a significant portion of our time simply enabling basic functionality. After much discussion within the group we decided to use another the Spark Core processor. This processor has a built-in WiFi chip with an extensive library, which interfaces easily with many different APIs and services.

**3.5 Setting Up the Spark Core**

Description

The Spark Core is an Arduino-compatible, Wi-Fi enabled, cloud-powered development platform. The first step was creating a Spark account in order to use the online IDE. This cloud based platform allowed simultaneous access to the project code and no cable was needed to flash the program to the core.

Design

This is not applicable as the Spark Core processor came assembled.

Testing and Evaluation

One of the difficulties that our group faced during setting up the Core was finding the device ID to connect the core to the Spark cloud. We initially attempted to connect to a mobile hotspot since a WPA2 Enterprise network, such as UBC-Secure, was an entirely invalid option for WiFi setup. We then also realized that to connect to a hotspot we required the TI CC3000 driver in order for the network to be able to interface with the device. By downloading PuTTy, a terminal emulator, and the CC3000 driver, we connected the Core to the given SSID and were able to generate the device ID. Further testing included challenges for uploading new code to the Spark Core. We found that this was due to the processor being busy at time of upload (i.e. stuck in a loop or process with a delay). To fix this, we simply implemented our code to delay using the millis() function instead of delays.

**3.6 The Casing**

Description

The original cereal dispenser had a very short and shallow cavity under the door where the user would have their bowl filled. This opening was not sufficient for a pet to both have their head and bowl underneath. We decided to split up into two teams and develop options for the design prototype.

Design

Our two design prototypes had distinct philosophies between them. One design group used a platform and a ramp to dispense the pet food into a bowl that was separate. The other design featured a ramp and a built-in tray for the food to dispense into. The first prototype was wide and designed to prevent the pet from knocking over the dispenser. The second prototype that was developed to fully integrate the ramp and bowl together for both visual appeal and simplicity of use. Deciding on which prototype was the best was a difficult process because both prototypes functioned equally well. We decided on the second prototype due to its aesthetic benefits, and further honed the design.



Figure 1. Our final Dog Feeder Design

**3.7 IFTTT (If This, Then That)**

Description

This step was to make our project Internet connected and have the dispenser able to interact with many different platforms regardless of locations. The use of the IFTTT platform opens up a wealth of opportunities for wireless interaction with our dispenser. In this step we were able to make our device be activated by SMS, Twitter, Website Intervention (discussed in a later step), and an Android Application (also discussed in a later step). This was all done by HTTP request to the IFTTT server, which was synced up to our Spark Core (through our account credentials) and various other platforms (i.e. the user’s phone number).

Design

The platform IFTTT is a scripting platform that has numerous APIs and capabilities built in for a user. We used the Spark Core's seamless integration with the web to activate methods and send messages to the user. IFTTT mainly responds to prompts, whether it is a text message, a tweet, or an email. The first step towards using this platform was creating the account; we were then able to add our own telephone numbers and the Spark Core as "ingredients" in a "recipe". These "recipes" could interact with specific functions programmed onto our Spark Core, check the return values of these functions and even monitor values of variables that we have made visible to outside programs. The use of predetermined hashtags in text messages and tweets can activate these functions and set in motion further "recipes".

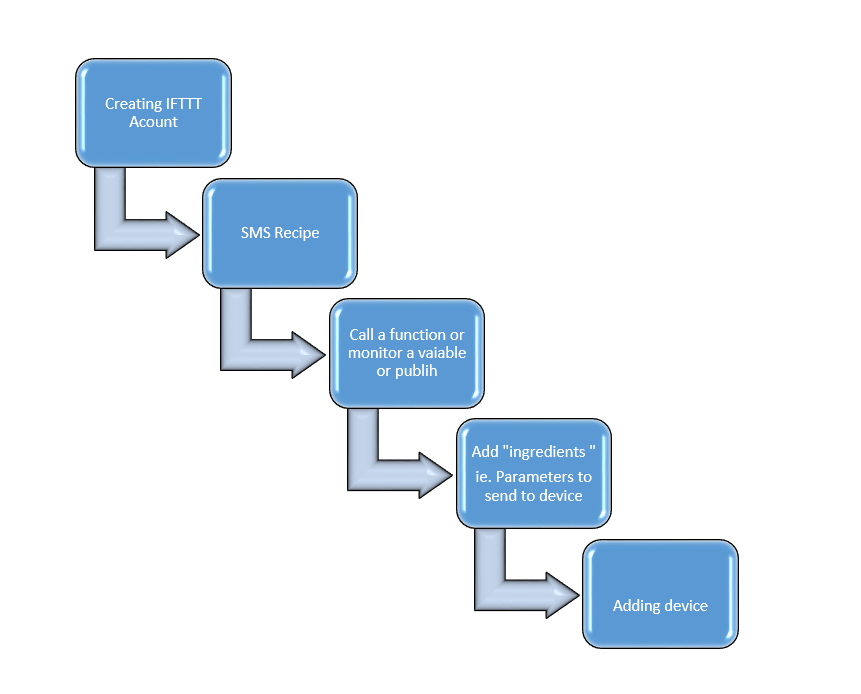


Figure 2. IFTTT Set Up

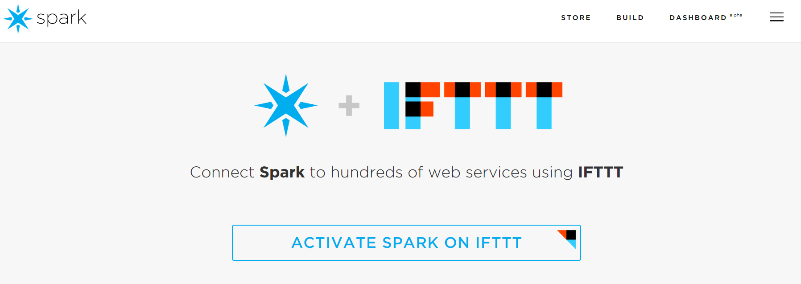


Figure 3. IFTTT Website

Evaluation/Testing

During this step, we faced various difficulties, one of which was to be able to monitor the progress of the specific "recipes".

**3.8 Android Application**

Description:

Design:

Evaluation/Testing:

**3.9 The Website (www.pawzles.com)**

Description

This step was to build a website in order to monitor the device remotely. The website will allow the user to fill the pet's bowl whenever it's needed as well as having the system on the " Scheduler " mode which will automatically fill the bowl after the certain amount of time has passed. Additionally, the website has a live feed feature which will allow the user to monitor their pet while they are not at home.

Design

The website was well designed with a simplistic and minimalist use of color and a text which was easily read with various graphic. The organized layout allows users ease of use of the website. The website includes a welcome page, our purpose, who we are, and a live video feed. Figure 5 shows a first page view. The steps taken to our website is described in the flow chart in figure 6.

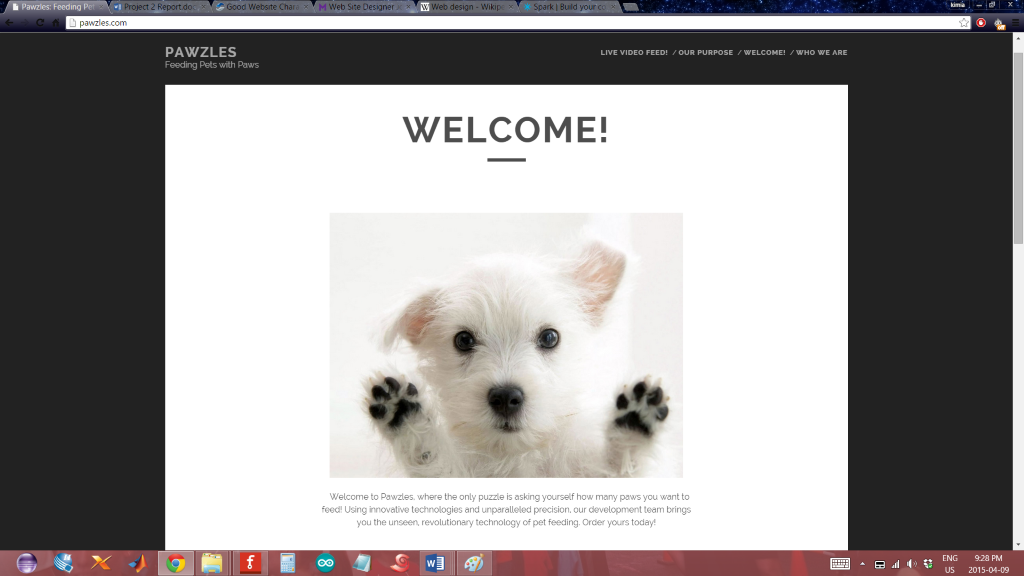


Figure 5. Homepage of www.pawzles.com

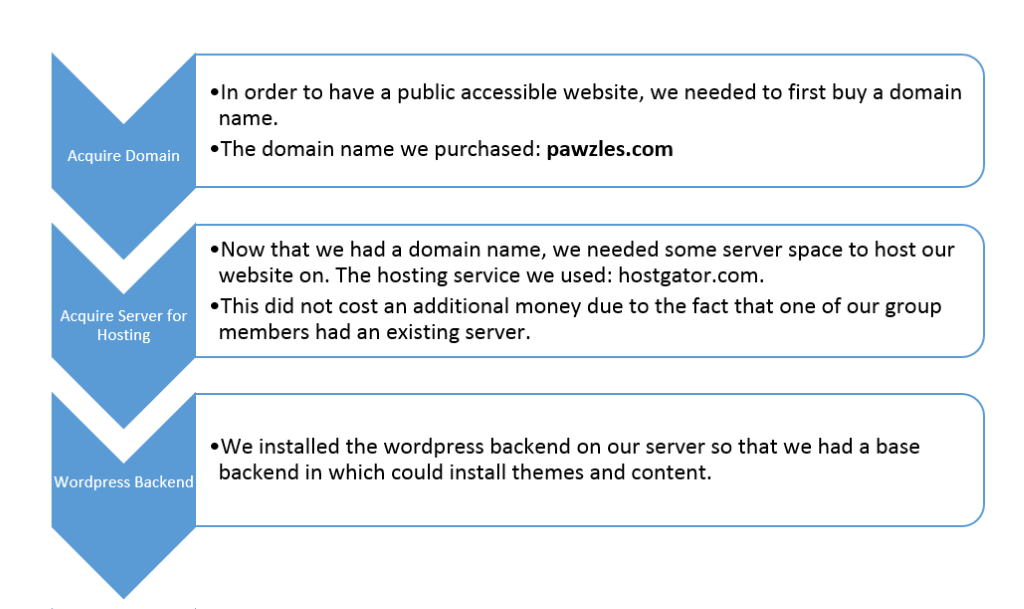
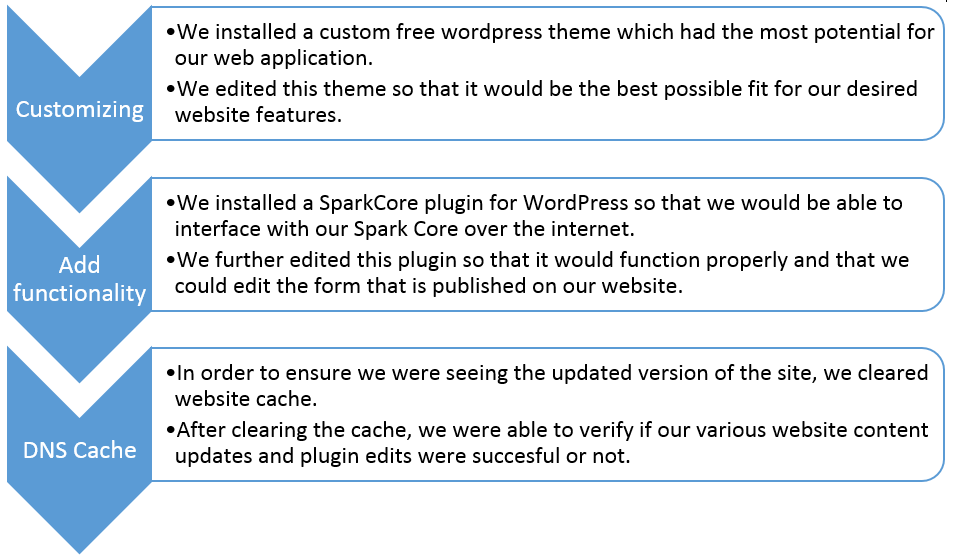


Figure 6. Website Implementation Flowchart

Evaluation/Testing

In order to test and review each step completed in this part, we had to publish the website after each updates and access it from an external device. One of the difficulties in this step was the DNS propagation delay. Typically, the delay is about 24 to 48 hours, however, this delay was bypassed by manually editing the MX records.

**4.0 CONCLUSION**

This project not only improved our technical skills, but also improved our communication and team-based skills. This project was a learning experience in not only implementing the use of new hardware but also in researching the products we wish to purchase in more depth before they are bought. We gained an adequate understanding of working with WiFi and implementing API's, new and innovative technologies that are available as well as time management with a large group (there was a new addition to our project group). Many of the issues we encountered were related to setup of hardware and methods in which to use the hardware. While using a Spark device may not be ideal in the eyes of others when we already possessed an Arduino device, our project group agrees that the Spark Core was incredibly useful and less time consuming.

**5.0 REFERENCES AND BIBLIOGRAPHY**

<https://www.spark.io/> : Spark Core website

<https://ifttt.com/recipes> : Recipe Creation (Spark Core)

<http://www.steadlands.com/data/interlink/fsr406.pdf> : Force Sensor datasheet

<http://stackoverflow.com/questions/28291910/how-to-spark-core-time-based-event> : Scheduling events on a Spark Core

<http://docs.spark.io/> : documentation on anything Spark related

<http://arduino.cc/en/Tutorial/BlinkWithoutDelay> : Tutorial on using millis() function

<https://developer.android.com/training/basics/firstapp/index.html> : Building your first Android app

**Appendix A: Spark Core Code**

#define WEIGHT\_MARGIN 2100

Servo myservo; // create servo object to control a servo

int pos = 0; // variable to store the servo position

int fsrAnalogPin = A1; // FSR is connected to analog 0

int fsrReading = 0; // the analog reading from the FSR resistor divider

long currentMillis; // variable to store current milliseconds elapsed for delay

long previousMillis = 0; // variable to store the previous milliseconds to compare for delay

int interval = 120000; // interval passed between current and previous milliseconds before running a function

int automode,ab,al,ad=0; // variables to run auto modes when true

int dinner=0; // run dinner if true

int lunch =0; // run lunch if true

int breakfast =0; // run breakfast if true

int forever, fb, fl, fd = 0; // variables to run forever modes when true

int stop = 0; // variable to send forever mode

int minute = 0; // minutes between dispenses in auto mode

void setup()

{

myservo.attach(A0); // attaches the servo on pin 9 to the servo object

pinMode(fsrAnalogPin, INPUT); // instantiate as input to read from FSR

Spark.variable("fsrReading", &fsrReading, INT); // initialize spark variable to be accessed by APIs

Spark.function("PetFeeder", dispenseApp); // initialize spark function to be accessed by APIs

Spark.subscribe("feed", dispenseText, MY\_DEVICES); // subscribe to keyword "feed" to run dispenseText if found

Spark.subscribe("check", checkBowl, MY\_DEVICES); // subscribe to keyword "check" to run checkBowl if found

Spark.publish("send-message", "Hi! Welcome to your new Pet Feeder!"); // send message

}

void loop() {

if ((automode) && (((Time.minute() == minute ) || (Time.minute() == (minute + 1)) || (Time.minute() == (minute + 2))))) // if automode for the next 3 minutes is activated

{

// minutes setup

if (Time.minute() == (minute+1) && ab){

dispenseApp("food");

ab = 0;

}

else if (Time.minute() == (minute+2) && al){

dispenseApp("food");

al = 0;

}

else if (Time.hour() == (minute+3) && ad){

dispenseApp("food");

ad = 0;

}

}

if (stop){ // if stop == 1 stop the forever feeding on schedule

forever = 0;

stop = 0;

fb = 0;

fl = 0;

fd = 0;

}

if (Time.hour() == 23 && !(fb) && !(fl) && !(fd) && forever){ // on the 23rd hour of everyday, the forever variables are reset

fb = 1;

fl = 1;

fd = 1;

}

if (ab == 0 && al == 0 && ad ==0){ // if the feeding for the day is done, set automode to zero

automode =0;

}

if ((forever) && (((Time.hour() == 8) || (Time.hour() == 12) || (Time.hour() == 18)))) // forever mode

{

if (Time.hour() == 8 && fb){

dispenseApp("food");

fb = 0;

}

else if (Time.hour() == 12 && fl){

dispenseApp("food");

fl = 0;

}

else if (Time.hour() == 18 && fd){

dispenseApp("food");

fd = 0;

}

}

else if (dinner && Time.hour() == 18){ // dinner mode

dispenseApp("food");

dinner = 0;

}

else if (breakfast && Time.hour() == 6){ //breakfast mode

dispenseApp("food");

breakfast = 0;

}

else if (lunch && Time.hour() == 12){ //lunch mode

dispenseApp("food");

lunch = 0;;

}

fsrReading = analogRead(fsrAnalogPin); // read FSR pin

currentMillis = millis(); // read how many milliseconds elapsed

if(currentMillis - previousMillis > interval) { // if interval has elapsed, check bowl

previousMillis = currentMillis;

if(fsrReading < WEIGHT\_MARGIN){

Spark.publish("send-message", "No food in bowl! Text #feedmypet to dispense.");

}

else if(fsrReading >= WEIGHT\_MARGIN){

Spark.publish("send-message", "Bowl is filled! Dog has not eaten yet.");

}

}

}

// this function checks the bowl upon user's request

void checkBowl(const char \*event, const char \*data) {

fsrReading = analogRead(fsrAnalogPin);

if(fsrReading < 2100){

Spark.publish("send-message", "No food in bowl! Text #feedmypet to dispense.");

delay(1000);

// Spark.publish("send-message",tmpstr);

}

else if(fsrReading >= 2100){

Spark.publish("send-message", "Bowl is filled! Dog has not eaten yet.");

delay(1000);

// Spark.publish("send-message",tmpstr);

}

}

// dispenses food upon a user's request via text message (through IFTTT)

void dispenseText(const char \*event, const char \*data) {

pos = 0;

myservo.write(pos); // tell servo to go to position in variable 'pos'

delay(1500); // waits 1s for the servo to reach the position

pos = 85;

myservo.write(pos);

delay(1000);

Spark.publish("send-message", "The bowl has been filled.");

}

// dispenses food upon a user's request through an Android app.

int dispenseApp(String command) {

if(command == "food"){

pos = 0;

myservo.write(pos); // tell servo to go to position in variable 'pos'

delay(1000); // waits 1s for the servo to reach the position

pos = 85;

myservo.write(pos);

delay(1000);

Spark.publish("send-message", "The bowl has been filled.");

return 42;

}

// code for auto scheduling via Android app

else if (command == "automode"){

automode = 1;

minute = Time.minute();

ab = 1;

al = 1;

ad = 1;

Spark.publish("send-message", "Automode Activated");

return 1;

}

else if (command =="dinner"){

dinner = 1;

Spark.publish("send-message", "Dinner mode Activated");

return 2;

}

else if (command == "breakfast"){

breakfast = 1;

Spark.publish("send-message", "Breakfast mode Activated");

return 3;

}

else if (command == "lunch"){

lunch = 1;

Spark.publish("send-message", "Lunch mode Activated");

return 4;

}

else if (command == "forever"){

forever = 1;

fb = 1;

fl = 1;

fd = 1;

Spark.publish("send-message", "Forever mode Activated");

return 5;

}

else if (command == "stop"){

stop = 1;

Spark.publish("send-message", "Stop Triggered");

return 6;

}

// there is an error so return -1

return -1;

}

**Appendix B: Android App/Website**

* Please find the code for our Android app at <https://github.com/johndeppe/Feedr>.
* Also find our website at [www.pawzles.com](http://www.pawzles.com). Note that the live feed camera will not display if our IP camera and system is not running, however the website will still be live. Please right click and select “inspect element” for more.