LONG BEACH STATE UNIVERSITY
College of Engineering

Final Report

California State University Long Beach

College of Computer Engineering and Science

CECS-490B : Senior Design Project II

Abigail Kwan, Jeremy Escamilla, Yamin Yee

5th May 2020

Table of Contents

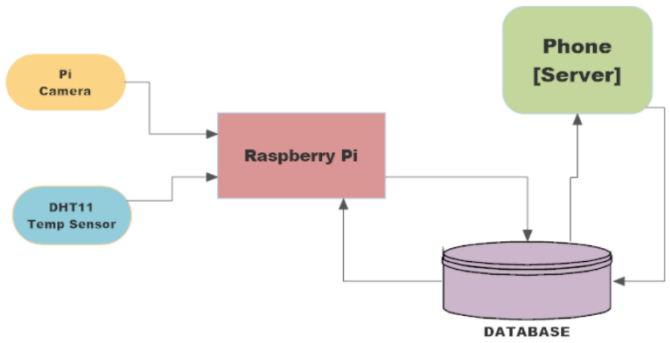
1. Introduction

1.1 Executive Summary

This report will act as an analysis of the Food Spy project and workings of the Food Spy team. The components of the Food Spy Project will be accessible via the Table of Contents.

The following portions have been updated/created:

- Challenges: Ethical, Technological, Recent Events: COVID-19
- Timeline: Demonstration List, Progress
- The App: Server Interaction, Food Profile Listings



Team Blue Jay produced a convenient food tracking system to allow for awareness of the items in one's fridge while the user is away from home. This portion of the project will be broken down into three pieces: The Fridge, The App, and the Sensors. These portions will work in conjunction to measure and alert the user regarding the fridge's temperature, allow for the user to build profiles for items stored in the fridge, and allow for the user to visually assess the fridge.

1.2 Meet Blue Jay!

1.2.1 - Abigail Kwan

I'm Abby and I'm the Core App and Pi-App Communications Developer. What interests me about Computer Engineering is the problem solving and the ways code can interact with hardware, especially with regards to IoT. What brought me to Food Spy was the creation of the App and Server, which was a challenge that would prove both fun and educational to tackle. Learning new languages, building Databases, and understanding communications between the Backend of the App and the Frontend proved satisfying to tackle in this project.

1.2.2 - Yamin Yee

I am Yamin and I'm the primary Fridge Technician/Modifier and Raspberry Pi Specialist. I had entered the Computer Engineering field due to my belief that computers can have a positive impact on people's lives and be convenient! For Food Spy, I really liked the idea of the fridge and allowed for people to visually monitor their food. Additionally, the safety aspect of the project where people can be safer knowing if their food has spoiled due to the fridge becoming too hot attracted me to the project. It was fun working with both the Sensor aspect of Food Spy and the App aspect of the project, allowing me to get an understanding for both aspects of the project and how each piece is integral to the Food Spy system.

1.2.3 - Jeremy Escamilla

I'm Jeremy, and I'm the Temperature Measurement Specialist and Schematic/Flowchart Designer for Food Spy. What brought me into the field of Computer Engineering was my interest into the process of computer creation, the diversity in different types of computers, and how computers can help make safety and comfort all the easier to have. For Food Spy, I became interested in the project as a result of its potential to assist in the awareness of fridge conditions for users away from home, and usage in the healthcare industry to allow for patients to have both safety and comfort when storing their food. It was fun to act as a both a fellow debugger/developer of the Pi to Yamin and overall consultant for how the fridge is monitored via Temperature for both the App and Raspberry Pi.

1.3 Team Roles

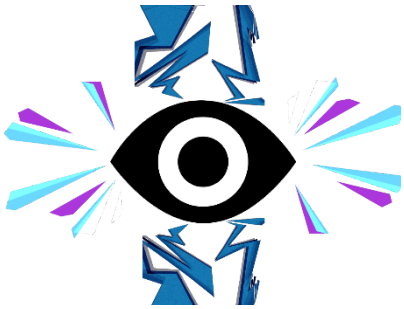
1.3.1 - Abigail Kwan

Abigail's role was over the App. This includes development App Frontend and Backend Developer for the Food Spy project. This involves the development of the web server that will store user data regarding the food in the fridge, the transfer of images from the Camera, and the storage (and processing) of Fridge Temperature.

1.3.2 - Yamin Yee

Yamin's role in the group was over the Fridge. This includes the implementation of the final code for the Pi Camera, DHT11 sensors, and for interacting with the App backend with respect to the Raspberry Pi. Additionally, the fridge requires modification to facilitate optimal camera, sensor, and Pi placement.

1.3.3 - Jeremy Escamilla



Jeremy's role was over the Sensors. He was responsible for researching, programming (or assisting with, given recent events), and debugging the DHT-11 and Raspberry Pi Camera. Additionally, he was responsible for assisting members with regards to temperature sensing logic with both the Fridge portion of the project, and the App portion of the project.

1.4 Team Contribution

1.4.1 – Yamin Yee

- Drilled and Wired the Fridge to Raspberry Pi, Camera, and DHT-11 Sensor
    - Modified firmware of Pi (memory) to facilitate Camera usage
- Programmed DHT-11 for Raspberry Pi (w/ Jeremy Escamilla)
    - Incorporated connection to web server (w/ Abigail)
- Programmed the Raspberry Pi Camera
    - Incorporated connection to web server (w/ Abigail)
- Programmed the CRON scheduler into the Pi

1.4.2 – Abigail Kwan

- Programmed App GUI via Flutter
- Programmed Front End behavior – Food Profile
    - Creation, Deletion, Editing
- Programmed Front End (App) behavior – Temperature Display and Alert (w/ Jeremy)
- Programmed API between Front End and Back End
    - Utilized 000webhost.com to create server
- Programmed Back End (Server) behavior – SQL Query interaction between API and database

1.4.3 – Jeremy Escamilla

- Programmed and tested DHT-11 (via ESP-32)
    - Temperature alert and Frequency Check, latter was dropped upon switch to Pi.
- worked with Yamin to:

- determine angle/placement of components in Fridge
- Debug Raspberry Pi Camera
- Research best choices for fridge and compare with FDA requirements
- DHT-11 programming/wiring/debugging/troubleshooting
- Create hardware schematics and software flowcharts

- Worked with Abby on Temperature Alert logic

2. Food Spy

2.1 The Fridge

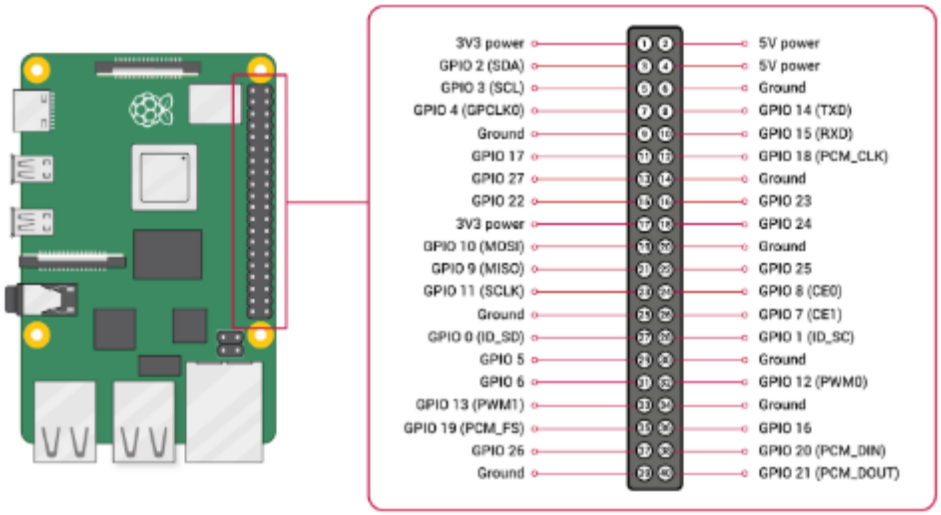
Instead of building a fridge from scratch, we decided to just modify an existing fridge. The fridge we chose is a mini thermoelectric system cooler. It comes with two power adapters: one for a standard wall outlet and the other is for 12V cigarette outlets in vehicles. This mini fridge is very portable and can cool up to 0 Celsius (freezing temperature). Yamin drilled the fridge so that the wiring for the camera and the sensors can connect to the Raspberry Pi that is attached to the outside of the fridge.

2.1.1 Raspberry Pi 3 Model B+

The Raspberry Pi 3 Model B+ acts as the middleman (or "Broker") for sensor data. By "Broker", it is meant that the Raspberry Pi receives the data from the sensor and camera and transfers the data to the App's backend for processing and decision making. The Raspberry Pi 3 Model B+ was chosen for its versatility, programmability, and small size.

The Raspberry Pi 3 Model B+ has 40 GPIO, 2 USB-A Ports, HDMIPort and 5V Power for the Pi via MicroUSB. With a Cortex-A53 64-bit SoC Processor, and 1GB SDRAM, including the option for an additional SD Card Memory, the Pi can subsequently utilize both the DHT11 and Raspberry Pi Camera Module simultaneously. Rasbpian is the primary OS, which utilizes Python in its terminal to allow for ease of programming and modification. The DHT11 and Camera Module Functions are accessible from the terminals themselves.

2.1.1.1 - Pin Layout of Raspberry Pi 3 Model B+



2.2 The Sensors

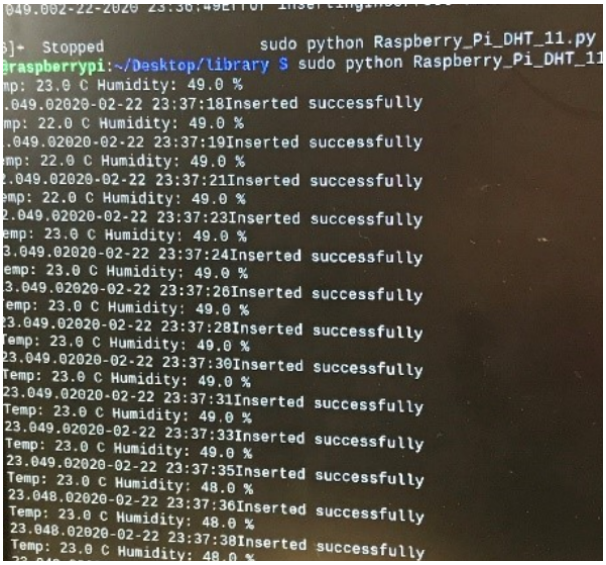
2.2.1 The DHT-11 Sensor

For our purposes, we utilized a specific variant of the DHT that comes as a module, complete with a pre-soldered element. In this case, this eliminates the need for a secondary resistor. In addition, said module can and is used to connect directly to the Pi, as it did with the ESP32. The primary difference between communication between communicating with the Pi and communicating with the ESP32 is that the ESP32 requires the Arduino IDE. while the Pi communicates with the device via the Python IDE.

2.2.2.1 - Programming

```
import sys
import Adafruit_DHT
import time
import requests

while True:

        humidity, temperature = Adafruit_DHT.read_retry(11,4)

        print 'TEMP: {0:0.1f} C Humidity: {1:0.1f} %'.format(temperature, humidity)

        time.sleep(1)


        url = 'https://abigailkwan.000webhostapp.com/index.php'
        humid = '{:0.1f}'.format(humidity)
        temp = '{:0.1f}'.format(temperature)

        params = {'temperature': temp, 'humidity': humid}

        x = requests.post(url,params)
        print(x.text)
```

The terminal output is not needed for the final product, but we used it for verification to show that we did test the sensors and that the Raspberry Pi was reading it properly. The data being read from the DHT11 is sent to the API using the “requests” library for Python. That library allows for data to be sent to the Front End of the App via SQL Query. The Results can be seen below:

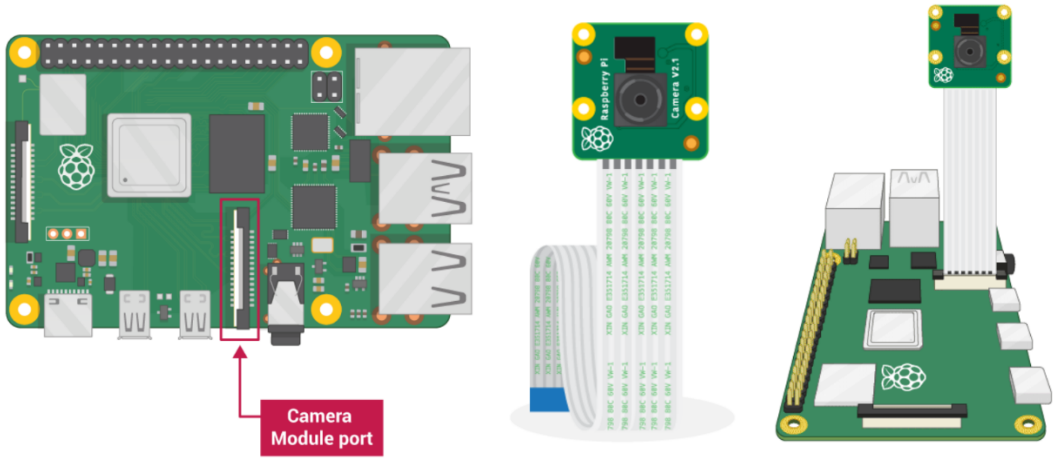


2.2.2 The Raspberry Pi Camera

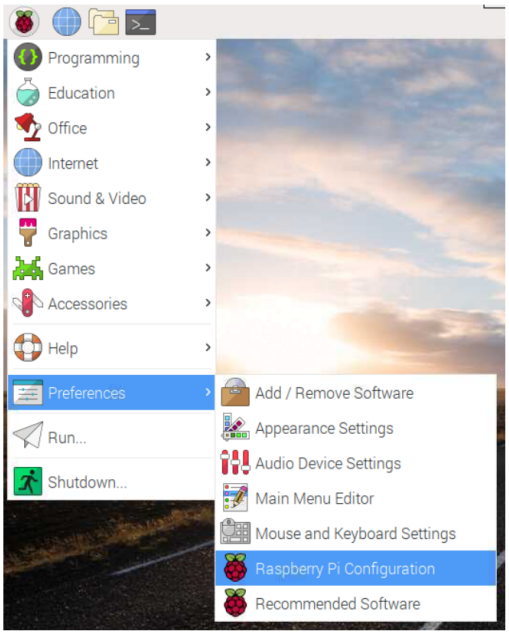
This camera module is what we use for the camera inside the fridge. Instead of using the official Pi Camera, which only has a specific field of vision, we chose to use this one for its fisheye lens so it can have a greater degree of vision. It also has LED lights attached to it so that it can see inside the fridge. This module can be connected directly to the Raspberry Pi without having to install any external drivers or additional parts.

2.2.2.1 - Installation Instructions

Step1: Set Up Pi Camera

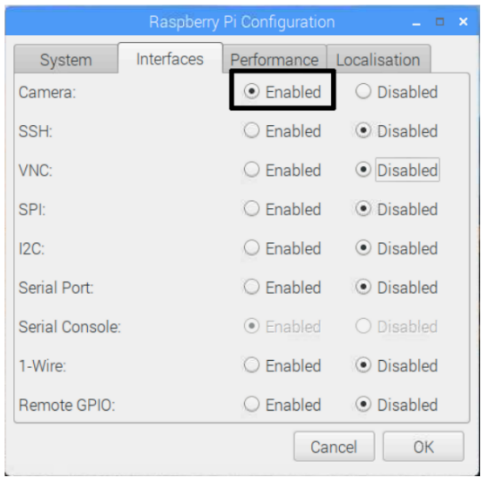


Above picture is how Yamen set up the Pi camera. After the setup, Yamen needed to install openCV on the Raspberry Pi.



Step 2: Raspberry Pi Configuration

Select the Interfaces tab and ensure that the camera is enabled:



Step 3: How to control the Camera Module via the command line

There are two command line tools raspistill and raspivid

- "raspistill" is the command for taking picture
- "raspivid" is the command for taking video

Use the terminal window to take a picture

The command line is as the following and then enter press to take picture

```
raspistill -o Desktop/image.jpg
```

The user can resize the picture by using the following command line.

```
raspistill -o Desktop/image-small.jpg -w 640 -h 480
```

Step 4: Controlling Pi camera with Python Code

The following code is used to control the pi camera

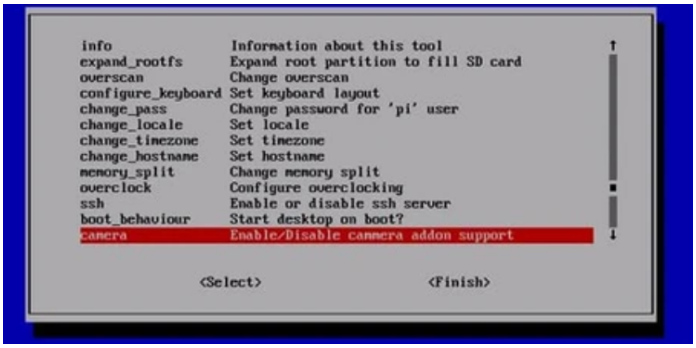
```
from picamera import PiCamera
from time import sleep

camera = PiCamera()

camera.start_preview()
sleep(5)
camera.stop_preview()
```

With this, the user can rotate the image by using 90, 180, or 270 degrees. To reset the image, set rotation to 0 degrees.

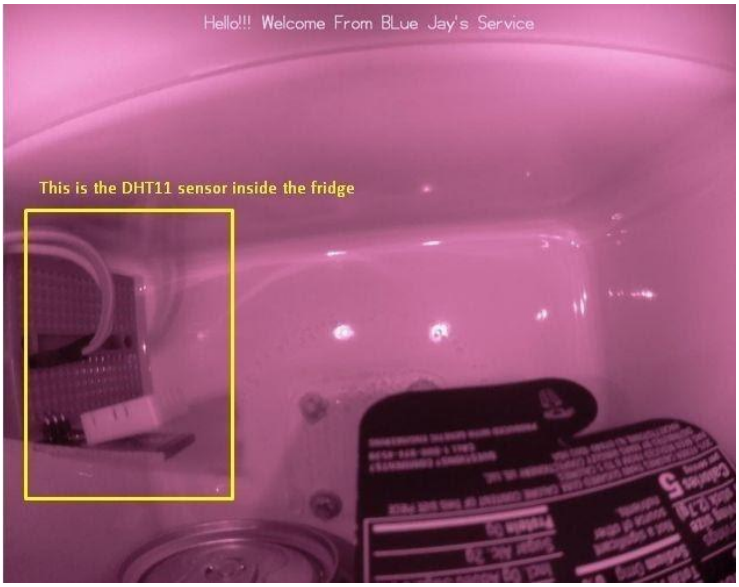
Also, the pi camera can be previewed by passing an alpha level parameter through start_preview.

```
camera.start_preview(alpha=200)
```

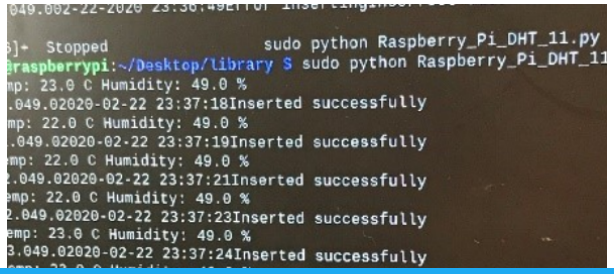
Now that everything is properly installed, all that is left is to reboot in order to make sure the Pi is running properly.



2.2.2.2 - Verification



Successful image capture of fridge interior



2.2.2.3 - Pi Camera Python Code

camera.py

C: > Users > Yamin Yee > Downloads > camera.py > ...

```python
import sys
import picamera
from time import sleep
import requests as req

cam = picamera.PiCamera()

while True:
#Set up the camera

#Start the Camera
    cam.start_preview()

#Display Text On Image
    cam.annotate_text="Hello!!! Welcome From BLue Jay's Service"

#FLip camera Horizontally TRUE/FALSE
#cam.hflip = True
#cam.vflip = True


    cam.capture('/home/pi/Desktop/camera/abby.jpg')

    sleep(5)
#for i in range(5):
    #Directory where you wish to capture the pictures
#   cam.capture('/home/pi/Desktop/camera/pic%s.jpg'%i)
```

```python

#Done after it is capturing the picture
#cam.stop_preview()

    url = 'https://abigailkwan.000webhostapp.com/upload_img.php'

    with open('abby.jpg','rb') as f:
        files = {'file' : f}

        r = req.post(url, files = files)
        print(r.text)

    cam.stop_preview()

    #sleep(30)
```

2.2.2.4 - Coding Details

The pi camera is set up first by importing the pi camera library. After importing the library, we set up an instance of the camera to start the preview. After the image is captured, the preview is stopped and then uploaded by using the Requests library in Python. The requests library allows the user to package the image file as a “POST” request that gets sent to the API. The image capture and the upload are all encapsulated inside a while loop so that it would upload a picture every minute to the server.

2.2.2.5 - Virtual Result

Here is the sample image and **video** of how the Pi Camera is utilized. When running the code, the screen “blinks” every minute, which indicates the Raspberry Pi was successful in taking the picture and uploading it to the server. The new picture will overwrite the previous picture on the server.

2.2.2.4 - Code to take Video

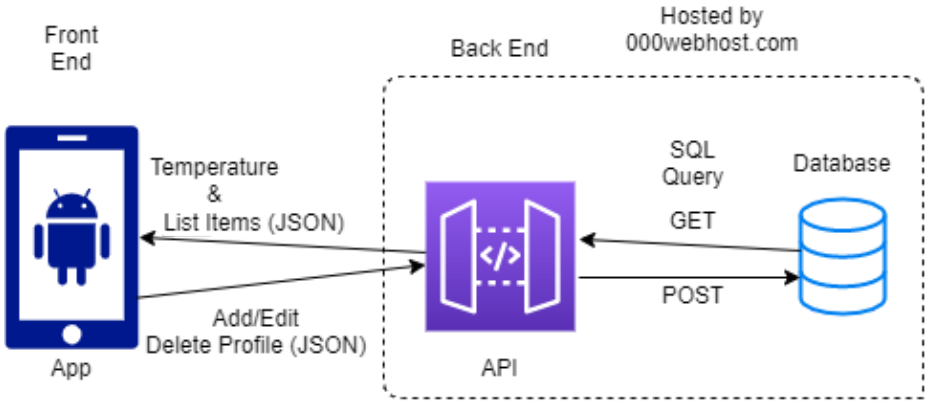
```
import picamera
from time import sleep

#Set Up Camera
cam = picamera.PiCamera()

#Sart Camera
cam.start_preview()

#Text to Display when taking the video
cam.annotate_text = 'Hello!!Welcome From BLue Jay Servie'

#Start Recording the Video
for i in range(40):
        cam.start_recording('/home/pi/Desktop/camera/video%s.h264'% i) #Directary to record the video with the location of poython code
        sleep(60)

#Stop Recording the Video
cam.stop_recording()

#When it stops, review the record
cam.stop_preview()
```

Sample Video with Title of “Hello, welcome from Blue Jay Service”

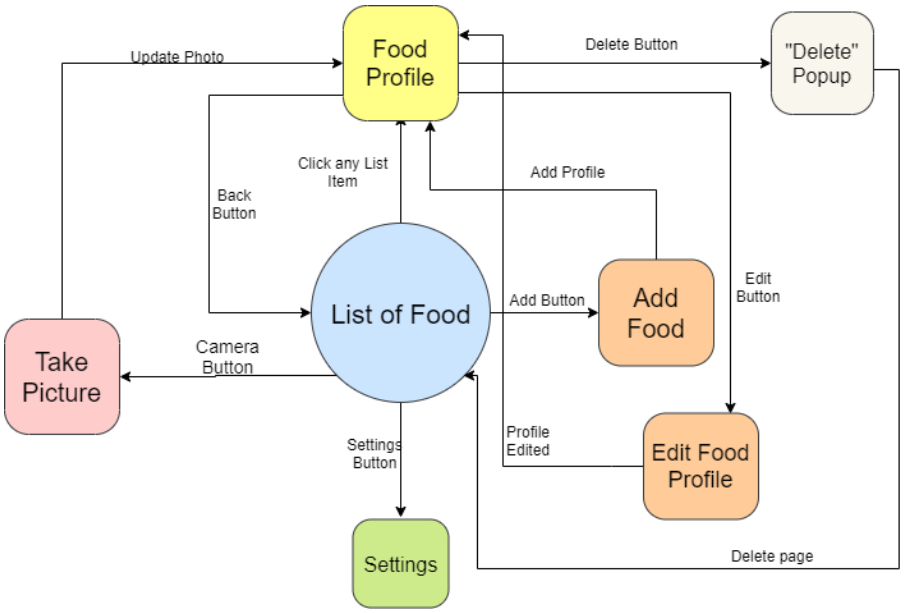
Video of Pi with Title

2.3 The App

The App has two portions: The Front End (that the user interacts with) and the Back End (the server that acts as a database). The Frontend of the App is developed using Android Studio with the Flutter Software Development Kit, with the Backend being made via PHP and is hosted on the 000webserver.com site.



Each shape in the UI graph below represents a page or a subpage (like a popup) and the arrows represent the actions taken when the user clicks on the button or item. The main menu is centered around the “List of Food”. When the list is empty, it will display a message saying that there are no items in the list. When the list has items, it will generate a profile for each item on the list. The information displayed on the profile will be retrieved from the database using the API.



2.3.1 Server Interaction

The Back End of the App is hosted on the 000webshost.com site, which facilitates the creation of databases that can interact with a given API. As a result, the server consists of an API and a database, which stores the fridge temperature, images, and food profiles. If the Temperature, or images from within the fridge are requested from the Front End, then the Back End of the App utilizes the API to execute an SQL Query to get the necessary information from the Database. Once a food profile has been constructed or updated, the Front End of the App sends the info to the API which posts the data into the Database.

The way our API works is that we have several PHP pages that are triggered using a link. For example, the page we use for retrieving the food list is “index.php”. This page does not do anything by itself when it is called. It only does something when the user has set the value of certain variables to a specific variable. It is similar to setting a state/mode (example: Setting data=1 will output a JSON list). When it comes to “POST” pages, they also don’t do anything

unless the user has specified the values that must go into the queries. For example, when the user wants to edit a profile, the app will send an http request to “edit.php”. The app will specify what row will get edited using the food id provided. The API will then insert the new data into that row using the data provided by substituting that into the SQL Query.

2.3.1.1 - Retrieve List & Temperature

```php
if ($_SERVER['REQUEST_METHOD'] == "GET")
{
    //echo $result["food_id"];
    if($_GET["data"] == 1)
    {
        $result = $conn->query("SELECT * FROM FoodProfile");
        $dbdata = array();
        while ($row = $result->fetch_assoc()) {
            $dbdata[] = $row;
        }
        echo json_encode($dbdata);
    }
    if($_GET["data"] == 2)
    {
        $result = $conn->query("SELECT temp_data FROM tempData LIMIT 1");
        $tempdata = $result->fetch_assoc();
        echo json_encode($tempdata);
    }
}
```

This code is a get method that consists of two parts: retrieving the food list and retrieving the temperature. When data=1, it retrieves all items in the FoodProfile table inside the database. The resulting rows are stored into an array, then encoded as a JSON object. When data=2, it fetches the temperature data from the tempData table. The value of the data variable is decided by the app, the API only responds based on it.

2.3.1.2 - Update Temperature Data

```php
if($_SERVER['REQUEST_METHOD'] == "POST")
{
    if(true)
    {   //this works
        echo $_POST["temperature"];
        echo $_POST["humidity"];
        $currentdate = date('Y-m-d H:i:s');
        echo $currentdate;

        $temp = $_POST["temperature"];
        $hum = $_POST["humidity"];


        $sql = "UPDATE tempData SET temp_data = '$temp',
        reg_data = '$currentdate', humid_data='$hum' LIMIT 1";

        if(mysqli_query($conn,$sql)){
            echo "updated successfully";
        }else {
            echo "Error updating:".mysqli_error($conn);
        }
```

This code is a method that posts to the database. It updates the temperature data in tempData table and sets the temperature, the time it was measured, and the humidity data (though it is not used). It only updates up to the first item in the table to make sure that it only updates once every time it is triggered by the Raspberry Pi. This makes the size of the table efficient, which is needed when we are using a free hosting service (using more data will mean spending money on the hosting service).

2.3.1.3 - Post Edited Profile

```php
$foodid = json_decode($_POST["food_id"]);
$name = json_decode($_POST["name"]);
$expiration = json_decode($_POST["exp_date"]);
echo $expiration;
    //this works
$sql = "UPDATE FoodProfile SET name = '$name', exp_date = '$expiration' WHERE food_id = '$foodid'";

if(mysqli_query($conn,$sql)){
    echo "Edited successfully";
}else {
    echo "Error inserting".mysqli_error($conn);
}
```

This PHP code decodes a JSON object and updates a specific profile based on the food id that was provided by the app. The update sets the new name and expiration date of the item. It works similarly to the process that was used for updating temperature data.

2.3.1.4 - Add New Profile

```php
date_default_timezone_set('America/Los_Angeles');
$currentdate = date('Y-m-d H:i:s');
$name = json_decode($_POST["name"]);
$expiration = json_decode($_POST["exp_date"]);
echo $expiration;
    //this works
$sql = "INSERT INTO FoodProfile (name, reg_date, exp_date)
            VALUES ('$name', '$currentdate','$expiration')";

if(mysqli_query($conn,$sql)){
    echo "Inserted successfully";
}else {
    echo "Error inserting".mysqli_error($conn);
}
```

This php code inserts a new profile into the FoodProfile table inside the database. It decodes the name and expiration date from the JSON object from the app, then passes it in through the SQL query along with the current date. By default the time zone is set to Pacific Standard Time.

2.3.1.5 - Delete Food Profile

```
$foodid = json_decode($_POST["food_id"]);

    //this works
$sql = "DELETE FROM FoodProfile WHERE food_id = '$foodid'";

if(mysqli_query($conn,$sql)){
    echo "Deleted successfully";
}else {
    echo "Error inserting".mysqli_error($conn);
}
```

This “POST” request sends a query to delete a specific food profile. Only the food id is needed because we are only deleting one row with this specific request. The food id is also used so that it won’t affect other profiles (example: if there were two bananas, deleting by name would cause both to be deleted, instead of just one).

2.3.2 Frontend Functionality

2.3.2.1 - Displaying Food Profiles

The food list serves as the default main menu screen for the app. It will display a message “List is currently empty” when there are no items in the list. The list is generated after retrieving the JSON object from the API.

```
return new Card(
  margin: EdgeInsets.all(10.0),
  child: ListTile(
    title: new Text(snapshot.data[index].name,
        style: Style.listTextStyle), // Text
    onTap: () {
      Navigator.push(
        context,
        MaterialPageRoute(builder: (context) =>
            DetailPage(snapshot.data[index])), // MaterialPageRoute
      );
```

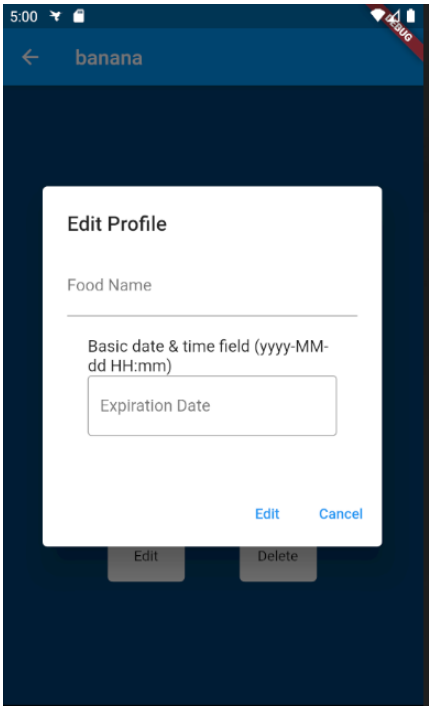
The list is displayed using a FutureBuilder widget. FutureBuilder widgets in Flutter are used for parts where the data is not yet present. It is a dynamic widget which means that the display can change. The list is also scrollable meaning no matter how many items you add to it, it can always be seen on the app. Since it is the main page, it's the first one on the bottom navigation bar.

2.3.2.1 - Creating Food Profiles

This page has several parts for the new food item to be successfully added by the API. This page has three main widgets: the name text field, the dateTimePicker, and the "Add Food" button. Unlike regular functions where the string data type can be passed through like regular parameters, the Flutter framework requires the use of a text controller in order to have data pass in and out of several widgets. Both the name text field and the

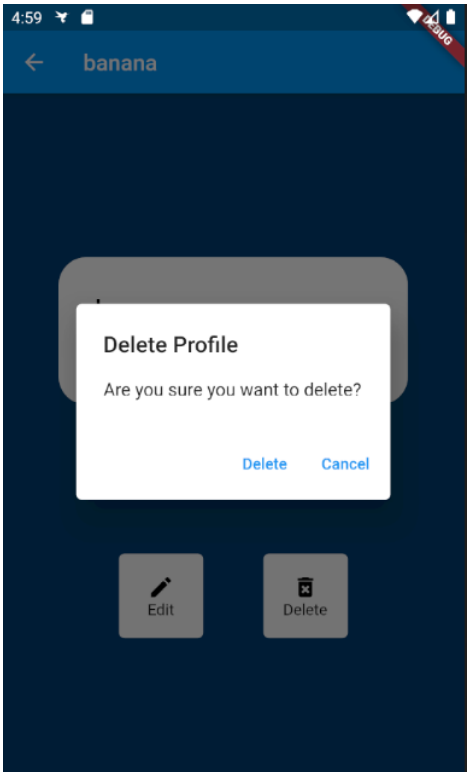
dateTimePicker makes use of controllers in order to package the strings into a JSON object.

2.3.4 Editing Food Profiles



The Edit Profile Popup makes use of an AlertDialog widget. This widget is only triggered when the user presses the "Edit" button on the food's profile page. Just like the "Add Profile" page, this popup makes use of text controllers to pass data from each field. When the user presses the Edit button on the bottom right side of the page, the App packages the data into a JSON object to the API. The user also has the option to cancel making edits, if they decided to change their mind.

2.3.5 Deleting Food Profiles



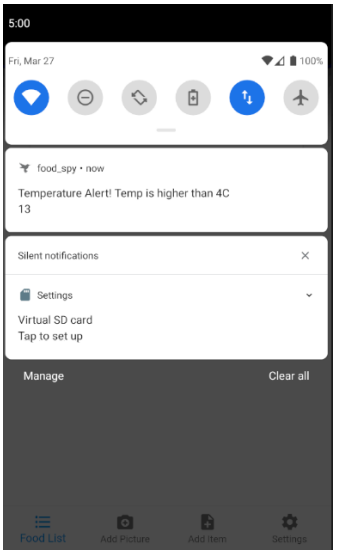

```
44    Future deleteProfile() async{
45      String food_id = jsonEncode(profile.food_id);
46
47      Map<String, dynamic> body = {
48        'food_id': food_id,
49      };
50
51      String postUrl = url + deleteData;
52      http.Response r = await http.post(
53        postUrl,
54        body: body,
55      );
56
57      int statusCode = r.statusCode;
58      String responseBody = r.body;
59      print("Posting Status: ${statusCode.toString()}");
60      print("responseBody: ${responseBody}");
61    }
```

The delete popup also works similar as the other popup. It

uses an alertDialog to show up. The difference is that instead of having two text fields, it just displays a message asking the user if they want to delete that specific food item. The code snippet shown is the function for the actual deletion process. It takes the food id of the item and then posts it to the API. We use the food id instead of the name because there could be an event where two items have the same name (example: two bananas). The food id is to differentiate with each item. That way when one item is deleted, it does not affect all the other ones.

2.3.5 Reading and Displaying Temperature and Hi-Temp Alert

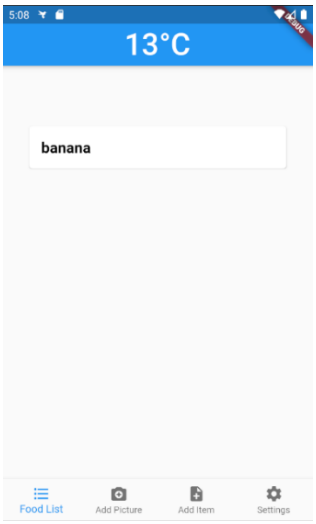
```
future: getTemperatureData(),
builder: (context, snap){
  if(snap.hasData){
    int currentTemp = int.parse(snap.data.temp);
    if(currentTemp >= 4){
      showOngoingNotification(notifications, title: 'Temperature Alert! Temp is higher than 4C',
          body: snap.data.temp);
    }
    return Row(
```



The Temperature Data (in Celsius) is read from the Database and passed from the Back End to the Front End via JSON. After this, the temperature is compared to the constant: 4 Celsius. If the current Temperature is greater than 4 Celsius, then an alert is given via an Ongoing Notification. Additionally, the code displays the current temperature, in Celsius, below the Alert in the same Ongoing Notification.

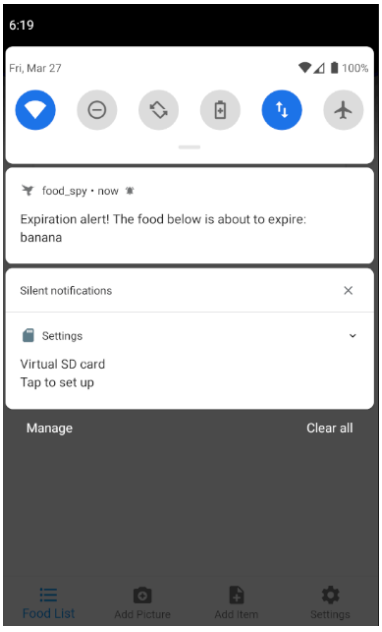
2.3.6 Displaying Temperature

```
appBar: new AppBar(
  title: new FutureBuilder(

    future: getTemperatureData(),
    builder: (context, snap){
      if(snap.hasData){
        return Row(
            mainAxisAlignment: MainAxisAlignment.center,
            children: <Widget>[
              Text(snap.data.temp, textAlign: TextAlign.left, textScaleFactor: 2,),
              Text("°C", textAlign: TextAlign.left, textScaleFactor: 2,),
            ]
        );
```



Regardless of whether an alert is sent or not, the current temperature of the fridge will be displayed in the menu of the App and is updated regularly at a 30 second interval. The process for retrieving the current Temperature of the fridge from the Database is the same as that of the High Temperature Alert in terms of coding, with the exception that the data is presented via the ’Text()’ command.

2.3.6 Food Expiration Alert



The food expiration alert operates on a similar concept as the temperature alert. When the items are decoded from the API, each individual item is compared to today’s date. When the date matches, the expiration alert triggers. It triggers no matter what the current time is, given that the expiration date matches the current date. When the app receives the food item’s expiration date, it is parsed into a string. The current date is also converted into a string when they are compared to each other. The reason

for this is because putting all the info into string allows for easier data management, compared to having a DateTime variable which can be represented in several different formats.

3. Tools

3.1 App

1. **Python:**

Jeremy and Yamin used python in the Raspberry Pi to collect data from the DHT11 sensor and Raspberry Pi Camera. The data is collected using the Adafruit_Python_DHT library, which is required to successfully interface with the DHT-11. We used that library because the original functions were in C, given their prior usage on the ESP-32, but the library converts it to python for the Raspberry Pi. For the Camera specifically, the Pi can communicate with the Camera via Python code, and this allows for the capture of images and even video via Python file executions. A key influence in our choice in using Python is because it is convenient to use with the terminal in the Raspbian Operating System for the Raspberry Pi.

2. **Komodo IDE (PHP):**

Abigail used the Komodo Integrated Development Environment in order to develop the server. Komodo can access the files hosted at 000webhost.com remotely. The server was developed using the PHP language with the MySQL database. The PHP language allows a developer to connect to and

manipulate databases. She chose this because there are many tutorials to learn from to develop in PHP.

3. **Android Studio (Flutter):**

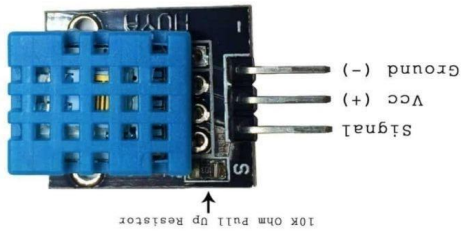
Abigail used Android Studio to develop the app due to its accessibility. Android Studio comes with a virtual device manager, which allows the developer to test their app without having to debug with a physical phone. The Flutter SDK is used with Android Studio. Compared to developing in Java, it is easier to learn. Flutter apps are made using the Dart language, which is a language developed by Google specifically for apps.

3.2 Hardware Listing

AstroAI Fridge Specifications

1. **Power Supply Voltage**: 12V DC
2. **Temperature Range**: ~32°F (0°C)
3. **Cooling Period**:
    a. Within 3 hours: ~32°F (0°C)
    b. Within 2 hours: ~50°F (18°C)
    c. Within 1 hour: ~64°F (18°C)
4. **Size:** 9.4 x 10 x 6.9 inches

5. **Inner Dimensions:** 5.5” x 5.3” x 7.85”
6. **Storage Capacity:** 4 Liters (~6 soda cans)



DHT-11 Temperature Sensor Specifications

1. Pre-Calibrated Digital Output Signal
2. **Power Supply Voltage**: 3.3V~5.5V DC
3. **Range**: 20-90% humidity, temperature: 0~50 degrees celsius
4. **Accuracy**: 1% humidity, 1 degree temperature
5. **Size:** 28x12x7.2mm (Amazon Variant) or smaller
6. Reference code and specific libraries are required for programming on every device, Pi included:

   ■ **Reference Code**: https://www.electronicshub.org/raspberry-pi-dht11-humidity-temperature-sensor-interface/

   ■ Library Needed for Code: https://github.com/adafruit/Adafruit_Python_DHT.git

Pi Camera Specifications

1. **Focal Length:** 2.1
2. **Camera Sensor**: 5 MP OV5647
3. **Power**: 3.3V
4. **Diagonal Angle:** 130 degrees

5. **Video Quality**: 1080 p @ 30 fps, 720 p @ 60 fps, 640 x480 p ● **User Manual:**

   https://www.waveshare.com/w/upload/6/61/RPi-Camera-User-Manual.pdf

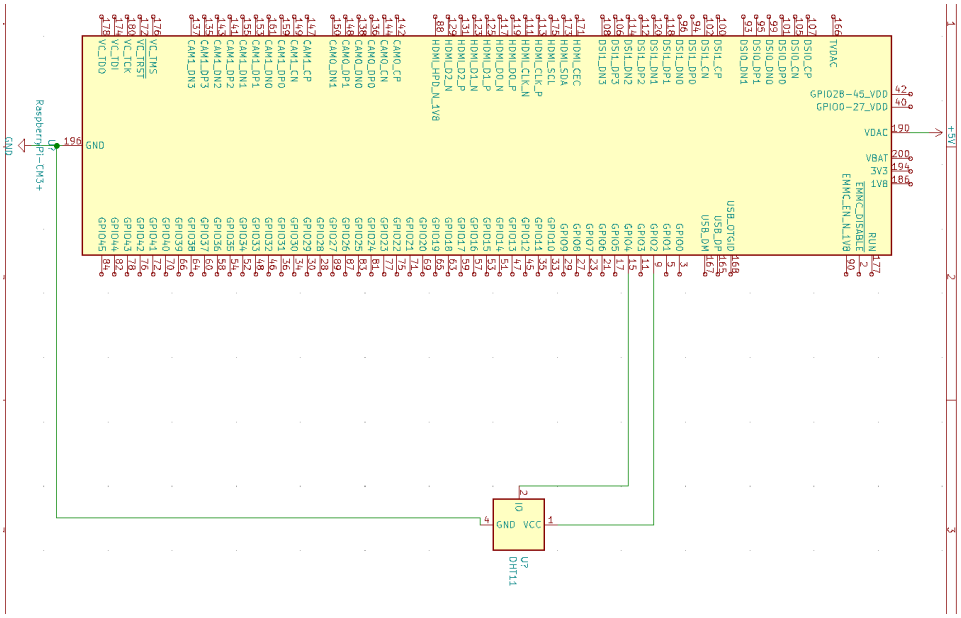
3.3 Cost Table

Item	Market Link	Quantity	Cost	Total
AstroAI Mini Fridge 4 Liter/6 Can Portable AC/DC Powered Thermoelectric System Cooler	Link	1	$49.99	$49.99
LANDZO Fisheye Wide Angle 5MP 1080p Night Vision Camera Module	Link	1	$13.99	$13.99
LED Light	Bought from EATS	1	$4.00	$4.00
HiLetGo DHT11 Temperature Humidity Sensors (Pack)	Link	5	(pack)	$10.49
ESUMIC DC 12V DIY Thermoelectric Peltier Refrigeration Cooling System Kit <Discontinued Use>	Link	1	$22.99	$22.99
3ple Decker Case for Raspberry pi (Blue)	Link	1	9.95	9.95

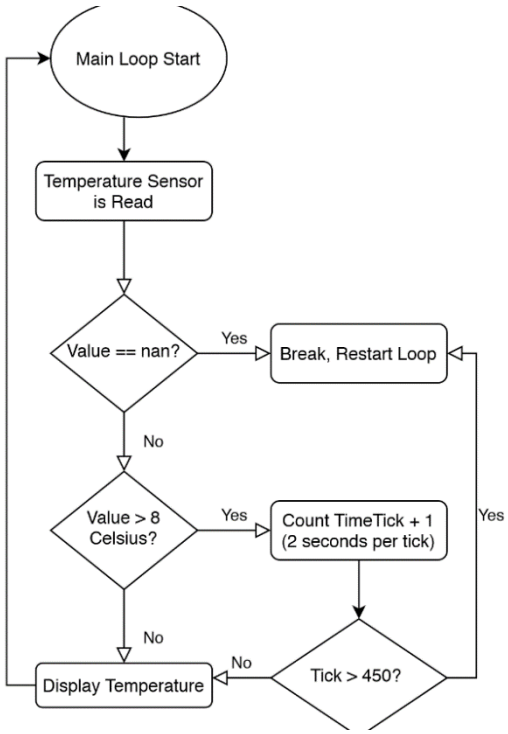
Low Voltage Labs - Raspberry Pi Camera Module Cable FFC FPC Ribbon 15-pin (50cm)	Link	1	5.00	5.00
Chanzon 100 pcs 5mm White LED Diode Lights (Clear Round Transparent DC 3V 20mA) Bright Lighting Bulb Lamps Electronics Components Indicator Light Emitting Diodes for Arduino	Link	1	5.67	5.67
Dorhea Raspberry Pi 3 b+ 4 b Camera Module Night Vision Camera Adjustable-Focus Module 5MP OV5647 Webcam Video 1080p with 2 Infrared IR LED Light for Raspberry pi 3 Model B+ 2 B 4B <Discontinued Use>	Link	1	16.99	16.99

Raspberry Pi Camera Module V2-8 Megapixel,1080 p <Discontinued Use>	Link	1	25.15	25.15
Total:				154.27

3.3 Schematic for DHT – 11 Connection

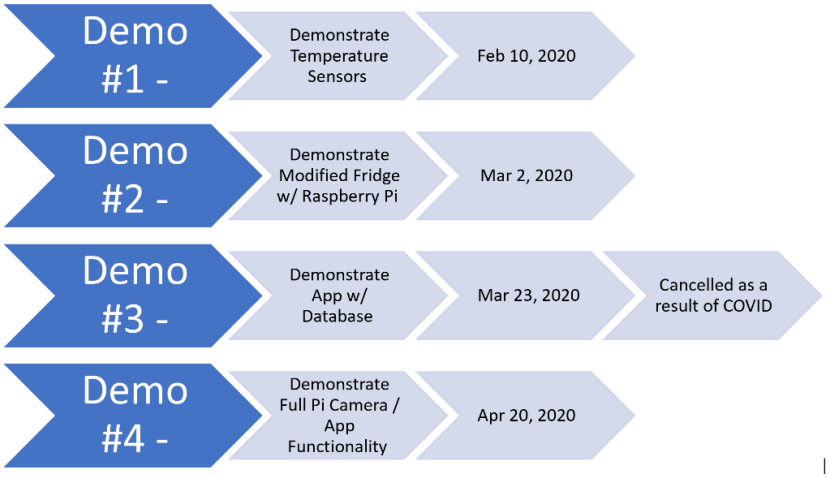


3.4 Flowchart Reference for DHT-11 Programming



4. Timeline

3.1 Demonstration List



3.2 Progress

Full Project Timeline

Name	Status	Timeline - Start	Timeline - End
Research Hardware Components	Done	2019-09-16	2019-09-22
Learn App Development Basics	Done	2019-09-16	2019-10-01
Research Raspberry Pi Capability	Done	2019-09-16	2019-09-26
Research Refrigerator Hardware	Done	2019-09-25	2019-10-01
Temperature Change Alerts (old system)	Done	2020-02-01	2020-02-14
Order Fridge Parts	Done	2020-02-01	2020-02-14
Create Database	Done	2020-02-02	2020-02-07
Accurate Temperature Reading	Done	2020-02-02	2020-02-07
Demo 1 Presentation	Done	2020-02-07	2020-02-10
Set up Raspberry Pi	Done	2020-02-07	2020-02-14
Retrieve Data from Database	Done	2020-02-07	2020-02-14
Create Static Menu for App	Done	2020-02-07	2020-02-14
DHT11 Connection to Pi	Done	2020-02-07	2020-02-17
Functions for Server/API	Done	2020-02-07	2020-02-21
Attach Pi to the Fridge	Done	2020-02-07	2020-02-24
DHT11 Raspberry Pi Schematic	Done	2020-02-14	2020-02-21
Profile Page Navigation	Done	2020-02-14	2020-02-21

5. Challenges and Solutions

5.1 Ethical/Legal

One notable ethical challenge our team had encountered was that of the FDA on the highest acceptable temperatures for food storage in fridges. This factors into our report through the Temperature Alert system, as the App will utilize a numerical constant to compare the current temperature of the fridge to. Given that the FDA's article recommends that fridge temperatures be at or below 40 degrees Fahrenheit (4 degrees Celsius), we utilized the Celsius number as our numerical constant to judge whether the current fridge temperature is high enough to be considered 'unsafe'.

5.2 Technological

One challenge our team had encountered was the programming of the DHT-11 and Camera file to act automatically and without requiring explicit commands for every execution of the file. This had been solved via the implementation of a CRON scheduler.

One technical challenge that had occurred during the project had been the memory of the Raspberry Pi, which had hampered our capability of utilizing the camera to take repeated images on 30 second intervals. With the assistance of Jeremy and Abigail, Yamin had modified the Raspberry Pi via Command Line to allow higher amounts of memory usage (128 mB to 256 mB).

Given the team's disparate class schedules and extracurricular activities, this has often left our teams separated physically and requiring contact via Discord or telephone. Additionally,

Google Drive didn't allow for the desired simultaneous work that Word had allowed, thus we utilized CSULB's online Microsoft Office to facilitate report creation.

5.3 Regarding COVID-19

A technical challenge had been the internet connection between the Raspberry Pi and the server given that all members of Food Spy were legally required to self-isolate at home as a result of the COVID-19 outbreak. The way we managed to overcome this is through the utilization of Yamin's phone hotspot to facilitate a stable internet connection to Abigail's web server.

With regards to team cohesion and roles, the required physical split had impacted our team's capacity to debug code/ troubleshoot hardware together. As a result, this required our team to have repeated Zoom meetings and Discord contacts when unoccupied by other classes, and at times limited the room for cross-role collaboration. We worked to overcome this challenge via live debugging, troubleshooting, and verification through Zoom, Discord, and Github.

6. References

Repository

1. Abigail Kwan's App Source Code
2. Yamen Yee's Pi Source Code
3. Jeremy Escamilla's DHT-11 w/ ESP32 Source Code (Used as Reference)

Fridge/Research

1. https://www.fda.gov/consumers/consumer-updates/are-you-storing-food-safely

Raspberry Pi

4. https://www.electronicshub.org/raspberry-pi-dht11-humidity-temperature-sensor-interfac e/
5. https://picamera.readthedocs.io/en/release-1.10/recipes1.html
6. https://www.raspberrypi.org/forums/viewtopic.php?t=126358
7. https://medium.com/@petehouston/capture-images-from-raspberry-pi-camera-module-usi ng-picamera-505e9788d609
8. https://projects.raspberrypi.org/en/projects/getting-started-with-picamera/1

DHT11

1. https://www.mouser.com/datasheet/2/758/DHT11-Technical-Data-Sheet-Translated-Vers ion-1143054.pdf

App/Server

1. https://flutter.dev/docs/development/ui/layout/tutorial
2. https://pub.dev/packages/datetime_picker_formfield
3. https://willowtreeapps.com/ideas/how-to-use-flutter-to-build-an-app-with-bottom-navigat ion
4. https://flutter.dev/docs/cookbook/networking/background-parsing
5. https://www.w3schools.com/tags/ref_httpmethods.asp
6. https://pub.dev/packages/flutter_local_notifications
7. https://medium.com/@nils.backe/flutter-alert-dialogs-9b0bb9b01d28