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

8th May 2020

Table of Contents

1. Introduction

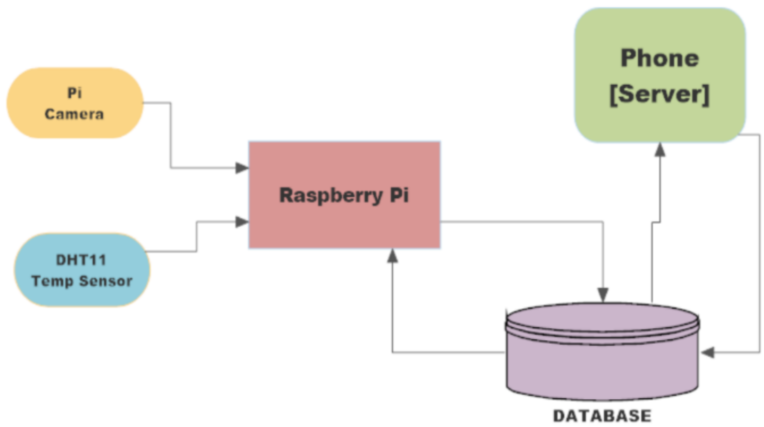
1.1 Executive Summary

This report will provide analysis about the Food Spy project and show insight into the workings of the Food Spy team. This includes details about each major portion of the project as well as a master timeline and hardware listing. This report will also include information about technical challenges we came across and how we solved them. The components of this project are 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: Frontend Design, API, Database

1.2 Project Description

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. The project was 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.



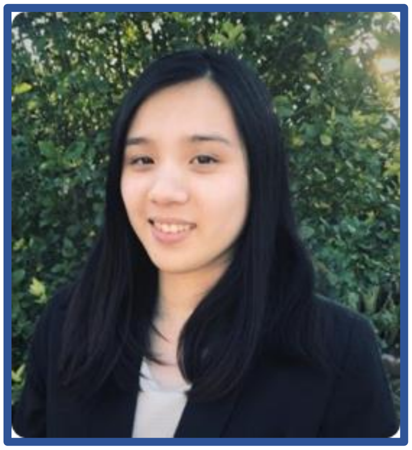
1.3 Mission Purpose

Team Blue Jay aims to incorporate an environmentally conscious focus with technology. We want to provide a convenient tracking system for many families who have a difficult time managing their food inventory. By making it more efficient to keep track of food spoilage, less food will be wasted. In today's world, people often juggle around multiple responsibilities at once: work, social life, and the dreaded bills. That is where we come in. We want to make it easier to quickly check the contents of the fridge, even when on the go!

To achieve our goals for food safety, we solved several problems:

1. Track the expiration date of an item of food.
2. Read the temperature inside the fridge.
3. Alert the user for when the temperature is reaching unsafe levels.
4. Alert the user for when the food has expired.

1.4 Meet Blue Jay!



Abigail Kwan

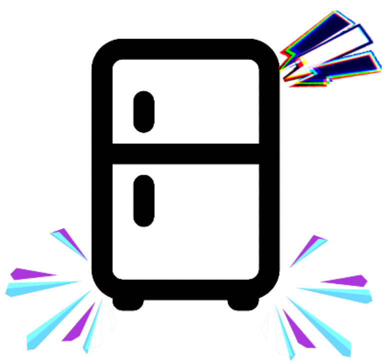
Yamin Yee

Jeremy Escamilla

1.4.1 - Abigail Kwan

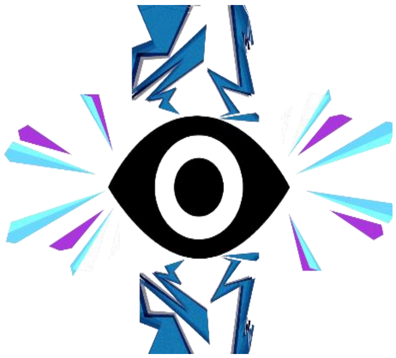
Abigail's role was to work as the software developer. This includes development of the frontend and backend of the App for the Food Spy project. This responsibility included the creation of the web server that stored the user data regarding the food in the fridge, the transfer of images from the Camera, and the storage (and processing) of Fridge Temperature.

1.4.2 - Yamin Yee



Yamin's role was to work on the Fridge hardware. This included the implementation of the final code for the Pi Camera, DHT11 sensors, and the interaction of the Raspberry Pi with the server. Additionally, she modified the fridge to facilitate optimal camera, sensor, and Pi placement.

1.4.3 - Jeremy Escamilla



Jeremy's role was to work on the temperature sensors. He was responsible for researching, programming, 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.

1.5 Team Contribution

1.5.1 – 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 timing for the Pi Camera image uploads

1.5.2 – 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

1.5.3 – 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

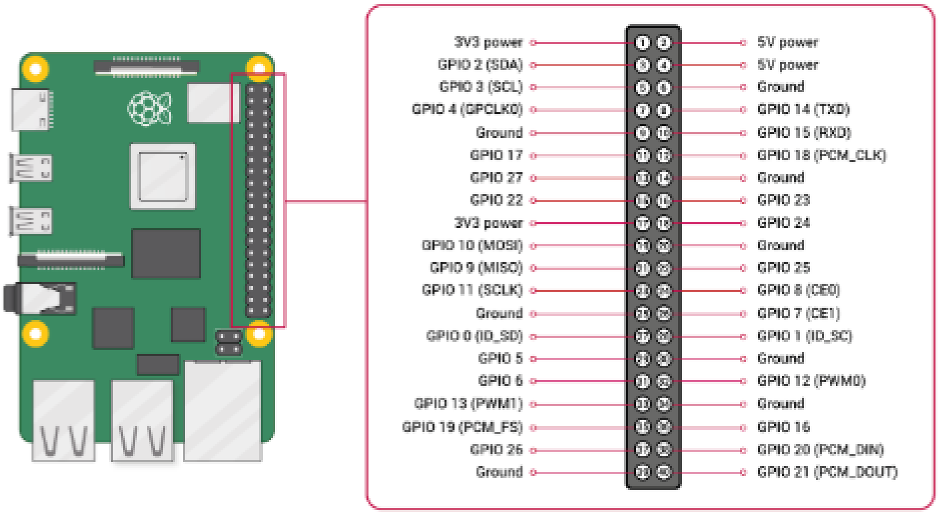
Instead of building a fridge from scratch, we decided to just modify an existing fridge. The fridge we chose is a mini thermoelectric system cooler. It comes with two power adapters: one for a standard wall outlet and the other is for 12V cigarette outlets in vehicles. This mini fridge is very portable 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+



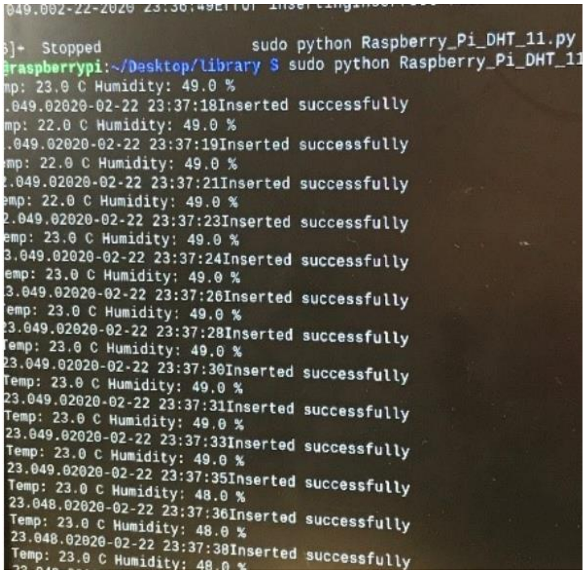
2.2 The Sensors

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 – Temperature Verification

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 from the DHT11 is sent to the API using the "requests" library for Python. That library allows for data to be sent to the frontend of the app via SQL Query. The results can be seen in the image on the right.



The repository for the sensors and the Raspberry Pi can be found in this link:

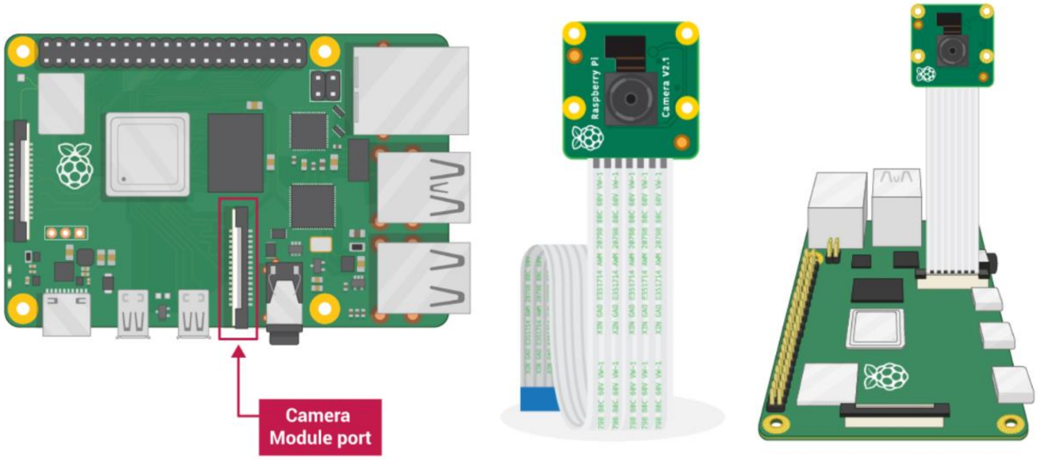
https://github.com/yaminshweyee/BLUEJAY_FoodSpy

2.2.2 The Raspberry Pi Camera Sensor

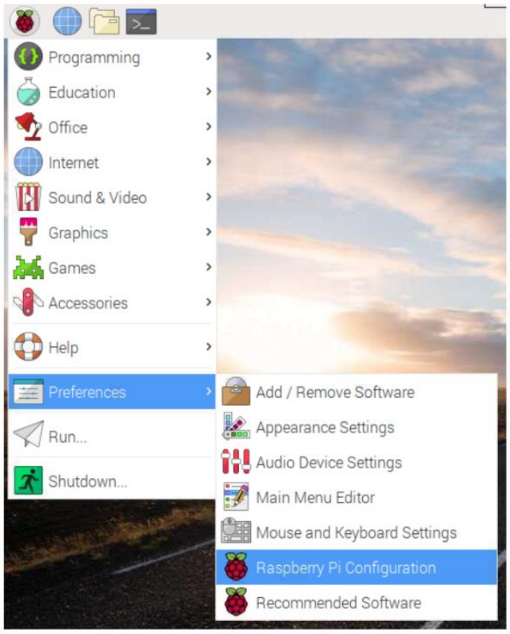
This camera module is what we used for the camera inside the fridge. Instead of using the official Pi Camera, which only has a specific field of vision, we chose to use this one for its fisheye lens so it can have a greater degree of vision. It also has LED lights attached to it so that it can see inside 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

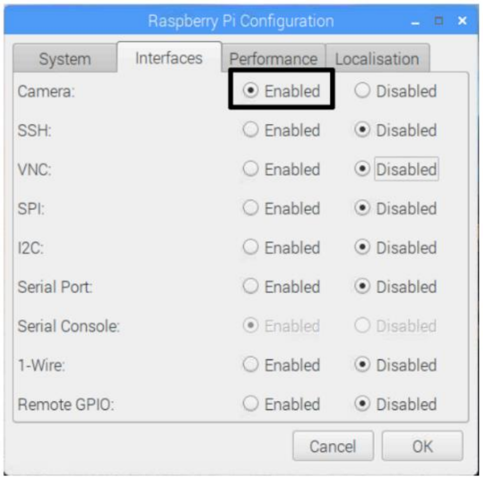


The diagram above shows how the Pi camera was arranged for the setup. After the setup, OpenCV must be installed.



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:

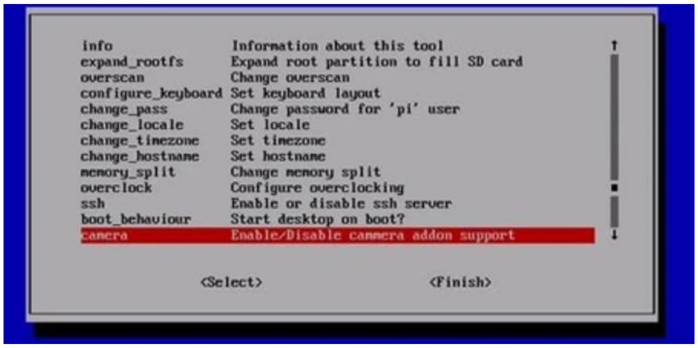
```
from picamera import PiCamera
from time import sleep

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 to make sure that the Pi is running properly.



2.2.2.2 – Camera 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)
```

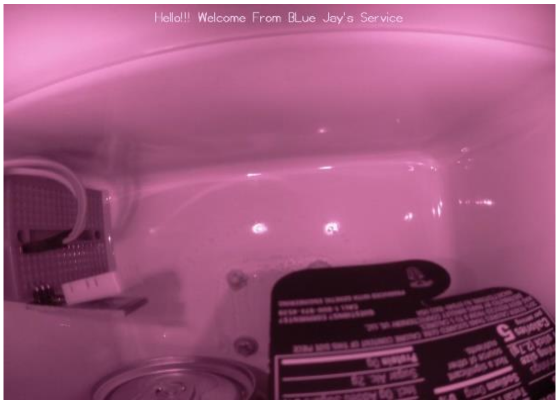
```python

#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.



2.2.2.6 - Code to take Video

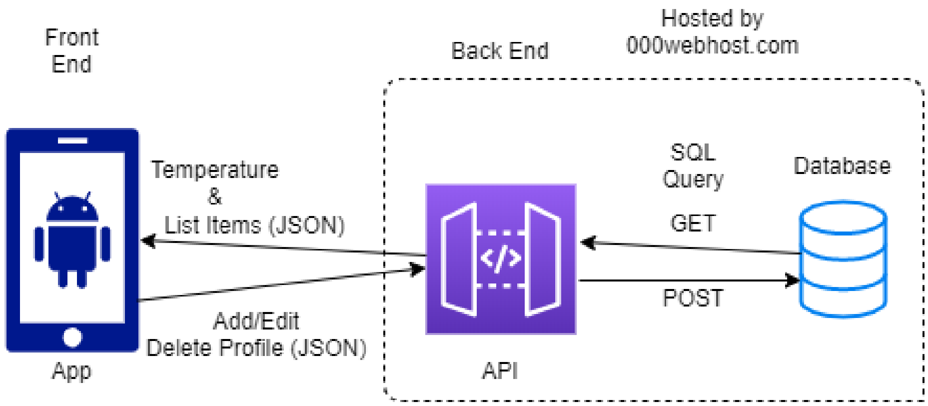
```
import picamera
from time import sleep

#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()
```

This code was also used to test the Pi Camera's recording capabilities. Although we ended up not using livestreams, this was still a vital piece in testing out what we would need to track food using the app. The link below is a sample video of this code in action.

Video of Pi Camera Stream

2.3 The App

The App section has two main portions: the frontend (the app) and the backend (API with database). The frontend of the App was developed using Android Studio with the Flutter Software Development Kit, while the Backend was made via PHP and is currently hosted on the 000webserver.com site. The code for the app is being hosted in this repository: https://github.com/abigailkwan/food_spy.

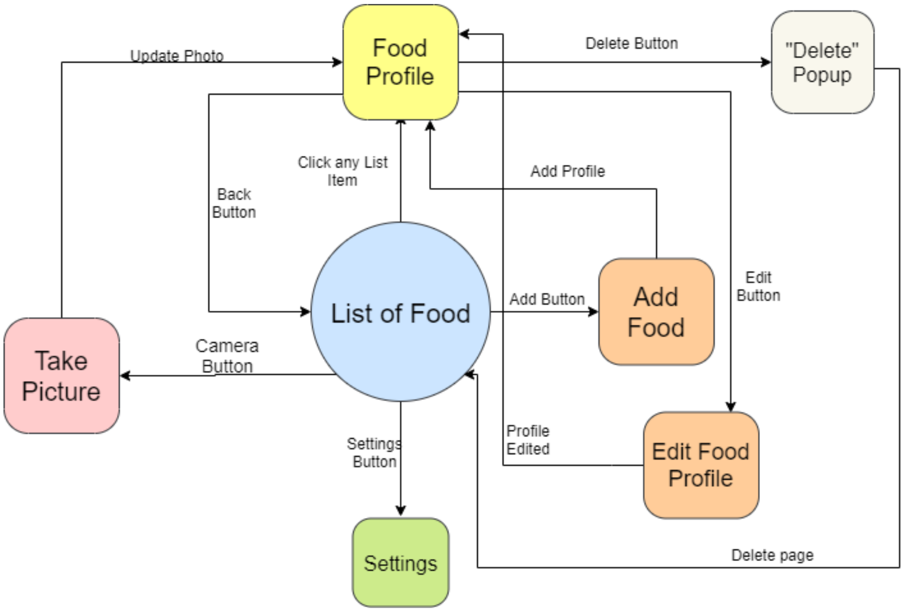


Frontend and Backend Integration Diagram

The app's main purpose is to integrate the data collected from the fridge and make it available for the user to use as they see fit. The app will allow the user to see the current temperature and an updated image of the inside of the fridge. The app must be able to:

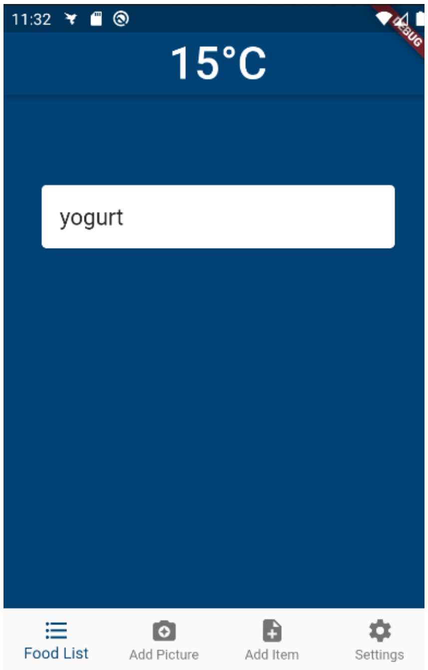
1. Display the fridge's current temperature.
2. Display the list of food items currently in the system.
3. Allow the user to add, edit, and delete food items.
4. Alert the user when the temperature is above FDA levels.
5. Alert the user when a food item is about to expire.
6. Display and add images from the fridge.

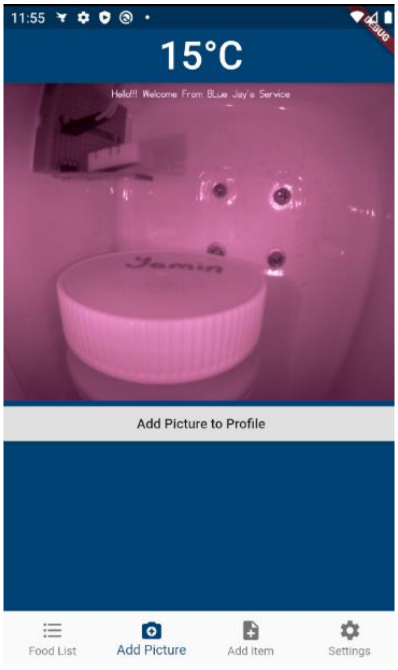
Before developing the app, we created a UI flow diagram as a plan for the structure of the app. Each shape in the UI diagram represents a page or a subpage (like a popup) and the arrows represent the actions taken when the user clicks on the button or item. The main menu is centered around the "List 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.

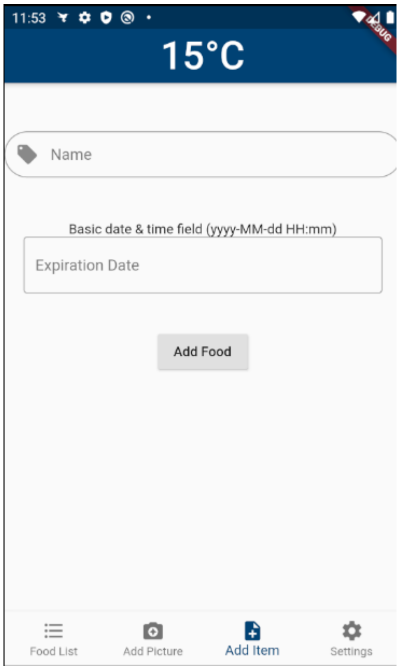


UI Flow Diagram

2.3.1 Frontend Design



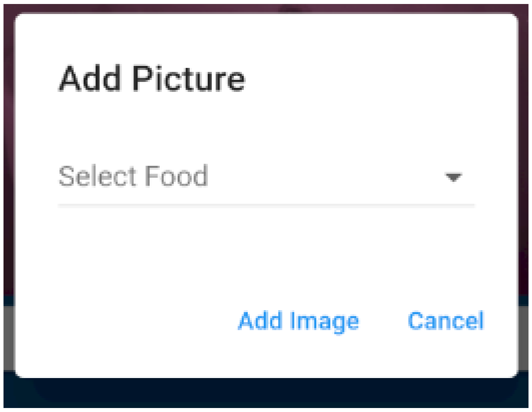


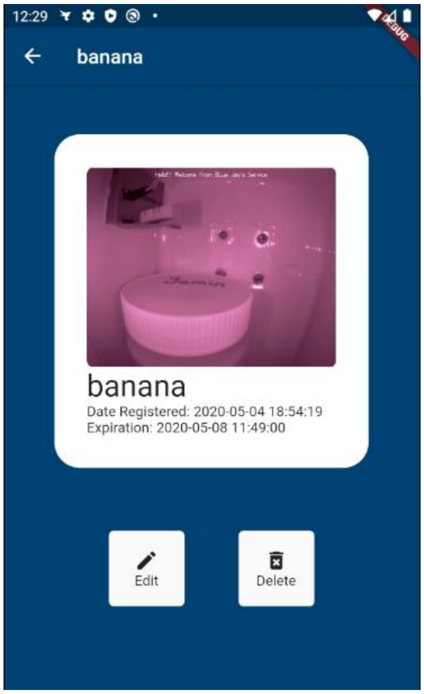


Home Page (Food List) **"Add Picture" Page** **"Add Item" Page**

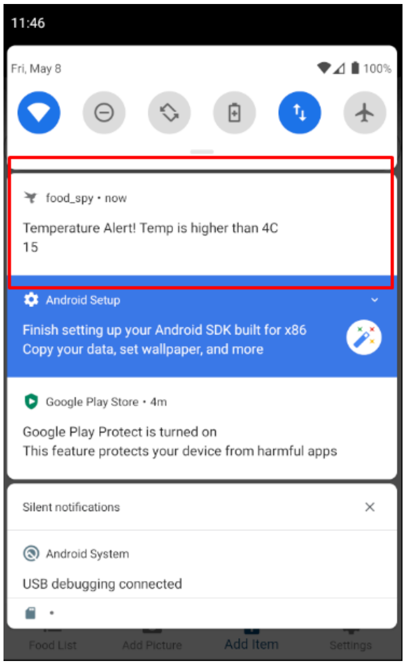
When the app is opened, the first page that is shown is the Food List. This is set as the home page because the app is centered around tracking food items currently in the fridge. The App displays the current temperature at the top of all primary pages. This ensures that the user always knows the temperature of their fridge. The bar at the bottom is a navigation bar for the three primary pages of the app: the Food List, the “Add Item” page, and the “Add Picture” page.

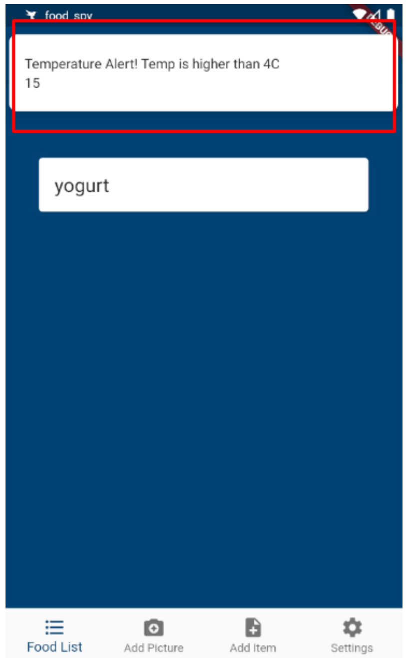
When the user wants to add an item to the list, they can go to the “Add Item” page which allows them to fill out the name of the item and its expiration date. Once it is completed, they click the “Add Food” button which adds the food instantly to the list. When the user wants to add an image to a specific profile, they go to the “Add Picture” page and select the item they want to modify from the drop-down menu.

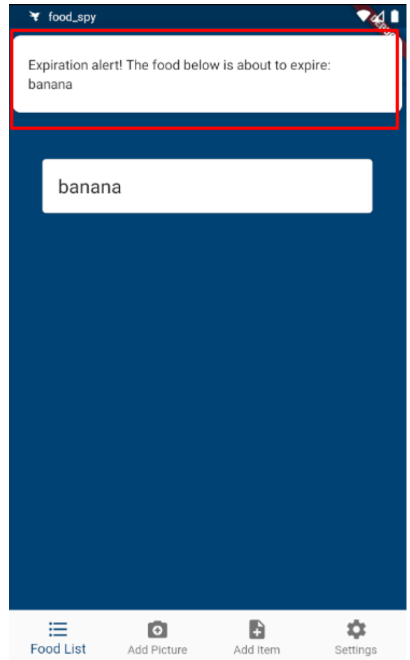




For viewing information about a specific food item, the user must click on a currently existing item in the list to display the profile page. The profile page lists important information about the item such as its name, registration date, and expiration date. It also shows an image of the item if the user added it. The user can also edit and delete from the same page. When editing an item, a popup will display where the user can enter in the new information. We designed the profile page this way so the user has a straightforward area for where they can view and modify information about their food.





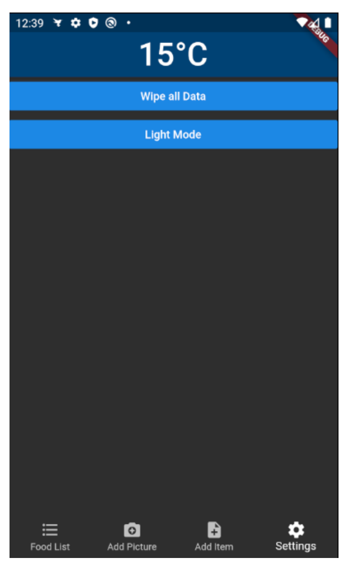


Alerts (Phone Notifications)

Internal Alerts (Temperature Notification)

Internal Alerts (Food Expiration)

Since we wanted the user to get alerted when their food has expired and when the temperature is too high, we created notifications both inside and outside the app. The left image above shows the notifications in the phone's quick settings menu. It displays it with a bird logo to indicate it is an alert coming from our app. The middle image above shows the alert given when the temperature is too high. The right image above shows the alert when a food is about to expire.



Lastly, we also added a settings page. Although it was not essential, we still added it to give the user even more customization and control of the app. It contains a "Wipe All Data" button for the user if they want to delete all their food information. It also has a light/dark mode button so the user can change the app's colors to fit their preferences.

2.3.2 API

The API is the mediator between the app and the database. It is connected to the server and executes SQL queries depending on the action taken by the user in the app. For example, whenever the food list is displayed, the app sends a request to the API to execute an SQL query to display all the items in the database. When the user wants to send data such as adding a new image or a new food item, the app connects to a specific PHP URL and packages the information into a JSON object that the API decodes. The code for the server is being hosted in this repository: https://github.com/abigailkwan/foodspy_PHP.

```
$conn = new mysqli($servername, $username, $password, $database);

if ($conn->connect_error){
    die("Connection failed: " . $conn->connect_error);
}

    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"]);
    $placeholder = "no_image_placeholder.jpg";
    echo $expiration;
        //this works
    $sql = "INSERT INTO FoodProfile (name, reg_date, exp_date, food_image)
                VALUES ('$name', '$currentdate','$expiration', '$placeholder')";
```

2.3.3 Database

The database stores the food profile information, temperature data, and images of the Food Spy system. As mentioned before, the data is manipulated by the API using SQL Queries. The information is sorted into different tables to make it easier to classify them for the API. For example, the *tempData* table stores temperature data sent by the Raspberry Pi. The *FoodProfile* table, on the other hand, stores information about a food's name, ID (for the system), expiration date, and registration date.

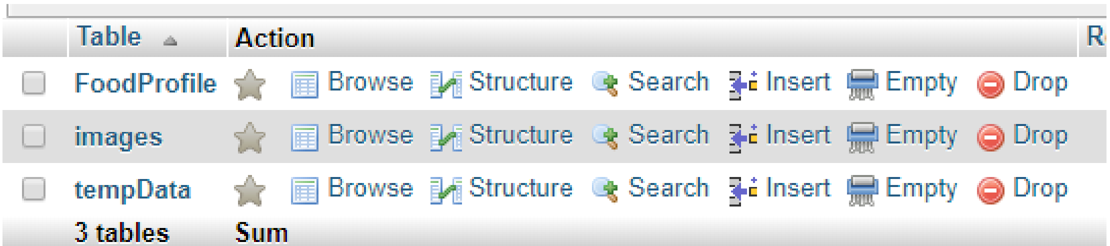
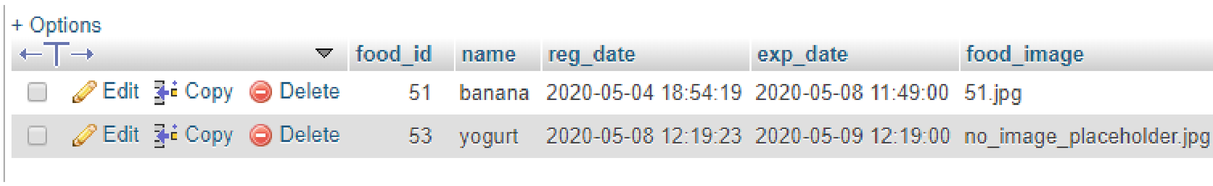


	Table		Action						R
☐	FoodProfile	★	Browse	Structure	Search	Insert	Empty	Drop	
☐	images	★	Browse	Structure	Search	Insert	Empty	Drop	
☐	tempData	★	Browse	Structure	Search	Insert	Empty	Drop	
	3 tables		Sum						



+ Options

				food_id	name	reg_date	exp_date	food_image
☐	Edit	Copy	Delete	51	banana	2020-05-04 18:54:19	2020-05-08 11:49:00	51.jpg
☐	Edit	Copy	Delete	53	yogurt	2020-05-08 12:19:23	2020-05-09 12:19:00	no_image_placeholder.jpg

The images in the *FoodProfile* table does not actually contain the physical data of the images itself. Instead, that table contains the directory location of the images. The images are hosted in a folder in the server where the API just retrieves it based on the location provided by the *images* table. For example, if an SQL query is executed to retrieve an image called "33.jpg", it will go to the *img* folder in the server and provide the URL link to the API. That same link is then sent to the app to display the image for the user. How the *img* folder looks when accessed in the server is seen in the image below. All the images are named for the food's corresponding id, so that each picture can be differentiated based on their profile.

☐	32.jpg	669.6 kB	2020-04-19 03:41:00
☐	33.jpg	669.6 kB	2020-04-19 03:46:00
☐	34.jpg	669.6 kB	2020-04-19 04:04:00
☐	35.jpg	668.0 kB	2020-04-23 01:08:00
☐	36.jpg	668.0 kB	2020-04-23 01:17:00
☐	37.jpg	668.0 kB	2020-04-23 01:36:00

3. Tools

3.1 App

1. Python:

Jeremy and Yamin used python in the Raspberry Pi to collect data from the DHT11 sensor and the 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 into python for the Raspberry Pi. For the Camera, this allows for the capture of images and videos via Python file executions. The key reason we used 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 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.

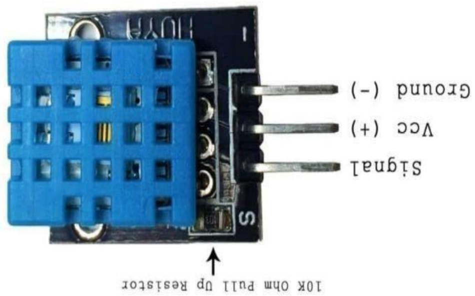
3. **Android Studio (Flutter):**

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. This SDK was used because it allows for both iOS and Android development using the same code. Flutter apps are made using the Dart language, which is a language developed by Google specifically for creating 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-s ensor-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:**

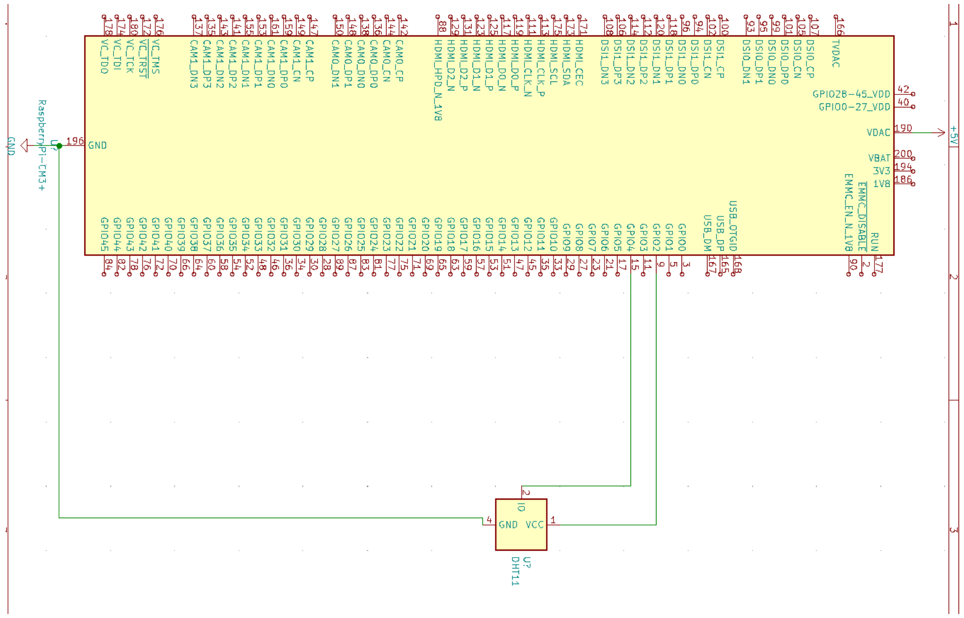
https://www.waveshare.com/w/upload/6/61/RPi-Camera-User-Manual.pdf

3.3 Cost Table

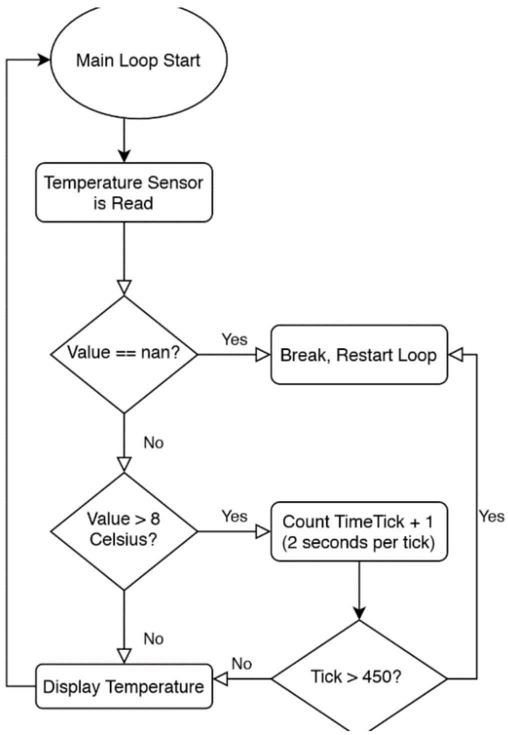
Item	Market Link	Quantity	Cost	Total
AstroAI Mini Fridge 4 Liter/6 Can Portable AC/DC Powered Thermoelectric System Cooler	Link	1	$49.99	$49.99
LANDZO Fisheye Wide Angle 5MP 1080p Night Vision Camera Module	Link	1	$13.99	$13.99
LED Light	Bought from EATS	1	$4.00	$4.00
HiLetGo DHT11 Temperature Humidity 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	Link	1	16.99	16.99

OV5647 Webcam Video 1080p with 2 Infrared IR LED Light for Raspberry pi 3 Model B+ 2 B 4B <Discontinued Use>				
Raspberry Pi Camera Module V2-8 Megapixel,1080p <Discontinued Use>	[Link]()	1	25.15	25.15
Total:				154.27

3.4 Schematic for DHT – 11 Connection

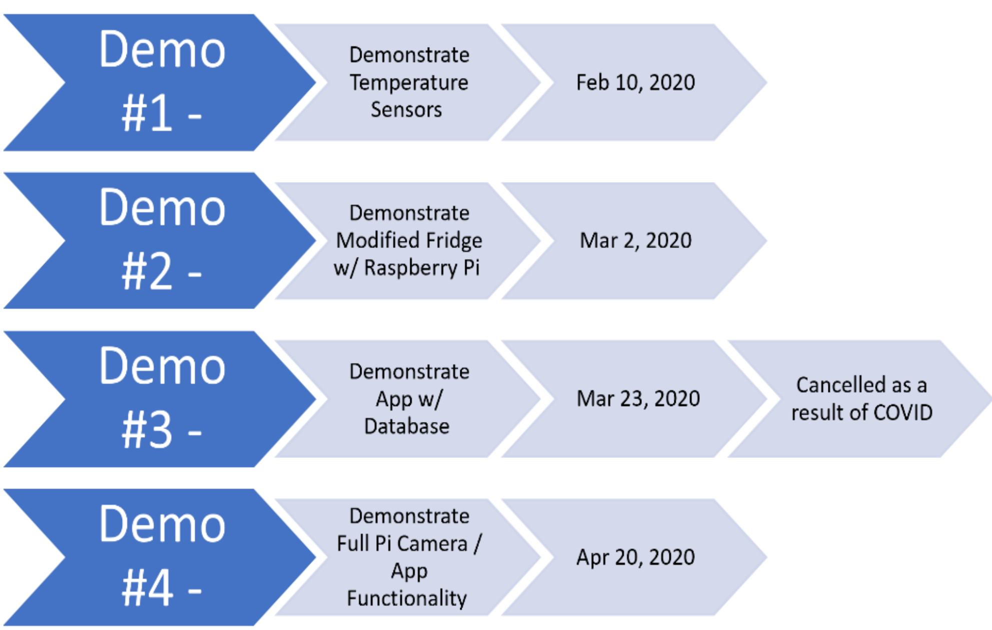


3.5 Flowchart Reference for DHT-11 Programming



4. Timeline

4.1 Demonstration List



4.2 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
Test Camera	Done	2020-02-14	2020-02-21
Set Up Camera with Raspberry Pi	Done	2020-02-14	2020-02-21
Drill Fridge to connect Hardware	Done	2020-02-14	2020-02-25
Setup IR Camera	Done	2020-02-21	2020-02-28
Create "Add Item" Page	Done	2020-02-21	2020-02-28
Add Bottom Navigation Bar	Done	2020-02-22	2020-02-28
Revise Diagrams for Report	Done	2020-02-22	2020-02-28

Revise Group Report for new Format	Done	2020-02-22	2020-03-06
Add "Select Expiration Date" widget	Done	2020-02-28	2020-03-06
Add Profile Text Field	Done	2020-02-28	2020-03-06
Demo 2 Presentation & Powerpoint	Done	2020-03-01	2020-03-03
Edit/Delete Buttons	Done	2020-03-06	2020-03-13
Incorporate Temperature into the app	Done	2020-03-06	2020-03-13
Settings Page	Done	2020-03-13	2020-03-24
Notifications	Done	2020-03-13	2020-03-24
Setup Local Stream of Pi Camera	Plan Changed	2020-03-20	2020-03-29
Progress Report 2	Done	2020-03-21	2020-03-27
Demo 3 PowerPoint	Done	2020-03-21	2020-03-24
Port Forward Home Router for Online Camera	Plan Changed	2020-03-27	2020-04-06
Connect App to Online Video Stream	Plan Changed	2020-04-06	2020-04-13
Set up While Loop for Image Capture	Done	2020-04-07	2020-04-19
Create Functions for Adding Image to Profile	Done	2020-04-14	2020-04-20
"Add Picture" section for App	Done	2020-04-12	2020-04-14
Demo 4 Presentation / Practice	Done	2020-04-14	2020-04-21
Test Entire System (App / Fridge functionality)	Done	2020-04-21	2020-05-01
Final Report	Done	2020-04-21	2020-05-08
Final Demo	Done	2020-05-01	2020-05-05
		2019-09-16	**2020-05-08**

Notes: We were able to finish our project, but we had to make a couple of modifications towards the 4th demo. Instead of a livestream, we decided to just use pictures taken every 1 minute. The specific solutions we made for this problem are discussed in the next section.

5. Challenges and Solutions

5.1 Ethical/Legal

One notable ethical challenge our team had encountered was the FDA standard for the highest acceptable temperatures for food storage in fridges. This factors into our project through the temperature alert system, as the App utilizes FDA requirements to monitor the current temperature of the fridge. Given that the FDA recommends that fridge temperatures be at or below 40 degrees Fahrenheit (4 degrees Celsius), we used that 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 *while* loop while the python file is being executed.

One technical challenge that had occurred during the project had been the connection between the camera and the app. The WiFi at Yamin's house would not work with a livestream so we had to switch to taking pictures every 1 minute instead. This led to another issue which was the memory size of the Raspberry Pi. 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). This allowed the Raspberry Pi to continually take pictures for the app.

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 Docs had some issues with formatting the pages, thus we utilized CSULB's online Microsoft Office to facilitate report creation instead.

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 because 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

Code Repository

1. Abigail Kwan's App Source Code
2. Jeremy Escamilla’s DHT-11 w/ ESP32 Source Code (Used as Reference)
3. Yamin Yee’s Sensors and Camera Source Code

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-interface/
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-using-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-Version-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-navigation
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
8. https://medium.com/@suragch/how-to-include-images-in-your-flutter-app-863889fc0b29
9. https://stackoverflow.com/questions/5772769/how-to-copy-a-file-from-one-directory-to-another-using-php
10. https://stackoverflow.com/questions/49273157/how-to-implement-drop-down-list-in-flutter
11. https://www.techcoil.com/blog/how-to-upload-a-file-and-some-data-through-http-multipart-in-python-3-using-the-requests-library/
12. https://stackoverflow.com/questions/68477/send-file-using-post-from-a-python-script
13. https://www.w3schools.com/php/php_file_upload.asp
14. https://stackoverflow.com/questions/11442779/copy-rename-a-file-to-the-same-directory-without-deleting-the-original-file
15. https://flutter.dev/docs/cookbook/forms/text-field-changes
16. https://pub.dev/packages/dynamic_theme
17. https://sergiandreplace.com/planets-flutter-adding-content-to-the-card/