Smart Refrigerator

E6765_2019Spring_MOHA_report

Xiren Zhou xz2754, Yiqian Wang yw3225, Maoxin Hou mh3895
Columbia University

Abstract

We have designed a smart fridge that helps users to monitor and track the real-time status of the fridge and get recommendations on foods and recipes. The fridge recognizes every single food that users put in or retrieve out. On our App, users are able to see the real-time collections of current food inside the fridge with corresponding information including food image, weight, calories, expected expiration date as well as some physical stats inside the fridge like temperature and humidity. Also, there are some recommended recipes based on what food they currently have. We have overcome several challenges including sensor deployment, threads scheduling, image recognition, label parsing, database establishing for food nutrition information, and some difficulty in data visualization and synchronization. We have successfully created an intelligent and helpful smart fridge that performs well in the real-time environment and has fairly high food recognition accuracy.

1. Overview

1.1 Problem in a Nutshell

With the significant development of IoT and deep learning, we are able to integrate these technologies more closely with our daily applications. For this project, our goal is building a smart refrigerator system which helps users to track the status of stored food and internal environment, and we can also provide some recommendations based on the current food.

To achieve this goal, we need to fulfill several system requirements: First, our system must be able to determine the category of the food that user puts into the refrigerator with image recognition. Then, we need to build an internal sensor node to collect and transmit the data which provide us the key information about the food and environment. Furthermore, we will process these data in the cloud, update them in real-time and get them ready for the queries from the users. Finally, it is necessary for us to create a mobile application for the user to conveniently monitor their fridge and get further information on their devices.

1.2 Prior Work

The pivotal part in our project is the food image recognition. The smart fridge must effectively and accurately recognize an arbitrary incoming food. This is essentially a deep learning problem. Over recent years, plenty of works have been done regarding image classification and recognition by deep neural network, including AlexNet, GoogleNet, VGGNet, ResNet, and etc.[1] Within our project, we apply AWS Rekognition[2], which is a powerful cloud service that performs great on visual analysis. The functionality of AWS Rekognition includes label detection of images, on which our project based.

2. Description

In this section, we are going to discuss the detailed objectives and the challenges we are facing in the every part, then we give the solution to these problems and demonstrate our design for the system.

2.1. Objectives and Technical Challenges

Our system consists of three parts: Hardware, AWS and User App. We will describe their functions and challenges separately.

2.1.1 Hardware

The objectives of hardware part:

1. Collect the real-time temperature and humidity data of internal environment. This helps us track of the environment of the fridge which is related to the storage status of the food.
2. Detect the different motion of the users, and we can initiate the corresponding process to get the necessary data.
3. Gather the data of the food, which includes the image of the food used for recognition, the storage date of the food and its weight measurement.

4. Transmit the data from sensor node to the server/cloud service.

The key challenges in this part:

1. We need to create a profile for every food in the fridge. It requires us to distinguish every food in the environment.
2. We need to determine the user's action is taking food out, putting food in or just checking what we have in the fridge .

2.1.2 AWS

In terms cloud service architecture design, the objectives are:

1. We need a cloud storage that receives real-time food images uploaded by the smart fridge. Also, it must stores at least images corresponding to all the currently available food items inside the fridge.
2. The cloud backend is capable for responding to the changes of the fridge state on a real-time basis. Specifically, changes of the fridge state include: putting a new food inside; retrieving a food outside; update on temperature and humidity data. The backend is able to determine the specific category of the food and furthermore give the nutrition information for the food item with low latency..
3. It need to maintain a real-time database for App/web users to access.

Some technical challenges are:

1. By means of AWS Rekognition, we are able to fulfill the image detection. However, for a given food image, Rekognition returns a list of all possible labels, but we merely need the most proper one. Specifically, for a clear input apple image, Rekognition gives an answer containing labels "Food", "Fruit", "Plant", "Apple", … We do not want the item be tagged as "Fruit", instead we want it to be labeled as "Apple".
2. Suppose we have successfully and correctly detect an apple. How do we get nutrition information about the apple? Specifically, it is not easy to directly determine the calories in the apple as well as the expected expiration date of it.

2.1.3 App

For the App part, we have these objectives:

1. Show the real-time temperature and humidity based on the data collected in the refrigerator.
2. Show which food in the refrigerator, and update the information of input or output as soon as possible.
3. Draw line charts for the temperature, humidity and daily calorie consumption.
4. Recommend recipes based on the food in the refrigerator.

Challenges for Web App:

1. Data transmission is the most difficult part in the App development. When the front end part uses axios to request POST data to the back end, it triggers cross-domain problem because the port numbers or the ip addresses are not the same.
2. Also, refresh page to request the updated data is hard because it takes some time to read the updated information in the DynamoDB, and it needs an asynchronous function when reading new data, or the data might not have enough time to be updated.

2.2. Problem Formulation and Design

There are three main parts in our system, and each of them has the different functions:

1. Sensor node in the fridge is responsible for

   1) Detecting the motion of the user
   2) Capturing the image of the food, and the image will be sent to AWS S3
   3) Weighing every food, then send the meta data along with the image
   4) Collecting the temperature and humidity data

2. AWS is responsible for

   1) Receiving and keeping the food image(S3)
   2) Detecting food label(Rekognition and functions in Lambda to derive the most relevant one)
   3) Analysing the meta data and storing the result into DynamoDB

3. APP is responsible for

1) Displaying the foods in the fridge
2) Real-time temperature and humidity and calorie consumption tracking
3) Recommendation for the recipe based on the current food in the fridge

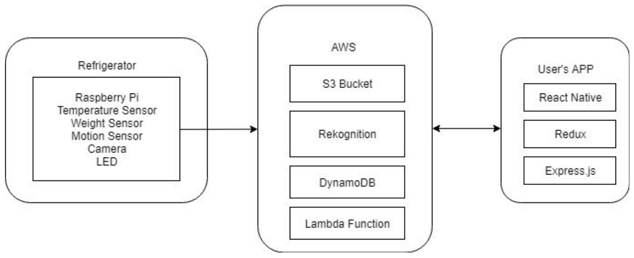


Figure 2.1 Block Diagram

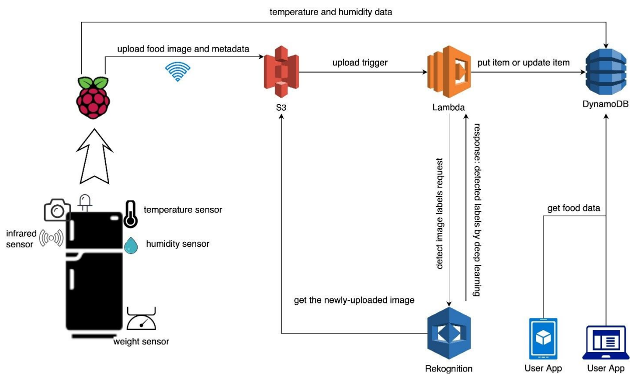

temperature and humidity data
upload food image and metadata
upload trigger
put item or update item
S3
Lambda
DynamoDB
detect image labels request
response: detected labels by deep learning
get food data
temperature sensor
humidity sensor
infrared sensor
weight sensor
get the newly-uploaded image
Rekognition
User App
User App

Figure 2.2 Flow Chart

2.2.1 Hardware Design

The first problem is to create a profile for every food in the fridge, but it is too hard to achieve this with a single image of the inside of the fridge. To simplify this problem, we choose to determine what is the changed food under every action of the user. That is to say, we need user to take the picture of single food every time they put it into the fridge. Thus, we can keep tracking the status of all foods.

The second problem is determining the actual action is taking out or putting it in. Since we can not use a single camera to fulfill this requirement, we introduced the weight sensor, a weighing platform made by ourselves, which weigh all the food inside the fridge, then we could determine the user's action according to the change of the total weights of the food.

Typically, we use the weight sensor to weigh all the food every time we detect there is an action of the user, and we denote the weight as *original_weight.* Then we will keep recording the measurement readings until it reach the timeout time we set. We define the lastest reading of the weights as *final_weight.* Then, we can derive a difference between these two weight readings:

$$diff = final\_weight - original\_weight$$

If the *diff* is close to 0 (there might be some small error which smaller than 1 gram on the readings), we could say no change are made on the foods, and we just need to wait until the timeout.

If the *diff* is positive and large enough, practically we set the threshold value to 3 grams, we can infer that user is putting the food into the fridge.

If the *diff* is negative, we can infer that user is taking some food out of the fridge.

For the implementation of the hardware:

1. We used the DHT22/AM2302 sensor to gather the temperature and humidity data, and we would initiate a separate thread to keep collect and send data.
2. A motion sensor/PIR sensor, HC-SR501, was used to detect the action of opening the fridge door.
3. The image of the food is generated by the camera module OV5647.
4. The weight sensor consists of a load cell, two glass fixed on it and the HX711 amplifier.
5. We chose Raspberry Pi to control all the sensors and establish a Wi-Fi connection to transmit the data. And we also tested the Bluetooth connection, which could get a good performance as well but needs an extra gateway outside the fridge to forward the data to the cloud.
6. We added a LED to indicate the working status of the system and guide the user to take the picture of the food. If the LED is on, it means the system detected the action of the user. Then the LED is going to blink, which tells user that the camera is going to take the picture.

2.2.2 Cloud Backend Architecture Design

We use Amazon Web Service(AWS) to build up our cloud architecture. The overall cloud architecture is demonstrated within Figure 2.2. There are four crucial components/services in our architecture, namely, S3, Lambda, Rekognition, and DynamoDB.

1. S3 is used to receive images uploaded by Raspberry Pi. It also keeps images corresponding to all currently available foods in the fridge. These images can be useful as the App demonstrates the food image icon for each item.
2. An upload into S3 triggers Lambda. Lambda sends the newly-uploaded image to Rekognition which detects labels for the image by deep learning.
3. Upon getting response from Rekognition which probably contains many valid labels for a single image, Lambda analyze all these labels and picks up the one that describes the food closest in terms of semantics. Refer to our example before, on receiving labels “Food”, “Fruit”, “Plant”, “Apple”, Lambda picks up “Apple” and tags the food item with string “Apple”. Once having determined what the food is, Lambda is able to give the amount of calories and expected expiration date for the food. Then Lambda integrates all the data(food label, time created, calorie, expiration date, weight, image URL of the food) and fill in the DynamoDB with the integrated data.
4. We keep two different DynamoDB tables, one for storing the food item information, and the other for storing the temperature and humidity records. The two tables are accessible for App designers to do data visualization and etc.

2.2.3 App Design

The Web App uses React Native as the front end framework, and use Express framework in Node.js as the back end program.

1. Node.js gets the data in DynamoDB and saves them as json files.
2. React Native uses Redux to deal with data in json files. And Redux has three parts, actions, stores, and reducers. When showing pages in the website, it triggers a request action to query data in a store. The store look up for data by reducer functions. Then transmits the data to React Native View.
3. React Native uses React Router to change pages, eg. the Home page, History page and Explore page. The page is changed by clicking a button in the menu bar.
4. The front end interact with back end by axios POST query. When refreshing the page, it automatically trigger a action and request new data in the back end by POST. When the back end receives the query, it looks up information in the DynamoDB, and downloading the latest data to the json files. Then the front end can load the lated data.
5. When finishing all the steps, the app needs to be built by webpack.build and packed into distinct html, js, png, css files. Then the program can be successfully upload to the server public folder.
6. Run the server on 3000 port. And we can view the website both on a PC or a smartphone.

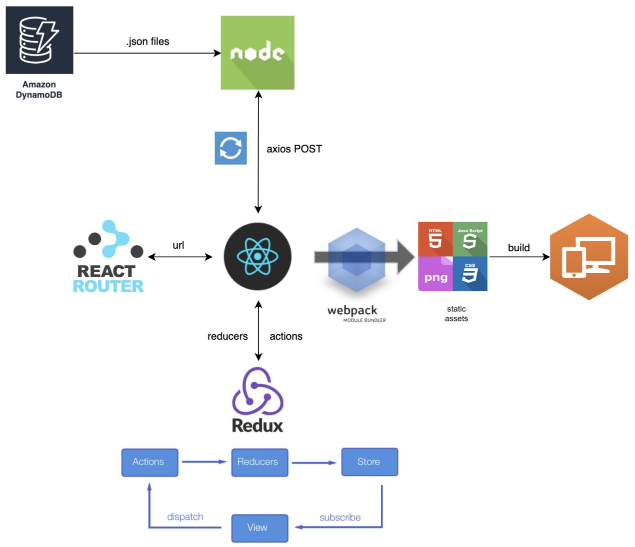


Figure 2.3 Flow Chart of App

2.3 Software Design

In this section we will focus on the detailed implementation of AWS backend and user’s App.

2.3.1 AWS backend implementation

As we illustrated in Figure 2.2, our system contains multiple Amazon Web Services.

S3 serves as a trigger for Lambda. There is a convention/agreement between hardware-end and AWS-end in terms of the naming of any food image that is to be uploaded into S3. There is no restriction on the format of the image file, but any image file uploaded into S3 bucket must obey the naming schema:

<in/out>_<timestamp>_<weight_of_food>_<username>.format

For example, in_1557241952_234_john002.jpg stands for a food that was put inside the fridge at time 1557241952 and has weight 234(in gram). The username is john002. And out_1557242666_410_xz2754.png stands for a food that was retrieved from inside the fridge at time 1557242666 and has weight 410g. The username is xz2754. This convention of naming guarantees the consistency and integrity of data in our data flow.

Lambda plays a pivotal role in our AWS architecture. The following is the pseudo-code for Lambda function which clearly describes the implementation of it.

```
def lambda_handler(event, context):# An upload into S3 bucket triggers lambda
    # Get the object from the event
    bucket = get bucket name
    key = get the filename of the newly-uploaded object from bucket
    # key: filename, a filename is expected to be of the form: "in/out_<timestamp>_<weight>_<username>.format"
    # (Note that in S3, it is a convention that key stands for the object / file name in a bucket)
    if key.startswith("in"): # correspond to a food that is put inside the fridge
        # call Rekognition to do label detection for the image
        response = detect_labels(bucket, key)
        # A response of the AWS Rekognition detect_labels() function is a dict containing a key named "Labels".
        if response['Labels'] do not contain "Food":
            raise Exception("The image is not detected to be of a food.")
        label = get_the_closest_label(response['Labels'])
        # Given many labels, get the most appropriate one and return,
            or in other words, the closest label in terms of semantics.
        # e.g., input ["Food", "Plant", "Fruit", "Apple", ...]. return "Apple".
        item = create an item to send to DynamoDB
        """
        item is a dict that has following fields:
        {"timestamp": <timestamp>,
         "label": <label>,
         "inside": True,
         "username": <username>,
         "imgurl": <image url>,
         "weight": <weight>,
         "calorie": <calorie>,
         "expirationDate": <expected expiration date>
        }
        this requires:
        1. parsing the key(filename of the food image);
        2. compute total calorie contained in the food;
        3. determine the expected expiration date for the food.
        """
        send item to DynamoDB table named "foodRecod"
    else if key.startswith("out"): # correspond to a food that is retrieved from inside the fridge
        set the corresponding item in DynamoDB table "foodRecord" and set the "inside" field to be False
    else:
        raise Exception("Image filename is not of the expected form!")
        # The naming of the uploaded image file must obey the mandatory schema
```

(Note that the pseudo-code here is actually a simple version that focuses on the important procedure and does not contain too much details such as error handling and etc.)

As we talked before, in lambda function, the two biggest challenges are:

1. get_the_closest_label(response['Labels']) which determines the closest label in terms of semantics, given a list of labels;
2. compute calorie and determine the expiration date.

Here we give pseudo-code for implementing get_the_closest_label:

```
def get_the_closest_label(labels):
  """
  Input a list of labels from the result of
detect_labels(), and return the most appropriate one
that describe the food best
  :return: a string of one label name
  """
  label = None
  max_parents = 0
  for each_label in labels: # find the one that has the
largest number of parents
      if {'Name': 'Food'} in each_label['Parents']:
          if len(each_label['Parents']) > max_parents:
              label = each_label['Name']
              max_parents = len(each_label['Parents'])
```

In terms of computing amount of calories, we have built up a database that includes calories per 100g data for nearly a thousands of different kinds of food. And we just load and look up the database and determine the total amount of calories within the food according to its weight.

In terms of estimating the expected expiration date, by far we just have fulfilled a simple version: we give an approximate value based on the larger category to which the food item belongs. For example, we give the expected expiration date for fruit, regardless of whether it is apple or banana or orange. We will get rid of this kind of solution in future work and come up with one by means of web crawler or database.

2.3.2 Web App Design:

1) Use axios POST request to get the latest data,

```
export const getListData = ()=>(dispatch)=>{
  axios({
    method: 'post',
    url: 'http://<IP addr. of server>:3000/api',
    data: {
      url: '<url for .json files in server>'
    }
  }).then((resp) => {
    dispatch({
      type: <List name>,
      obj: resp.data
    })
  });
```

2) Use sort() function to sort the temperature and humidity by #timestamp. Then use Google Chart API to draw the line chart of temperature and humidity.

```
renderTempItems(){
    let list = <Temperature List>;
    list.sort((a, b) => a.time - b.time);
    let tempdata = [["Time", "Temperature (℃)"]];
    list.map((item,index)=>{
      if (index >= Object.keys(list).length - 7) {
        tempdata.push([String(<function to translate
#timstamp>(item.time)), parseFloat(item.temp)]);
      }
    });

    return tempdata;

  }
```

3) After get the food data, randomly choose one or two unduplicated food, and request edamam recipe API to get the result of cooking recipes. Normally three recipes can be found.

```
  let items = <function to delete duplicated food
name>(food);

  index_1 = Math.floor(Math.random() * items.length);

  let item_1 = items[index_1];

  items.splice(index_1, 1);

  if (items.length > 0) {

    index_2 = Math.floor(Math.random() *
items.length);

    item_2 = items[index_2];

    combine = item_1 + '+' + item_2;

  } else {

    combine = item_1;

  }

request("https://api.edamam.com/search?q=" +
combine +
"&app_id=<MYAPPID>&app_key=<MYAPPKEY>&
from=0&to=3&calories=591-722&health=alcohol-free
"
```

3. Results

Based on the designs in the previous sections we have the following achievements:

3.1 Hardware

we had a fully functional hardware part which could collect the necessary data and capture corresponding image and sent them to the web services, as shown in the below Figure 3.1.

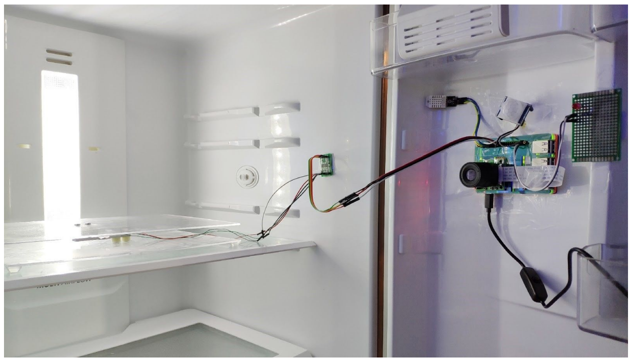
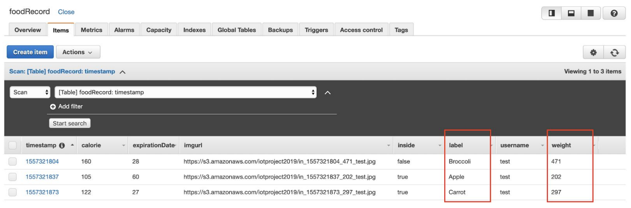


Figure 3.1 Hardware Layout

On the left side, it is our weighing platform which is used to place all the food. On the right side, we can see the Raspberry Pi and other sensors.

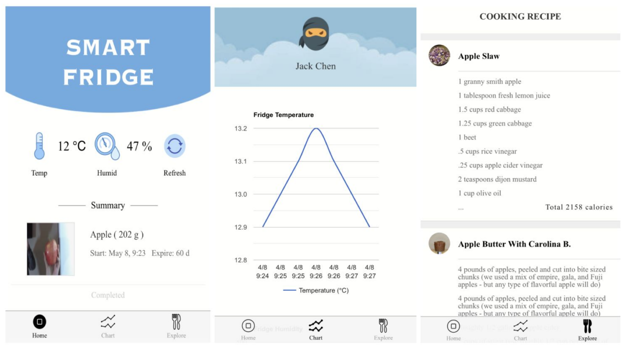
After the meta data and food image are uploaded to the AWS, it will trigger the Lambda Function and then Rekognition. Lambda Function processes the data, receives the recognition result and finally store all the information in the DynamoDB. As shown in Figure 3.2, we get the label, weight, timestamp and other necessary information in our foodRecord table.



timestamp	calorie	expirationDate	imgurl	inside	label	username	weight
1557321804	160	28	https://s3.amazonaws.com/iotproject2019/in_1557321804_471_test.jpg	false	Broccoli	test	471
1557321837	105	60	https://s3.amazonaws.com/iotproject2019/in_1557321837_202_test.jpg	true	Apple	test	202
1557321873	122	27	https://s3.amazonaws.com/iotproject2019/in_1557321873_297_test.jpg	true	Carrot	test	297

Figure 3.2 DynamoDB

Finally, the front end of web app is developed by React Native and the back end part uses Node.js Express framework. We have three main tabs in the App, the first tab displays the food information, temperature and humidity in the fridge. And the second tab has real-time temperature and humidity and calorie consumption tracking. For the last tab, it has recommendation for the recipes based on the current food in the fridge, and these recommendations will be updated automatically with the foods.



1 granny smith apple
1 tablespoon fresh lemon juice
1.5 cups red cabbage
1.25 cups green cabbage
1 beet
.5 cups rice vinegar
.25 cups apple cider vinegar
2 teaspoons dijon mustard
1 cup olive oil
...

4 pounds of apples, peeled and cut into bite sized chunks (we used a mix of empire, gala, and Fuji apples - but any type of flavorful apple will do)

4 pounds of apples, peeled and cut into bite sized chunks (we used a mix of empire, gala, and Fuji apples - but any type of flavorful apple will do)

Figure 3.3 App UI

4. Demonstration

For the layout of hardware in the fridge, we have a demonstration video:

https://www.youtube.com/watch?v=imrrzTbKiTo

For the complete working process of the system, we have a testing video:

https://www.youtube.com/watch?v=1YkmIRnyfyo

Our website for this project:

https://iotcolumbia2019moha.weebly.com/

5. Discussion and Further Work

One of the key performance metrics for our work is the accuracy of food recognition. Over nearly two months, we have tested the fridge on various categories of foods, including fruits, vegetables, meat, noodles, pizza, burgers, etc. Rare did it give wrong label. We observe that the AWS Rekognition gives wrong answers mostly when we put in a piece of meat. It is obvious and not surprising since many kinds of raw meat really have similar appearance.

The user is able to see all relevant data visualized on App. And every time the user refresh the page, he gets all most updated data(collection of current food items, food nutrition information, expiration dates, temperature & humidity curve, relevant recipe recommendations) with low latency. What we are possibly going to improve in the future is add a trigger from Lambda to App: once there is any update for food data, Lambda triggers the App to automatically update its data visualization.

Our implementation for estimating the expiration date is to be improved, we will either develop a python crawler approach or deal with it by means of database.

Moreover, we plan to add more powerful functionality to the current version of our smart fridge: enable "foods to buy" based on what the user has bought recently. This is a tough problem because it has something to do with "healthy diet plan".

For now, the users are limited to change one food every time they open the fridge door, it is a little bit inconvenient. Thus, we could add physical buttons inside the fridge in the future, so that users are able to use buttons to control the image capturing of multiple foods, and they can also re-capture the image if they did not place the food in the correct place in time.

Furthermore, the web app can be transformed into a hybrid app, and then can enhance the user's experience. For instance, we can create a new page which is triggered by the click or touch on the food item in the home page, which includes more detailed information of the food and allows users to modify the informations on their demand.

6. Conclusion

We have built up an intelligent fridge which has a high recognition accuracy for majority of food classes. The data is updated in real time with low latency. The operations for using the fridge is easy and friendly for users. The backend design is robust and efficient and has high scalability.

We have spent a lot of time and effort dealing with hardware issues. We now have rich experience in coping with physical data fluctuation and unstability. Also we get much more familiar with cloud computing and have obtained experience in AWS architecture design and realization. What's more, we introduced and learned some state-of-art front-end development techniques while designing the App.

We will do some further improvement to our smart fridge, including estimating a more accurate expiration date, warning/alarming user when any food reaches expiration date, enabling users to edit or modify the food information data if they happen to think of it as incorrect or inaccurate.

7. References

[1] Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556.

[2] Working with Images - Amazon Rekognition. (2019). Retrieved from https://docs.aws.amazon.com/rekognition/latest/dg/images.html