**Week 5 – Update (22/10/2018)**

This week we began researching how we would complete this project. Following the lecture on Friday during which we set up the Raspberry Pi’s and began playing with Node-Red. I started having a look and playing around with the Google Cloud vision API and seeing if I could get Node-Red to send an image to this in order to see if it could be recognised. I managed to get this working after copying some sample code and then modifying it to suit my needs. I started of by entering a URL to an image I found online and then sending this to the Cloud Vision API and the recognition seemed to be pretty good. I had a go at trying to get it to work with an image from the live webcam although unfortunately I had problems getting the web cam to work with the browser on the Raspberry Pi which made it difficult to send the image. I did however manage to create a Python Script that took an image locally and then sent that to the Node-Red program which was communicating with Google. During the Friday lecture Stavros also showed us IBM’s image recognition feature for us to consider which appeared to work similarly to the Google Cloud Vision API (although his example didn’t seem to be working). If we have any problem with the Google Cloud Vision API we will be able to consider this and use it as a back up although we will probably stick with Google because I have a fair bit of experience using Google Cloud plugins and as this API is used fairly widely with similar software being used in the Google Lens feature on Android I felt it would probably be the better API.

**Week 6 – Update (29/10/2018)**

This week I returned to working on my computer as the Raspberry Pi screen was a bit small to program on and also the Pi was starting to struggle with the number of tasks I was trying to run. My focus for this week was to get the scanner system working properly. My first task was to get the browser sending an image to Google Cloud Vision whenever the user scanned an image. To do this I started off trying to send the video feed directly to a HTML canvas element as I knew then that I would be able to start modify the image and also save it in a base64 format that could be sent to Node-Red. I had difficulty getting this to work before I found a tutorial that showed me how to read a camera feed into a standard video element and then save a screenshot from that feed into the canvas element on the call of a function. During this time, following a discussion with Stavros we also decided that most of the functionality would be done in the cloud as he had suggested using the data on a wide scale to help organisations such as the NHS collect data on how healthily people were eating. Because this would involve storing all the data from the fridge in the cloud I decided it was probably easier and better to stick with the Node-Red session I had set up with IBM Bluemix as this would make it easier to communicate with other elements already stored in the cloud and would reduce the workload on the Raspberry Pi as we would not be needing to run Node-Red locally. It also means that we can reduce the data being sent to the Raspberry Pi as we will be able to only send information that is required by the interface.

**Week 7 – Update (5/11/2018)**

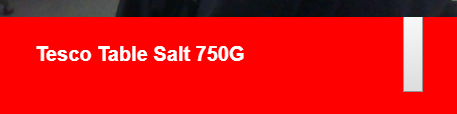
This week having got my computers web cam working and sending an image to Node-Red I now needed to get the Raspberry Pi to do the same. I spent a long time trying to get the web cam working with the browser on the Raspberry Pi which was difficult because the browser wasn’t recognising the Camera Module we were using as a webcam although we were able to make it work using a Python Script. I tried several methods to try and get the camera working with the browser including broadcasting the feed to a server which could then be accessed by the browser although this would probably have made if difficult to save the image in the canvas element. In the end I found a command that I could run on the Raspberry Pi[[1]](#footnote-1). This basically changed the software running the camera module meaning that it was now working the same as a webcam and could be recognised by the browser.

After doing this I started looking into how I could get a barcode scanning system to work. This was harder than I thought as a lot of scanners that are built on the mobile platforms rely on the autofocus feature which a lot of webcams and also the camera module for the Raspberry Pi, do not have. I tried a number of different JavaScript plugins[[2]](#footnote-2) [[3]](#footnote-3) as well as having a go at building a Processing sketch which could be used in a canvas element using P5. Eventually I decided to go with the Dynamsoft API which seemed to work quite well when dealing with a still image. I then had to modify the code I already had in place to take an image so that it would also scan this image to look for a barcode. I decided to check if there was a barcode before sending the request to Google Cloud Vision so that I could limit the number of server requests as I assumed that if there was a barcode in the image then this would probably be able to identify the product. I eventually managed to get this working and so was able to start working on the other features of the site. As I was waiting on Yuzhou to complete the interface designs I put together a basic template which included all of the features but didn’t look very appealing. I then set to work on the other backend features of the site.

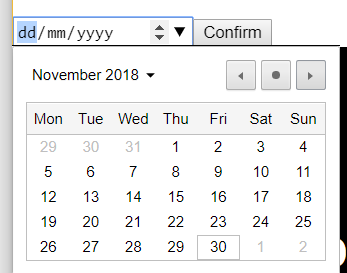


**Week 8 – Update (12/11/2018)**

During this week I was able to start work on the backend database part of the fridge. I decided for this that I would create a number of PHP scripts which would control all of this functionality as I knew I would be able to easily connect these to the SQL database I was using and because it would all be running on the same server, I knew it would be more secure which is essential when dealing with user information such as this. I started by building the system that would store the food items that had been scanned into the fridge. I did this by creating a button in the interface that would allow the user to add in their scanned food once it had been identified.



Because one of the features of the fridge that we had discussed had been the idea of encouraging users to use up food before it’s use-by date I needed to find a way of getting data about a products use-by date. I had originally hoped that I would be able to get this data through the Tesco API however I found that this data was not available. I therefore had to find another solution to this issue. I ended up building a system that would ask the first 3 users to scan in a particular item to also enter the sell-by date on the packaging. Using this data as well as the date that the item was entered, I am able to assume how many days the item is expected to last. Using this data, I can then work out the average life-expectancy of individual food items which can then be used to work out the use-by date. Obviously, this solution isn’t perfect as it assumes that the item is being put into the fridge whilst it is still fresh. There could also be issues if the date is entered incorrectly and I found that I had to develop a special system for entering the date to ensure that it could distinguish between the UK and US format of the date. A solution to this problem would be to develop a system that would look out for any outliers in the average and then remove those from the calculation. I also had to work around the limitations of the JavaScript Date() function which I knew would struggle to know if the user was entering the date in the format dd-mm-yy, mm-yy or even mm-dd-yyyy. I therefore built a system that would present the user with a calendar that they could enter the sell-by date into.



1. https://forum.astroprint.com/t/raspberry-pi-camera-support-solved/62 [↑](#footnote-ref-1)
2. https://github.com/andrastoth/WebCodeCam [↑](#footnote-ref-2)
3. https://www.dynamsoft.com/Products/barcode-recognition-javascript.aspx [↑](#footnote-ref-3)