OLLSCOIL TEICNEOLAÍOCHTA
BHAILE ÁTHA CLIATH
DUBLIN
TECHNOLOGICAL
UNIVERSITY DUBLIN

TU Dublin - City Campus Online Dissertation Copyright Policy

The TU Dublin City Campus Library Online Dissertation Collection is comprised of undergraduate and postgraduate taught dissertations.

The copyright of a dissertation resides with the author. All TU Dublin students are bound to comply with Irish copyright legislation during their course of study. A dissertation is made available for consultation on the understanding that the reader will not publish it in any form, either the whole or any other part of it, without the written permission of the author.

To learn more about copyright and how it applies to you click **here**:

The Online Dissertation Collection is only available to be accessed by registered TU Dublin City Campus staff and students. Non-members of TU Dublin cannot be given access to the Online Dissertation Collection. (Please note that this also applies to graduates of TU Dublin and graduates of the former Dublin Institute of Technology requiring a copy of their own work). In using the Online Dissertation Collection students should be fully aware of the following points:

- Usernames and passwords are for personal use only. They must not be divulged for use by any other person.
- The Online Dissertation Collection is provided for personal educational purposes to support you in your courses of study. Use of this collection does not extend to any non-educational or commercial purpose.
- You must not email copies of any dissertation or print out copies for anyone else. You must not under any circumstances post content on bulletin boards, discussion groups or intranets etc.
- You must not remove, obscure or alter in any way any copyright information or watermarks that appear on any material that you download/print from the collection.

TU Dublin City Campus Library has to right to temporarily or permanently remove a digitised dissertation from the collection.

OLLSCOIL TEICNEOLAÍOCHTA BHAILE ÁTHA CLIATH

TECHNOLOGICAL UNIVERSITY DUBLIN

Recipe Search Engine with Image Processing

Final Year Project Report

TU856

Computer Science

Catriona Renaghan

Bojan Bozic

School of Computer Science

Technological University Dublin

31st March 2023

Abstract

AppliancesDirect.co.uk carried out a study on the number of times a British person opens the fridge and found that on average, British people look in the fridge or food cupboard 23 times per day. People look because they need inspiration, thus a web application that puts together recipes with the ingredients in the fridge or cupboard would reduce the time spent staring at the food in the fridge.

The aim of this project is to investigate, develop, test and document a recipe search engine in which users can use what is readily available to them and to reduce food wastage in the home. ClearMyFridge is a web application that reduces the time spent pondering what to cook by allowing users to search recipes with ingredients already in the fridge. Users can add ingredients by name or barcode and search recipes based on the ingredients entered.

This project uses Django web framework for backend features and React for a frontend user interface. This web application is hosted on the cloud using Heroku and uses a MySQL database to store data. ClearMyFridge uses Edamam Food Database API and FatSecret Platform API for search food items and recipes. The program uses image processing techniques to detect objects on the screen and to scan the barcode.

Declaration

I hereby declare that the work described in this dissertation is, except where otherwise stated, entirely my own work and has not been submitted as an exercise for a degree at this or any other university.

Signed:

Catriona Renaghan

_____

Catriona Renaghan C19332586

31/03/2023

Acknowledgements

Thank you to my project supervisor Bojan Bozic for guidance, support and advice throughout the development of the project. The supervision and feedback kept this project grounded.

Thank you to Michelle for taking the time to complete the UAT testing for this project.

Table of Contents

Table of Figures

1. Introduction

1.1 Project Background

Throughout the covid 19 pandemic, cooking became increasingly popular, however as the world started opening up again the time people had to cook reduced and meal ideas became more difficult. That is when the thought of having a web application that makes food decisions easier came into play.

Before researching technologies that could be used to develop this application, barcodes and APIs were researched. This gave the developer a background insight into how the application could work. Barcodes are used to track trade items. Barcodes consist of black bars and white spaces above a 12 digit number. The digit number is based on the manufacturer, item variable weight, pharmaceutical drugs, loyalty cards or vouchers. Barcode scanners read the barcodes and reads the black and white spaces and converts them into digits that are sent to a computer program that figures out the product. [1]

APIs were originally created for the business on the web, as businesses could then sell their products across multiple websites instead of one. APIs are used for many different reasons, for example integration between systems, adding functionality to systems and for customers, and for reducing development costs and time. APIs will allow this project to connect the frontend and backend and to connect to Food and Recipe databases. [2]

1.2 Project Description

ClearMyFridge is a web application in which users can enter in ingredients they wish to cook with and on the click of a button, recipes with the inputted ingredients are displayed. When a user first visits the web application home page, they can enter ingredients by typing in the search bar or clicking the option to open the camera and scan the barcode. When the user enters an ingredient in the search bar, responses from a food API are displayed, making selecting items quick and easy. When the user opts for scanning the barcode, the users camera will open and when the user clicks the capture button, an image will be taken and the barcode will be scanned. The user can alternate between typing the ingredient in and scanning the barcode until all the desired ingredients have been entered.

Once the ingredients list has been made, the user can view recipes with the desired ingredients. The user can add or remove ingredients at any stage. Recipe name, description and an image can be viewed at a quick glance. Once the user selects the recipes, the ingredients with quantities and instructions are displayed.

The user can login and save recipes, allowing quick access to the users favourite recipes. When viewing a recipe a heart icon will appear when the user has logged in, otherwise does not appear. A new page with all the saved recipes appear when the user has logged in. The user can remove a saved recipe from the saved recipe page and when viewing the recipe.

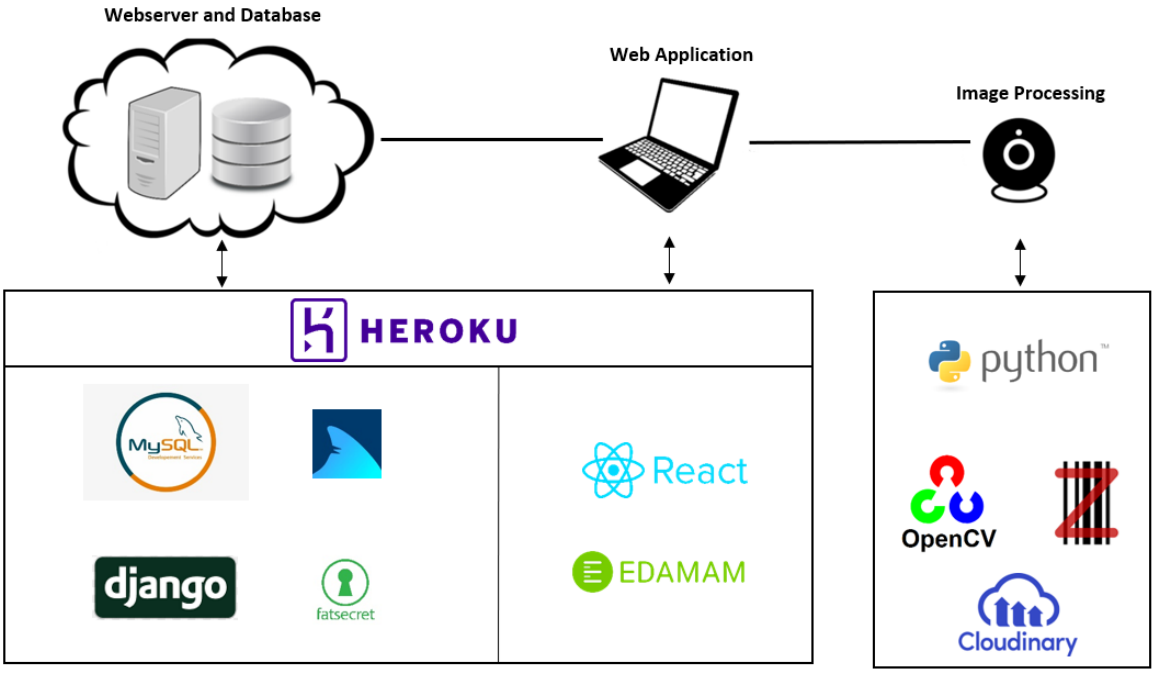


Figure 1 System Architecture

The system consists of three components: the web application, camera and a webserver. The backend and frontend system is deployed in the cloud using Heroku, that provides the functionality to interact with Edamam and FatSecret APIs. The database is stored in the cloud also, which stores user and recipe data.

A web application is the front-end for the system that allows users to enter ingredients, search recipes and save recipes. Through the web application, the user can open the scanning component of the system. This component scans and extracts the barcodes of food products.

1.3 Project Aims and Objectives

The overall aim of this project is to develop a web application that helps users decide on meals to eat. The goals for this project are:

- Implement a fully functional web application that allows users to enter ingredients.
- Design and develop an image scanning program to scan barcodes and search food API with the extracted barcode numbers.
- Implement a search box, to search food products in a food API with ingredient name.
- Deploy a backend using Heroku cloud service.
- Carry out testing on the program.
- Complete a Report on the project.

1.4 Project Scope

Within this project there is frontend and backend development.

1.4.1 Frontend

The frontend in this application is a web application. A web application allows the program to be available to anyone with access to the internet. This allows the application to reach a vast audience. The front end will allow users to enter relevant data and display information.

Search for an ingredient

Upon visiting the web application, the user can search for an ingredient to add to an ingredient list. Users will have two options on how to search for an ingredient; search by text or search by barcode scanning. The user can enter the ingredient name in a search bar. Alternatively, the user can open the camera and search an ingredient using the barcode on the packaging. The user can also remove an ingredient from the list. The objective is to make adding an ingredient as easy as possible.

Search Recipes

After a user has added all the ingredients they wish to cook with, the user can search for recipes. A list of recipes with the ingredients the user inputted will then be displayed. The user can then select a recipe and view the recipe details.

Login and Register

Users do not need to login to use the website, however, logging in provides additional functionality. If the user does not have an account, they can register an account.

Save Recipes

Provided the user has logged in, when the user views a recipes, they can save the recipe. When the user navigates to the recipes page, all the recipes they saved will appear.

1.4.2 Backend

The back-end of this application will maintain database entries and API connections. The back end will search for ingredients, add user and recipe details.

Search for Ingredients

When a user searches for an ingredient by entering by ingredient name, a query will be made to a Food Database API to validate that the ingredient exists. The ingredient will then be added to an ingredients table in the database.

Search Recipes

When a user searches recipes, a query will be made to retrieve all the ingredients from the database. With the ingredients retrieved, a query is made to the Recipe Search API. Recipe data will be returned and stored in the database.

1.5 Thesis Roadmap

This document consists of a number of different chapters, each focusing on a different topic.

Literature Review discusses the research carried out on different technologies and topics that are needed for the completion of this project. Alternative existing solutions and final year projects are examined and documented in this section also.

System Design section discusses design methodologies that are used when developing this system along with UML diagrams on how the system works.

Architecture and Development discusses the architecture of the web application in detail and the development of the systems backend and frontend. This section also discusses the Image processing implementation.

Testing and Evaluation section describes the functional and non-functional ways in which this project is tested during development and discusses the evaluation of the project upon completion.

Conclusions and Future Work section discusses the learning outcomes from the completion of this project, along with any future work that could be added. Limitations to the project are also described in this section.

2. Literature Review

2.1 Overview

When starting a project it is important to research existing solutions to the problem the project is going to fix and technologies that can be used to develop the project. Within this chapter, the following topics are discussed:

- Alternative Existing Solutions
- Technologies
    - Web Frameworks
    - Cloud Hosting Platforms
    - Databases
- Existing Final Year Projects
- Image Processing
- APIs

2.2 Alternative Existing Solutions

Whilst researching it was discovered that Tesco have implemented a similar idea to this project. Tesco allows users to enter the ingredients they wish to cook with and then returns recipes with the ingredients listed. Tesco software returns suggested recipes with additional ingredients other than those provided by the user. The user can select one food item to be a must have in the recipe but only the one. The user can only enter the ingredients in manually. Tesco recipe finder uses the web framework Microsoft ASP.NET to develop their web application. [3]

Another recipe finder website found is myfridgefood.com. This website asks the user to select from an existing list of ingredients and after selecting ingredients, recipes are shown. There is also video recipes available. Myfridgefood like Tesco recipe finder, displays recipes that include other ingredients also. Myfridgefood also uses Microsoft ASP.NET as their web framework. [4]

SuperCook is a recipe search engine that allows users to enter the ingredients they wish to cook with. As the user enters an ingredient, recipes are returned containing the ingredients and new recipes are added with each ingredient. The user can also filter recipes with other

categories such as cuisine, ratings and meal type. SuperCook displays recipes with additional ingredients also, however, within the recipe preview, it specifies if the user has all the ingredients needed. [5]

2.3 Technologies

2.3.1 Web Frameworks

A web framework is a set of resources and tools used by developers to build and manage web applications. They give projects structure and a foundation to start development. Some benefits of using web frameworks is security, scalability, maintenance and reduces code.

Within this section, Django, Flask, Ruby on Rails and React.js are discussed and explored.

Django

Django is a free open source RESTful web framework that uses Python to develop web applications. RESTful frameworks use HTTP requests, such as GET and POST, to access and use data. It was built by developers to encourage fast development and tidy code. Django encourages the reusability of code by removing duplicate code, therefore making the project maintainable. [6] To help aid the fast development of web applications, Django handles user accounts, permissions and cookies.

Django is a secure web framework as it protects applications from clickjacking, cross-site scripting, SQL injection and cross site request forgery. [7] When creating a website with Django, the project consists of one or more applications. This makes development much easier as different features of the website can be developed separately but also allows for scalability.

Web applications such as Spotify, Pinterest and National Geographic use Django web framework.

Flask

Like Django, Flask is a web framework that uses Phyton. Flask is considered a microframework as the applications core is designed to be simple and scalable. Due to this, web applications can be developed quickly and can always scale up due to its simplicity. The microframework is easy to learn, allowing developers to customise and have more control of their code. [8]

Flask needs extensions for some commonly used features such as connecting to databases and creating REST APIs. When connecting a web application to a database an extension is needed due to the lack of an Object Relational Manager (ORM). ORMs act as a middle layer between databases and a program. SQLAlchemy is an ORM and toolkit that can be used within Flask applications to connect to databases. [9]

Web applications such as Samsung, Netflix and Uber use Flask web framework.

Ruby on Rails

Ruby on Rails is a web framework that is written in Ruby. Developers enjoy using Rails because of its extensive libraries for repetitive tasks. Rails combines Ruby, JavaScript, HTML and CSS. It is a free and open-source framework which in return is cost-effective. It is also easy to develop allowing for productivity. [10]

Web applications such as GitHub, Twitch and Hulu use Ruby on Rails.

React.js

React.js is not a web framework but a JavaScript library for building user interfaces. React was built by Facebook to make development of user interfaces easier. React is simple to learn and fast to develop. When a developer creates elements in a program, those elements can be reused elsewhere in the project.

When combing React with Django, it creates a project with management for both frontend and backend as the code for each are written separately. This makes testing and debugging easier with an overall reduced project loading time. [11]

Web applications such as Facebook, Instagram and New York Times use React.js.

Django Web framework is used for the backend of this project. Django was chosen over Flask and Ruby on Rails as it is a python library that has a clear structure to its files. Image processing, discussed later in this chapter, is easily integrated into the Django project. Along with Django, React.js is used for the frontend. React.js makes the user interface quick and easy to develop over using html and css files in the Django Project alone.

2.3.2 Cloud Hosting Platforms

According to Springer, cloud computing enables "flexible, ubiquitous, on-demand and cost-effective access to a wide pool of shared resources". Using cloud hosting platforms reduces businesses costs as they do not have to purchase hardware infrastructures or software licenses but also they can focus on the service or product being sold. [12]

It is important when choosing a cloud hosting platform to research security, reliability, performance and support. In this section, Digital Ocean, Amazon Web Services and Google Cloud are discussed.

By using cloud computing, this project benefits in the following areas:

- Costs: Pay-as-you-go cloud services like Digital Ocean, AWS and Google Cloud, ensure customers only pay for what they need and use. Therefore this project does not waste money and time on services that are not used.
- Scalability: Customers can scale resources as they grow or when they need it. This means customers do not have to plan for the future and can focus on now, allowing this project to grow in the future.
- Agility: Customers can deploy websites within minutes as the cloud platform does most of the work for them. If storage, databases , machine learning is needed, they can be created quickly. The developer for this project can focus on the code and functionality and then deploy.

Digital Ocean

Digital Ocean in an Infrastructure as a Service (IaaS) and Platform as a Service (PaaS). IaaS means Digital Ocean provides customers with infrastructure services through the cloud. IaaS Digital Ocean users can customize and configure disk space, network settings and operating systems to suit their application. PaaS means Digital Ocean provides all backend infrastructures, allowing developers to only focus on code. PaaS Digital Ocean services is aimed at developers who want to launch applications quickly.

Digital Ocean provides support when setting up a cloud based application. They give recommendations on how to get the best performance from different features of an application for example, load balancers, spaces (buckets) and droplets (VM). Digital Ocean

also provides customers with tutorials on how to set up different features and aspects of an application.

Digital Ocean supports MongoDB, MySQL, Redis and PostgreSQL databases, therefore is suited for each of the databases discussed later.

Websites such as Teraone, myCast and Able Few use Digital Ocean as a Cloud Hosting Platform. [13]

Amazon Web Services

Like Digital Ocean, Amazon Web Services (AWS) is a cloud platform that provides cloud computing solutions. AWS is a pay-as-you-go service, customers only pay for what they use. AWS offers a wide range of cloud services, such as storage, databases machine learning, artificial intelligence, security and many more.

AWS is an Infrastructure as a Service however customers also have the option to purchase AWS Elastic Beanstalk which is a Platform as a Service. Elastic Beanstalk deploys and scales web applications for customers. It automatically handles load balancing, scaling, updates and much more. This allows customers to focus on their business and services.

Like Digital Ocean, AWS provides tutorials and instructions on how to set up applications within the cloud. These tutorials guide the user on how to get the best performance out of each component.

Websites such as Facebook, Netflix and LinkedIn use AWS. [14]

Google Cloud

Google Cloud is a cloud hosting platform that lets customers build, deploy and scale applications. Customers of Google Cloud have access to storage, databases, data analytics, networking and more. Like Digital Ocean and AWS, Google Cloud customers only pay for the services they use.

Google Cloud offers Infrastructure as a Service in which Google maintains backend infrastructure and customers manage the rest. Compute Engine services let customers create

and run virtual machines on Google's infrastructure. Google Cloud also offers Platform as a Service in which they manage and maintain backend infrastructure and the software features and tools. Cloud Run services allow customers to build and deploy web applications on a fully managed platform.

Websites such as Twitter, Sky and PayPal use Google Cloud. [15]

Heroku

Heroku is a container based platform as a service that allows customers to deploy, manage and scale applications. Heroku handles load balancing, logging, security, configuration and more making deploying a simple process. Heroku automatically creates a PostgreSQL database for customer applications but also other pre-integrated services such as MongoDB and JawsDB can be used to enhance and manage applications. Heroku provides a ready-to-use environment that allows developers to deploy their code quickly. As Heroku is fully managed, developers can focus on the development of the application and allow Heroku to take care of maintaining servers, hardware and infrastructure. [16]

Companies such as Accenture, Salesforce and Unsplash use Heroku.

Initially, the chosen cloud hosting platform for this project was Digital Ocean. The reasoning behind choosing Digital Ocean over AWS, and Google Cloud is for its PaaS services. There is an expansive document library and Digital Ocean looks after load balancers and buckets allowing the developer to focus of the web application features and functionality. However, after time spent using Digital Ocean and not succeeding, the developer resorted to using Heroku to host the web application. Heroku allows for quick deployment, that can be integrated with GitHub ensuring few changes are needed to the code in order for hosting to work as expected.

2.3.3 Databases

When creating a web application, a database is necessary for storing relevant and useful data. Data can include user information and recipes. Within this section, three different databases are explored.

MySQL

MySQL is a free open source relational database that can be used for small and large applications. A relational database stores data in multiple tables rather than one big table. Tables consist of rows and columns, each containing a primary key to make relationships between tables.

MySQL is simple to learn and use and contains a variety of data structures such as logical, numerical and alphanumeric data but also JSON data. This makes the development of a database easy as calculations can be made for you. [17]

MySQL is a fast and reliable database as it was developed many years ago and has a lot of support online. When developing a database with MySQL, scalability is not a problem. MySQL can be scaled by replication, clustering and/or sharding, however, speed will have to be accounted for. MySQL offers security to protect data integrity, such as Secure Sockets Layer protocol, data masking and authentication plugins. MySQL Enterprise package also includes firewall protection against cyberattacks. [18]

Companies such as Airbnb, Pinterest and Slack use MySQL.

PostgreSQL

PostgreSQL is a free open source object relational database system. An object relational database stores components within its schema and query language. Like MySQL, PostgreSQL is suitable for both small and large applications.

PostgreSQL databases support many different data types, for example; primitives, structured, document and geometry. It is a reliable database as it writes-ahead logging which is used for crash and transaction recovery. PostgreSQL offers different security options such as multi-factor authentication, and access control systems. [19]

Companies such as Instagram, Apple and Spotify use PostgreSQL.

MongoDB

MongoDB is a document-oriented NoSQL database for large applications. Unlike MySQL and PostgreSQL, MongoDB does not use tables, instead it uses collections and documents that can be retrieved in a JSON format.

As MongoDB is document-oriented and stores the data in documents it makes MongoDB adaptable to business situation and requirements. MongoDB is a scalable database, both vertically and horizontally. MongoDB is easy and quick to set up and is suitable for rapid development. MongoDB is best suited for larger applications. [20]

Companies such as Forbes, KPMG and Toyota Material Handling use MongoDB.

MySQL, PostgreSQL and MongoDB are supported by many different cloud hosting platforms, including AWS, Google Cloud and Digital Ocean that were previously discussed. The three different databases can also be used with the web frameworks; Django, Flask and Ruby on Rails, discussed above. MySQL is used for this project due to its ease of use and how it is a relational database. Querying data is a simple process and when integrated with Django, creating and removing data is fast. It requires little setup compared to PostgreSQL and MongoDB.

2.4 Image Processing

2.4.1 Pillow

Python Pillow is an image processing library that is a fork of the Python Imaging Library (PIL). According to Real Python [21], PIL was discontinued in 2011, therefore Pillow was forked to be the library active. Pillow can be used to crop and resize images but also to change the content of an image, however, Pillow does not support video.

2.4.2 OpenCV

"OpenCV is an open source computer vision and machine learning software library". [22] In other words, OpenCV is a free library for python, java and C++, that works with images and videos. Programs can use OpenCV algorithms to detect objects and faces, track camera movements, extract objects and improve image and video quality. OpenCV is the bases of any image editor.

OpenCV is used by Google, Microsoft and Intel.

Image Cleaning and Enhancing

In order to work with images, images must be clean and clear. Blurring, sharpening, thresholding and contrast enhancing are techniques used within image processing to clean and enhance images.

Image blurring is used to reduce noise within an image, by removing high frequency content. OpenCV has four types of blurring techniques; Averaging, Gaussian Blurring, Median Blurring and Bilateral Filtering. Averaging consists of splitting an image into box filters and takes the average pixel value and replaces the centre pixel with this value. Gaussian Blurring uses a Gaussian kernel instead of box filters which removes gaussian noise from an image. Median Blurring uses kernels and takes the median of the pixel values and replaces the centre pixel in the kernel with median value. Finally Bilateral Filtering uses the Gaussian function to determine if pixels have similar intensities to the central pixel and then only those pixels are used. [23]

OpenCV uses kernels to sharpen an image. Different kernels can be used depending on how intense the user wants the image to be.

Contrast can be enhanced using equalisation and histograms. By improving the contrast in an image, the intensity range is stretched, intensifying the difference in the colours. This can be very effective on grayscale images. [24]

Thresholding in OpenCV is used to create binary images. This technique is used to separate an object from background pixels, a form of segmentation. Simple thresholding involves assigning any pixel value within an image that is greater than the specifies threshold value a standard value. Adaptive thresholding involves having a different threshold value for different regions in an image. [25] Thresholding can be seen in Figure 2, the original image has shadows across the image, after thresholding the numbers are clear and there is no shadows.

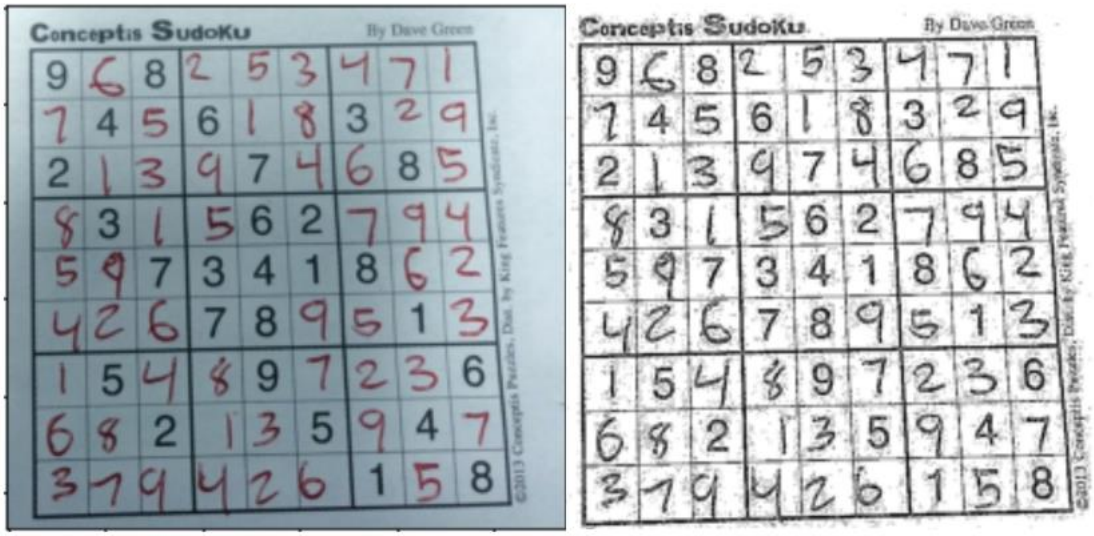


Figure 2 Thresholding

Morphology within image processing is transforming images based on shapes. Morphological operations transforms each pixel in an image to a value based on its neighbouring cells. There is four main methods of morphology; Erosion, Dilation, Opening and Closing. Erosion involves eroding the outer surface resulting in smoother boundaries around the object and eliminating blobs (a group of pixels). Dilation involved dilating the foreground of an image to also smooth boundaries and fill gaps. Opening involves erosion followed by dilation which is useful for removing unwanted blobs for example specs. Closing involves dilation followed by erosion to fill small holes. These morphology transformations or combined help reduce noise and distortions that can occur when adding a threshold to an image.

2.4.3 Pyzbar

Pyzbar is a python library used to read barcodes and QR codes using zbar library. Zbar is an open source software suite for reading bar codes. [26] Zbar supports many different barcode types such as EAN-13/UPC-A and QR code. After locating the barcode within an image, pyzbar decode is used to extract the barcode numbers which can then be used to search an ingredient in the Edamam Food API.

2.4.4 Cloudinary

Cloudinary is a cloud bases Software-as-a-Service image and video management solution for web applications that "store, transform, optimize and deliver all your media assets with easy-to-use APIs, widgets, or user interface". [27] Cloudinary offers HTTP and URL based API integration and an interface allows for full control over and management of media assets.

Within this project images are uploaded to a Cloudinary API for barcode scanning, allowing the images to be uploaded within the frontend and then accessed in the backend for processing.

2.5 Application Programming Interface

An Application Programming Interface (API) is code that enables data transmission between two software products. The API accepts data from users, forwards and processes the data in the application and returns the result to the user. Within this project an API is used to process the ingredients entered by the user and to retrieve recipes based on the ingredients entered.

2.5.1 Edamam

Edamam is a REST based API service for food and grocery, nutrition analysis and recipe search. Edamam provides access to a food and grocery database with over 500,000 food, restaurant items and packaged foods. The Food API can be searched using food product names and barcode numbers. The API returns data for nutrients values for each food product. Edamam also has a Recipe Search API that contains over 2.3 million recipes. When searching recipes, images, ingredients and nutritional information is returned. [28]

2.5.2 FatSecret Platform API

FatSecret Platform API is a REST based API service for food and nutrition. "The REST API can be used to build nutrition, diet and weight management solutions on any platform". FatSecret provides access to over 1.5 million food items and more than 14,000 recipes. All recipes include images, cooking directions, ingredients and nutritional information. [29]

Edamam Food API and FatSecret Platform API are used in this project to search ingredient product names, search the barcode numbers and to return recipes based on the ingredients entered by the user. Edamam Food API is used to search ingredient products by name and barcode and to validate that the product the user enters exists. Edamam Recipe Search API does not return recipe directions, therefore FatSecret Platform API is used to search for recipes containing the ingredients inputted by the user.

2.6 Existing TU Dublin Final Year Projects

2.6.1 Event Management System Using Image Authentication

The event management system by Eamon Tang is a mobile application that uses QR code scanning to authenticate an event attendee. A camera scans each attendees QR code which they receive through an Android app and the user is then authenticated into the event. The user can also create and manage events through the app.

The project uses OpenCV to detect and extract the mobile screen from an image. The image is firstly converted to grayscale and then edge detection is used to detect edges within the image, displaying the edge of the mobile phone. Contours are found using OpenCV functions and the screen and QR code are located. The time and date in the image is read using Tesseract and the QR code is read using ZBar.

The program uses Digital Ocean as its cloud hosting platform, PostgreSQL for its database and Django as its web framework. [30]

2.6.2 FindMyImage – Estimating the location of an Image

The image search engine by Ciaran Corson allows users to enter an image into the program and it returns an estimation of the image location based on other images at the same location.

FindMyImage uses a Flickr API to compare the uploaded images with other similar images. The project calls getInfo to an image which receives details about the returned image. The project also uses OpenCV functionality such as SIFT to find key points in an image, homographs to detect likeness between two images and histogram calculation and comparison.

FindMyImage uses a simple file storage just in two folders and an Apache web server. [31]

2.7 Conclusion

After researching existing solutions, technologies and image processing libraries the following conclusions were made.

This project uses Django to develop the web application. Django is used for backend work as it allows for the project to be scalable and secure whilst allowing for fast development. Django allows time to be sent on functionality aspects rather than the setup of the web application.

This project also benefits from using React.js for front-end development, as JavaScript has libraries and features to implement the project features. React.js makes development of user interface fast and simple and combing React with Django shortens web application development time.

Heroku is used to host the web application. Heroku makes launching web applications simple and quick and it provides backend infrastructure, taking the maintenance and configuration off the developers hands. It also supports all the different technologies that are used within this project.

MySQL is used to store data for this project. MySQL is easy to set up, fast and reliable. Querying data is a simple process within Django, allowing focus to be elsewhere in the project.

OpenCV image processing library is used to read the users camera input, locate the barcode and read the numbers. Edamam Food API then processes the barcode numbers and returns the ingredient product. The ingredient products is used with FatSecret Platform API to return desired recipes.

3. System Design

3.1 Overview

When developing a program it is important to follow a software methodology so that the project is structured, planned and the development process is controlled. In this chapter, four different software methodologies are discussed; Waterfall, Scrum, Rapid and Agile Development methods.

Before development it is important to create a system overview to understand how the system architecture interacts with each other. Creation of UML diagrams is important to understand the system flow and to further understand the system making development easier.

3.2 Software Methodology

3.2.1 Waterfall Development

Waterfall method is considered a traditional software development method. It consists of sequential phases as seen in Figure 3. Each phase has distinct goals that must be entirely completed before moving onto the next phase. As this methodology is linear, it makes it easy to understand and use and is best used for projects with clear objectives and requirements. However since this method is linear, it makes development slow and costly as developers cannot move onto the next phase without completion of current phase. [32]

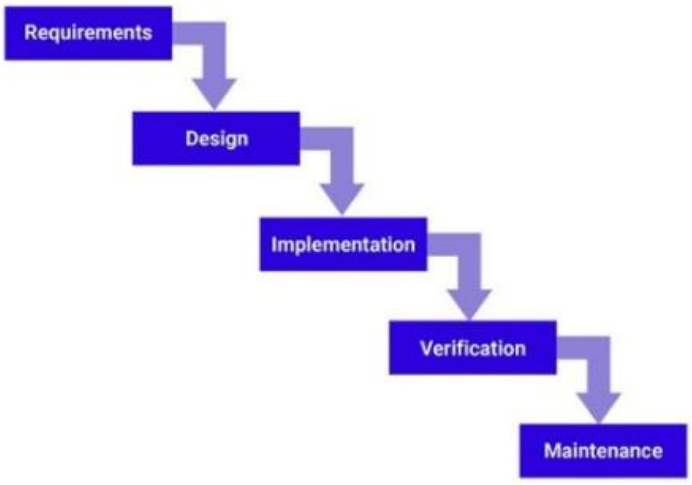


Figure 3 Waterfall Methodology

3.2.2 Scrum Development

Scrum method follows agile development to simplify the development process. The development team breakdown the requirements into goals and then break the goals into smaller goals. These smaller goals are then placed into sprints accordingly and developed within the allocated sprint time period. When changes are made to the requirements, it is quick and easy to adjust the goals to suit the changes. As there is no clear end date, it can be difficult to get frequent project updates. [33]

3.2.3 Rapid Application Development

Rapid application development method is a condensed development process that allows developers to adjust requirements quickly. It consists of four stages, as seen in Figure 4; requirements planning, user design, construction and cutover. User design and construction phases are repeated until the product meets all requirements. This methodology is effective for small or medium sized projects that need to be developed quickly, however, a skilled team is required. [32]

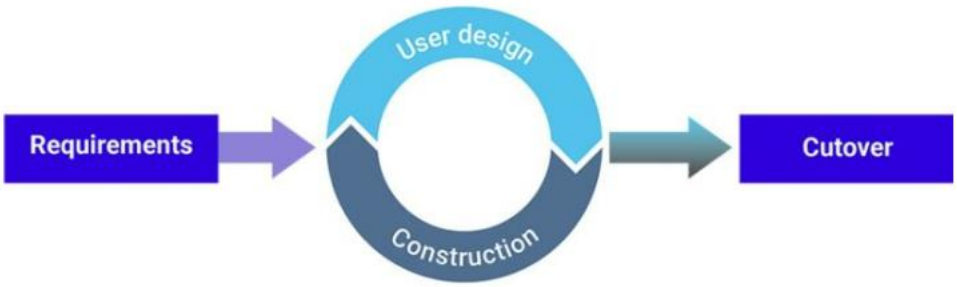


Figure 4 Rapid Application Development

3.2.4 Agile Development

Agile method consists of developing in short sprints that contain new functionality. Sprints, as seen in Figure 5, consist of planning, designing, development, testing deploying, reviewing and finally launching. At the end of each sprint, the aim is to have working software. This approach to development allows for the software to be released in iterations. Bugs and errors can be detected and fixed early on in development. Agile Development requires team

collaboration to ensure requirements are met for each sprint and extra consideration needs to be taken into account for documentation. [32]

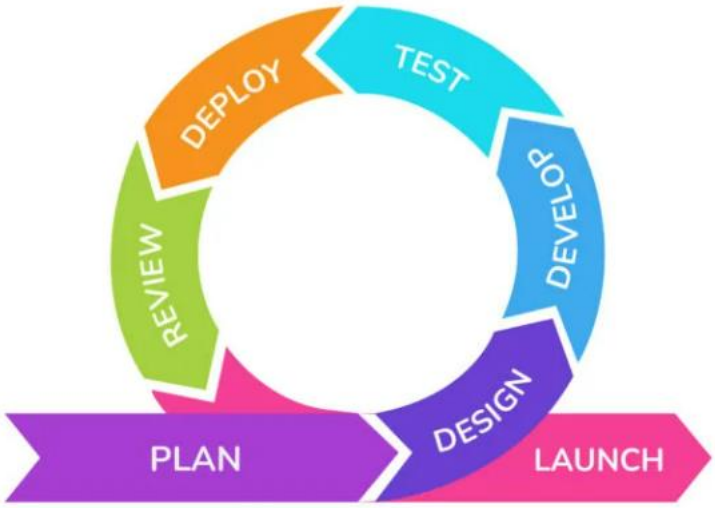


Figure 5 Agile Development

This project uses the Agile Development method for planning and developing. Following the agile methodology improves the planning phase of the project as requirements can change each sprint. When new ideas are thought of, they can easily be added to the requirements and can be developed in another sprint.

Testing is performed during the sprint, allowing faults to be detected after development and they can be fixed before the sprint is complete. When testing is performed each sprint, requirements can then change based on user feedback.

After each sprint, all new code and functionality is complete, therefore, there is always a working product. This allows the developer to focus on the task at hand, rather than trying to complete multiple requirements at once. The system is also built so that features can always be added, allowing the program to be scalable. [34]

3.3 Overview of System

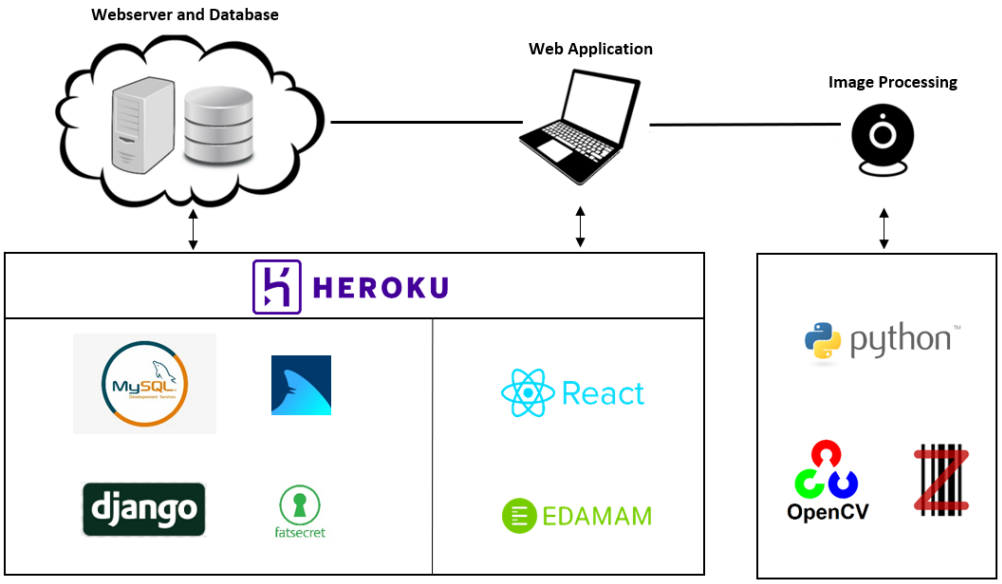


Figure 6 Overview of System Architecture

Figure 6 is an overview of the system architecture that is used for this project. The web application is hosted in the cloud using Heroku. The web application consists of a Django and React project connected to a MySQL database. Within this project, two connections are made to online APIs. One connection is made to Edamam Food Database API and another to FatSecret Platform API. The web application can then be used on a user's device which has a connected camera. The camera is used to capture an image for Image Processing with python and OpenCV to extract a barcode.

3.4 Unified Modelling Language UML

UML diagrams are a key aspect when planning software development. UML diagrams construct, document and visualise software systems. There is two types of UML diagrams; Structural and Behavioural. Structural Diagrams show systems structure, levels and individual parts. Behavioural Diagrams show how the system should function.

Creating UML diagrams provide the developer a greater understanding of the system. The developer can potentially detect any challenges that could arise or potential problems in the design. UML diagram also unifies design functionality and flow. [35]

3.4.1 Activity Diagram

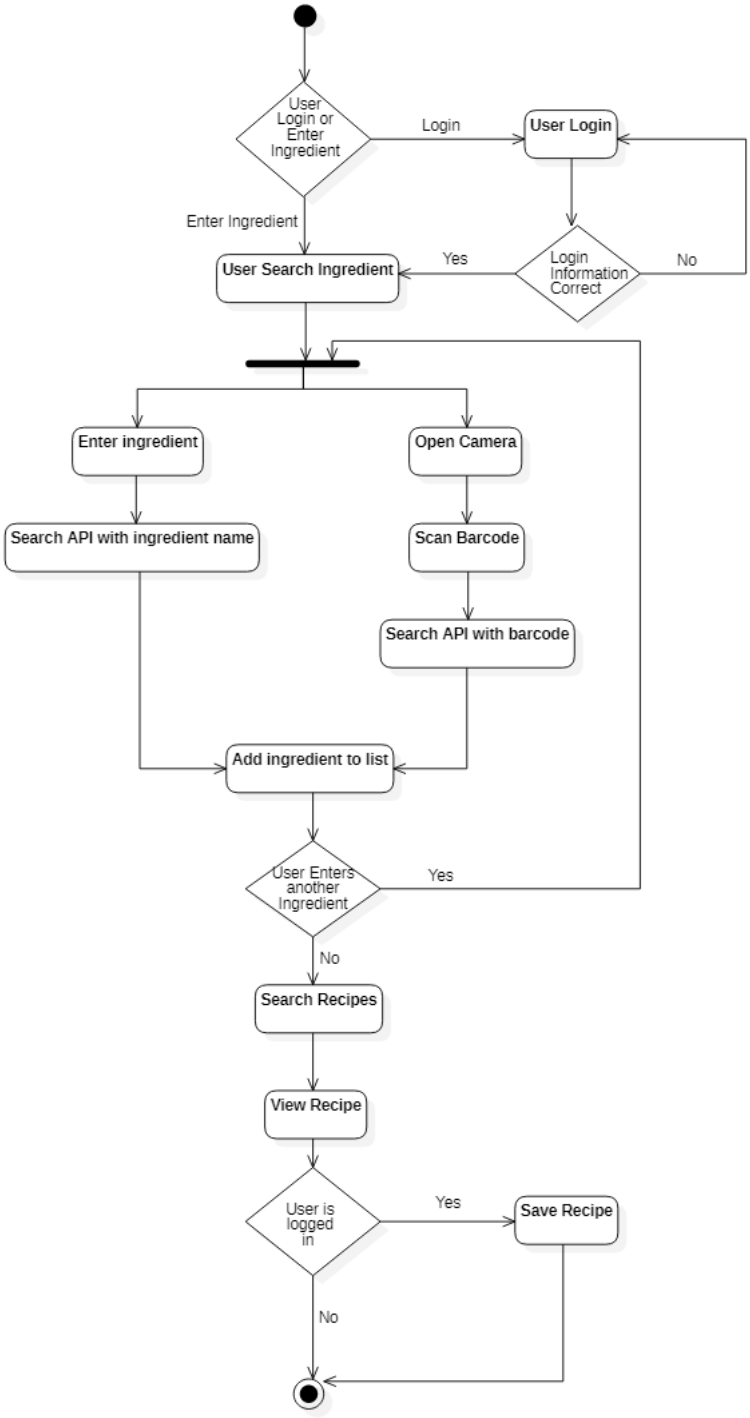


Figure 7 Activity Diagram

An Activity Diagram is a behavioural UML diagram that depicts the flow of the system. Figure 7 illustrates the work flow of this program. When the user starts the program, they can login upon starting or they can enter an ingredient. If the user decides to login, the user logins and

the login information is checked. When the login details are correct the user can continue to search an ingredient, if the login details are incorrect they are brought back to the login page.

The user does not have to be logged in when searching an ingredient. The user has two choices, enter ingredient or open camera. When the user choices to enter the ingredient, the ingredient name is used to search the Food API. When the user opts to open the camera, the barcode is scanned and the barcode is then used to search the Food API. For both cases the returning ingredient is added to an ingredient list.

The user can continue to add ingredients to the list for they can continue to search recipes. Next they can view recipes. If the user has logged in, the user can save the recipe, if the user has not logged in, the program finishes.

3.4.2 Use Cases

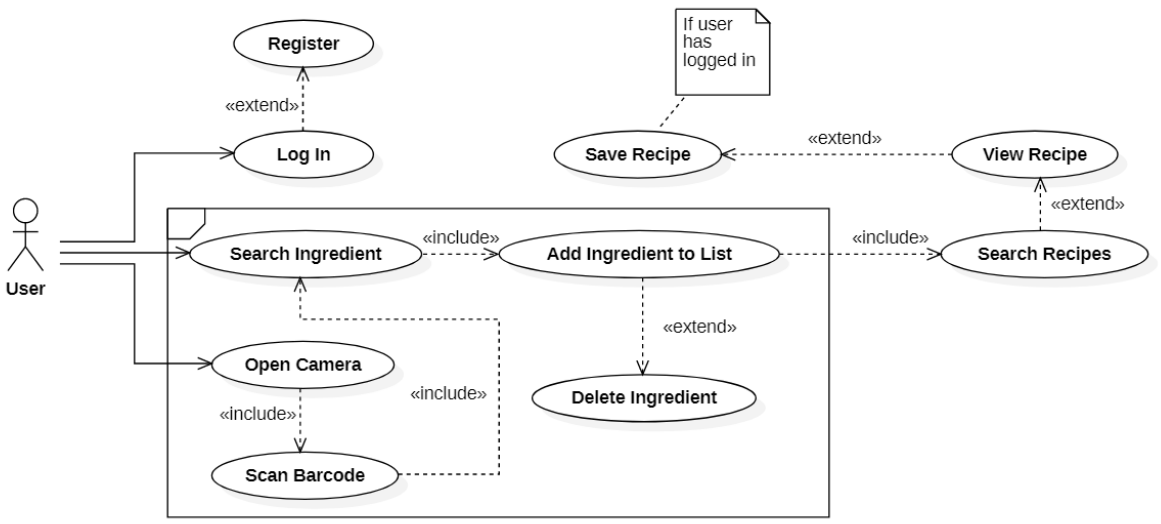


Figure 8 Use-case Diagram

Use case diagrams are behavioural UML diagrams that show the parts and functionality of the system and how they relate to each other. Figure 8 is a use case diagram of the interactions between the user and the system. A user has to the option to login, search ingredient or open camera. If the user opts to search ingredients, the user searches by ingredient name and the ingredient is added to the ingredient list that will be used to search recipes. The user also has the option to remove ingredients from the list at this stage. When the user opts for opening

the camera, the user can scan the barcode and the ingredient is searched using the barcode number. The search ingredient process is repeated until the user has entered the desired amount of ingredients.

The recipes are then searched based on the ingredient list. After the recipes have been searched, the user can view the recipes. If the user has logged in, the recipe can be saved. The user can register if they do not have login details.

3.4.3 Class Diagram

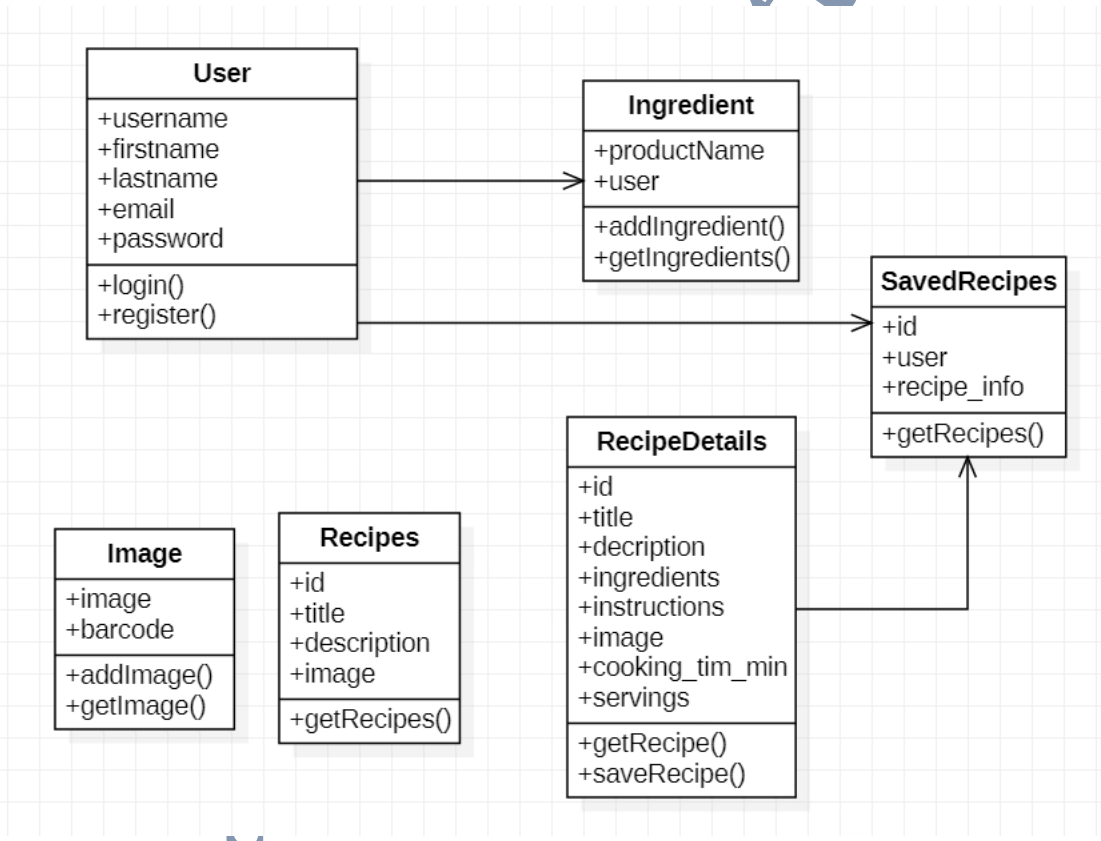


Figure 9 Class Diagram

Class diagrams are structural UML diagrams that depict the structure of the software system. Figure 9 illustrates the different classes and the relationships they share with each other. User class contains the relevant attributes for the user details. A single user can add multiple ingredients.

Ingredients class contains the relevant details that are needed for the ingredient list and recipes. One recipe can have many ingredients. Recipes class contains the attributes needed for displaying recipes to the user.

The recipes and image classes are used to stored data from image processing and FatSecret API responses.

A recipe and user is added to the SavedRecipe class when the user wants to save a recipe.

3.5 Requirements

Requirement ID	Name	Description	Priority
R1	Setup Django project	Create a django project for development to be made on	Medium
R2	Create MySQL database	Create MySQl database for data storage	Medium
R3	Connect to API(s)	Send data to API and receive a response back	Medium
R4	Read user ingredient Input	Store users input from search bar	Medium
R5	Pass ingredient Name to API	Send ingredient Name to API and receive ingredient details	Medium
R6	Add ingredient to list	Add ingredient name to list for recipe search	High
R7	Extract product from Screen	Use OpenCV to detect barcode and take snapshot	High
R8	Read Barcode Numbers	Use OpenCV to detect numbers and read numbers	High
R9	Pass Barcode to API	Pass numbers to API to receive product details	Medium
R10	Add ingredient to list	Add ingredient name to list for recipe search	Low
R11	Search API for recipes with ingredients	Search for recipes based on the ingredient list	High
R12	Display Recipe Details	View the recipes returned from API	Medium
R13	Design	Implement user friendly interface	High
R14	Login	Log user into the system	Low
R15	Register	Create new user	Low
R16	Save Recipes	Save Recipe to users account	Low
R17	Create Django Environment	Create environment on webserver needed for django project and database	High
R18	Communicate with Web Applic	Connect with application from the internet	High

Figure 10 Requirements

As seen in Figure 10, the projects requirements have been divided into phases for the development process. Each requirement has an ID, name, description and priority. The priority is in relation to how the feature effects the system and how complex it is.

Before development the requirements list contained 15 features, as development started, three more were added to account for changes and for focus on an area. Agile Methodology allows for changes to be made along the way.

3.6 Conclusion

System Design is an important part of software development. After researching waterfall, scrum, rapid and agile development methodologies, it was concluded that agile development is best suited for this project. When a new feature is developed, the feature is tested before moving onto the next feature.

After creating UML diagrams, it is clear on how the system will function and flow. The activity diagram and use case diagram describe the flow of the system and how the user is expected to use the system. The class diagram describes the structure of the system. These UML diagrams make the development of new features easier as the developer already knows how each component works with each other and how the final system works.

Once the functionality features and flow was established, the development of a requirements list was simple. These requirements are the basis of what features will be developed in each sprint.

4. Architecture & Development

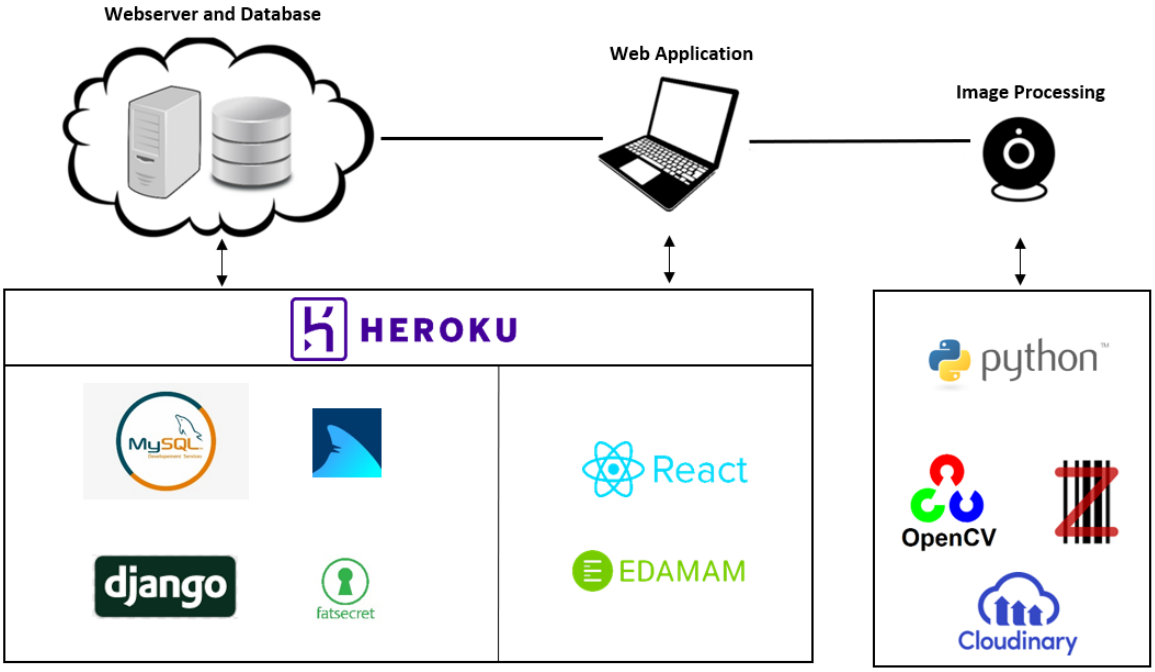
4.1 Overview

Within this chapter the system Architecture is discussed. The design and development of each component within the system is explained.

For the Django backend, each model, serializer and view is discussed in detail. The database is explained and discussed how the backend interacts with it. Hosting the web application within Heroku is explained. The frontend features of the application and the User Interface are demonstrated and discuss how they interact with the backend.

Finally the procedure in scanning a barcode and extracting the barcode numbers is explained in relation to image processing techniques.

4.2 System Architecture



4.2.1 Webserver and Database

The webserver is hosted on a Heroku dyno, along with the Django backend and react build. Gunicorn is a HTTP server that is used to run the python processes within the dyno for this web application. Gunicorn runs the server when the application is deployed and makes the application backend available to the frontend. The server is where Django makes HTTP

requests to FatSecret API for recipes and where image processing techniques are used to extract the product barcodes.

A MySQL database, JawsDB cloud MySQL database, is used to store data for the Django application within the webserver. Django manages the database migrations of models and data.

ClearMyFridge application can be accessed by following the domain: www.clearmyfridge.com.

4.2.2 React Web Application

The React web application was developed using JavaScript. The application communicates with the Django backend on the webserver via HTTP requests through REST APIs. The React application allows the users to search food product names or scan product barcodes and add those products to an ingredient list to then search for recipes with those ingredients.

The React Application was built locally and the build files were added to the Django backend directory. This makes making changes within the application difficult as the application would have to be built and redeployed each time. Heroku dynos have limited space (500MB) and building the application within the webserver increased the space used by over 100MB, causing the application to never deploy as the Django backend uses 492.7MB of space.

4.2.3 Camera Scanning

The barcode scanning component of the architecture consists of a webcam on the users device. The use of the webcam, start, capture and close camera, is completed within the React application. The image is stored in Cloudinary API database and retrieved within the Django backend on the webserver. OpenCV and pyzbar, python image processing libraries, are used to extract the barcode and send a response back to the frontend with the results.

4.3 Django

4.3.1 File Structure

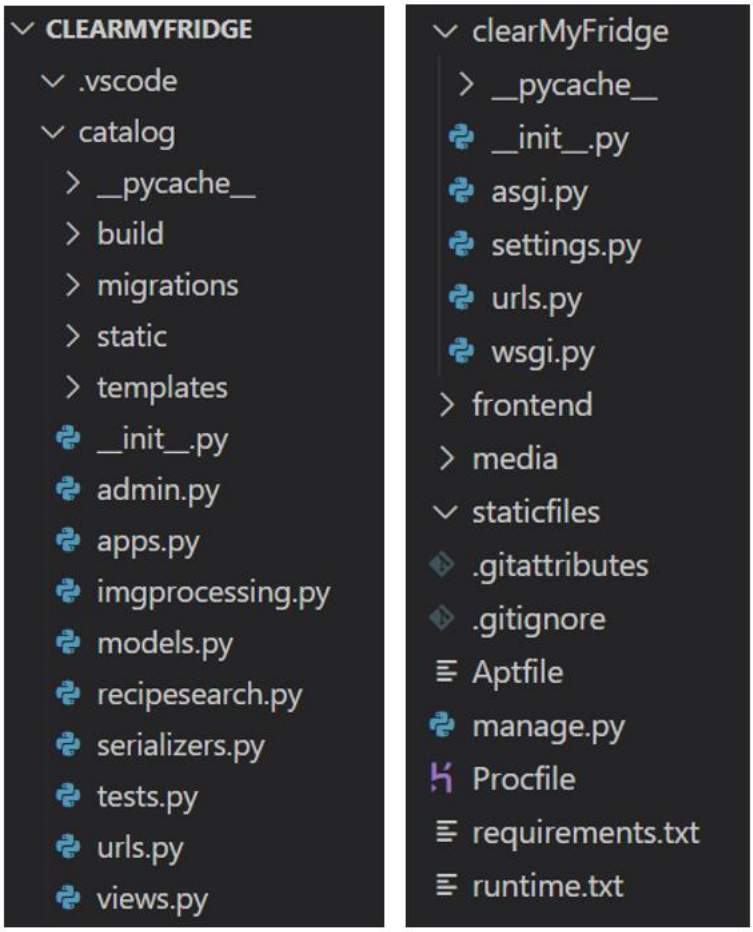


Figure 11 Django Backend File Structure

Django makes use of a directory structure that arranges the application in a way that is manageable. When the Django project, clearMyFridge, was created, a root directory was automatically created with the necessary files needed to run the application.

Manage.py

Manage.py is used to perform admin operations such as migrating data and models and running tests. Makemigrations and migrate commands allows changes to be made to the database. This file also contains code to run a test server which is used to during the development stage.

Urls.py

This file contains the end points for the Django project. When a view is created of a model, a url path is added to allow the developer to view the data that is stored within the database and ensure the system is performing as expected. When the React frontend sends a HTTP request to the Django backend, it connects using the API endpoint URLs.

Database

```
DATABASE_URL = 'mysql://kbsriydmn57dctux:ohp0e765ywd1m

db_config = dj_database_url.parse(DATABASE_URL)

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': db_config['NAME'],
        'USER': db_config['USER'],
        'PASSWORD': db_config['PASSWORD'],
        'HOST': db_config['HOST'],
        'PORT': db_config['PORT'],
    }
```

Figure 12 Database Configuration

Figure 12 shows the database connection to the JawsDB. This connection is configured in the settings.py file. This database is used in production, however during development and whilst running the application locally, a local MySQL database is used. Section 4.4.3 Pages discusses the reasoning to why two different databases are used.

Running the Application

```
web: gunicorn clearMyFridge.wsgi --log-file -
```

Figure 13 Procfile

The Procfile is used by Heroku to specify the commands to be executed when the application starts. The web: prefix indicates to Heroku that the command should run as a web process

and starts the Gunicorn webserver with the clearMyFridge.wsgi files as the entry point for the application.

```
import os

from django.core.wsgi import get_wsgi_application

os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'clearMyFridge.settings')

application = get_wsgi_application()
```

Figure 14 wsgi.py

The wsgi.py file is used to set up the Django application and configure the environment in the webserver. WSGI (Web Server Gateway Interface) is used to handle HTTP requests and responses. Gunicorn uses get_wsgi_application() to run the application using a webserver.

4.3.2 Models

Models define the structure of the database. Each field that is included in the models, is also included in the database. Each field type is defined and a __str__ method returns a representation for the model.

Within this project there is 5 different models: Ingredient, Image, Recipes, RecipeDetails and SavedRecipes. Each model in the database stores relevant details needed for the web application to work as expected.

```
# INGREDIENT MODEL
class Ingredient(models.Model):
    """Model representing an ingredient."""
    productName = models.CharField(max_length=100)
    user = models.CharField(max_length=100, default='user')

    def __str__(self):
        """String for representing the Model object."""
        return self.productName, self.user

# IMAGE MODEL
class Image(models.Model):
    """Model representing user image."""
    title = models.CharField(max_length=100, default='image')
    image = models.URLField(max_length=200)
    barcode = models.CharField(max_length=100, default='barcode')

    def __str__(self):
        """String for representing the Model object."""
        return self.title
```

Figure 15 Model Details - models.py

Ingredient and Image models are displayed in Figure 15 Model Details. The Ingredient model contains two fields, productName and user of which are both character fields. Both the productName and user represent this model. User is set to user as default and is used when the user has logged in, allowing logged in users to save their ingredients. The Image model contains title, image and barcode fields. Title and barcode are character fields whereas image is a URL field that stores the link to a cloudinary image of the users product.

See Appendix 8. Django Models for Recipes, RecipeDetails and SavedRecipe models.

4.3.3 Serializers

Serializers convert objects into data types that can be understood by JavaScript and the frontend. Within this project, ModelSerializers are used to automatically generate the fields within the models. Within this project there is 5 Serializers, one for each model.

```
# INGREDIENT SERIALIZER
class IngredientSerializer(serializers.ModelSerializer):
    class Meta:
        model = Ingredient
        fields = ('id', 'productName', 'user',)

# IMAGE SERIALIZER
class ImageSerializer(serializers.ModelSerializer):
    class Meta:
        model = Image
        fields = ('title', 'image', 'barcode',)
```

Figure 16 Serializers - serializers.py

Figure 16 displays two serializers for the Ingredient and Image models that both inherit the Ingredient and Image fields. All other serializers follow the same structure inheriting the model fields.

See Appendix 8.2 Django Serializers for RecipesSerializer, RecipeDetailsSerializer and SavedRecipeSerializer serializers.

4.3.4 Views

Views in Django are Python functions implement tasks based on HTTP requests and returns HTTP responses. Views contain the logic that is needed to return information to the React Frontend. The following HTTP requests are used within this project:

- GET: retrieve data from a table within the database.
- POST: send data to the database to create or update data.
- PUT: replace all the data within a table with the uploaded data.
- DELETE: remove data within a table in the database.

```python
# INGREDIENT LIST GET/POST
@api_view(['GET', 'POST'])
def ingredient_list(request):
    if request.method == 'GET':
        username = request.GET.get('user')
        if (username == "Guest"):
            ingredients = Ingredient.objects.filter(user="Guest")
        else:
            ingredients = Ingredient.objects.filter(user=username)

        serializer = IngredientSerializer(ingredients, many=True)
        return Response(serializer.data)
    elif request.method == 'POST':
        serializer = IngredientSerializer(data=request.data)
        if serializer.is_valid():
            serializer.save()
            return Response(serializer.data, status=status.HTTP_201_CREATED)
    return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)
```

Figure 17 Ingredient List View - views.py

Figure 17 Ingredient List View displays a GET and POST request to the Ingredient model. When a user adds an ingredient, the request is sent as a POST and serializes the request data that is sent, productName and user. If the data is valid, it is added to the Ingredient table, if the data is invalid, an error Response is sent.

When the user wishes to view the list of ingredients the username is retrieved from the request data and then checked if the user is “Guest”. If this is the case it retrieves all ingredients in the database that has the user “Guest” associated with it. If the username is not “Guest”, all ingredients

```python
# RECIPE DETAIL GET/POST
class recipeDetailView(APIView):
    def get(self, request, *args):
        recipes = RecipeDetails.objects.all()
        serializer = RecipeDetailsSerializer(recipes, many=True)
        return Response(serializer.data)

    def post(self, request, *args):
        recipe = SearchRecipes.searchRecipe(request.data['id'])
        recipes = RecipeDetails.objects.all()
        recipes.delete()
        print('Recipes deleted successfully')

        serializer = RecipeDetailsSerializer(data=recipe)
        if serializer.is_valid():
            serializer.save()
            return Response(serializer.data, status=status.HTTP_201_CREATED)
        return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)
```

Figure 18 Recipe Detail View - views.py

Figure 18 Recipe Detail View displays a simple python method to GET and POST details to the RecipeDetails Model. When the user requests to view a recipes details, the get method is used. This method retrieves all the recipes, objects.all(), stored within the RecipeDetails model. The RecipeDetailsSerializer is then used to serialize the recipes and the data is responded to the frontend using the Response class from the rest_framework.

When the frontend sends a POST request, the post method is used. This method sends the id within the request data to the searchRecipe function in recipesearch.py which is discussed later in this section. The recipes variable gets all the objects within RecipeDetails table and deletes them, ensuring there is only ever one recipe in the table. The recipe details returned from searchRecipe is serialized and if the data is valid it is added to the database and the serializer data and 201 status code is sent in the response. If it is invalid, the error and status code 400 is returned.

```python
# LOGIN POST
class loginView(APIView):
    def post(self, request, *args):
        username = request.data['username']
        password = request.data['password']
        user = authenticate(request, username=username, password=password)
        if user is not None:
            login(request, user)
            return JsonResponse({'success': True})
        else:
            return JsonResponse({'success': False, 'message': 'Invalid login credentials.'})


# REGISTER POST
class registerView(APIView):
    def post(self, request, *args):
        User = get_user_model()
        user = User.objects.create_user(
            username=request.data['username'],
            first_name=request.data['firstName'],
            last_name=request.data['lastName'],
            password=request.data['password'],
            email=request.data['email']
        )
        return JsonResponse({'success': True})
```

Figure 19 Login and Register Views - views.py

Figure 19 displays the Login and Register views used when a user registers an account and logs in. This project uses Django authentication system to register and login a user. Three different methods from the Django.contrib.auth are used:

- Get_user_model(): returns the currently active user model.
- Authenticate(): the user request, username and password are passed into this method. It calls User.check_password that returns true if the password is correct for the user.
- Login(): After the user is authenticated, the HttpRequest object and User object is passed into the login() method. The user's ID is saved in the session, using Django's session framework.

When registering a user the get_user_model() method is called and the user model that is active in the project is returned. An instance of the user model is created using create_user method with the username, firstname, lastname, password and email within the request data. Finally a JSON response is returned to indicate that the user has successfully logged in.

When a user logs in the username and password is retrieved from the request data and then authenticate checks if the username and password are correct. If the user is valid, the login()

method is called to log the user in. A JSON response is then returned with success flag set to true. If the user is not valid, a JSON response is returned with the success flag set the false and a message set to “Invalid login credentials”.

```python
# IMAGE LIST GET/POST
class imageView(APIView):
    def get(self, request, *args):
        images = Image.objects.all()
        serializer = ImageSerializer(images, many=True)
        return Response(serializer.data)

    def post(self, request, *args):
        barcode = ExtractBarcode.extract_barcode_nums(
            request.data['image'], url=True)
        request.data['barcode'] = barcode
        serializer = ImageSerializer(data=request.data)
        if serializer.is_valid():
            serializer.save()
            return Response(serializer.data, status=status.HTTP_201_CREATED)
        return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)
```

Figure 20 Image View - views.py

Figure 20 displays a GET and POST request to the Image model. This view is used to add and extract barcodes from the image uploaded by a user. When a GET request is called, all images within the Image table are retrieved using Image.objects.all(). The data returned is then serialized and returned in a Response.

When a POST request is called the image URL in the request data is passed to extract_barcode_nums function in the ExtarctBarcode class within the imgprocessing.py file. This function is discussed later in section 4.5.2 Extract Barcode from Image. The barcode is returned from this function and the barcode is inserted into the request data. The request data is then passed to the ImageSerializer. If the serializer is valid, it is saved and a positive response is returned. If the serializer is invalid, an error response is returned.

```python
# SAVED RECIPE LIST GET/POST/DELETE
class savedRecipeView(APIView):
    def get(self, request, *args):
        username = request.GET.get('user')
        recipes = SavedRecipes.objects.filter(user=username)
        serializer = SavedRecipeSerializer(recipes, many=True)
        return Response(serializer.data)

    def post(self, request, *args):
        recipeid = request.data['id']
        recipe_info = SearchRecipes.searchRecipe(recipeid)
        request.data['recipe_info'] = recipe_info
        serializer = SavedRecipeSerializer(data=request.data)
        if serializer.is_valid():
            serializer.save()
            return Response(serializer.data, status=status.HTTP_201_CREATED)
        return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)

    def delete(self, request, *args):
        recipe = SavedRecipes.objects.get(id=request.data['id'])
        recipe.delete()
        return Response(status=status.HTTP_204_NO_CONTENT)
```

Figure 21 Saved Recipe View - views.py

Within the savedRecipeView three HTTP Requests can be called; GET, POST and Delete. When the user requests the saved recipes, the recipe table is filtered by the username and the recipes returned are serialized and returned in a response.

When the user saves a recipes the POST HTTP request is sent. Similar to recipeDetailView, the recipe id is retrieved from the request data and passed to the searchRecipe function to retrieve the recipe details which is then inserted into recipe_info in the request data. The request data is then serialized and if valid a the serialized data and a positive response is returned, otherwise a negative response and its error is returned.

See Appendix 8.3 Django Views for ingredient_detail and recipesView.

4.3.5 Recipe Search

Recipesearch.py file contains two functions that communicate with the FatSecret API to retrieve recipes based on the ingredients the user has entered and to retrieve details about a specific recipe. A connection to FatSecret is established using the FatSecret module and passing the API key and secret of the account being used for this project.

```
class SearchRecipes:
    # Search Recipes based on Ingredients
    def search(myIngredients):
        recipes = fs.recipes_search(myIngredients)

        recipe_data = []
        for recipe in recipes:
            recipedata = {}
            recipedata['id'] = recipe['recipe_id']
            recipedata['title'] = recipe['recipe_name']
            recipedata['description'] = recipe['recipe_description']

            if 'recipe_image' in recipe:
                recipedata['image'] = recipe['recipe_image']
            else:
                recipedata['image'] = 'https://res.cloudinary.com/dt3iqlezo/image/upload/v1677598676/Constants/logo_kt4v4g.png'

            recipe_data.append(recipedata)
        return recipe_data
```

Figure 22 Search - recipesearch.py

The search function, figure 22, searches the FatSecret API based on the user ingredients. A list of ingredient is passed to the function and are then passed as a parameter to the recipes_search method within the FatSecret API. Recipes_search conducts a search within the recipe database based on the recipe names and ingredients.

The results are then looped through, adding the data needed for displaying the recipe preview to recipe_data array; recipe id, title, description and image. Not all recipes have an image therefore, the clearmyfridge logo image is used if no image exists. The recipe_data is then returned to be sent to the frontend.

```python
# Search Recipe based on Recipe ID
def searchRecipe(recipeID):
    recipe = fs.recipe_get(recipeID)

    recipedata = {}
    recipedata['id'] = recipe['recipe_id']
    recipedata['title'] = recipe['recipe_name']
    recipedata['description'] = recipe['recipe_description']
    recipedata['instructions'] = json.dumps(recipe['directions'])
    recipedata['ingredients'] = json.dumps(recipe['ingredients'])

    if 'recipe_images' in recipe:
        recipedata['image'] = recipe['recipe_images']['recipe_image']
    else:
        recipedata['image'] = 'https://res.cloudinary.com/dt3iqlezo/image/upload/v1677598676/Constants/logo_kt4v4g.png'

    if 'cooking_time_min' in recipe:
        recipedata['cooking_time_min'] = recipe['cooking_time_min']

    recipedata['servings'] = json.dumps(recipe['serving_sizes']['serving'])

    return recipedata
```

Figure 23 Search Recipe - recipesearch.py

The searchRecipe function, figure 23, searches the FatSecret API based on the recipe ID of the recipe the user wishes to view. The recipe ID is passed as a parameter to the function and is then passed to the recipe_get function within the FatSecret API. A dictionary, recipedata, is used to store the data needed for displaying the recipe to the user. A recipe id, title, description, instructions and ingredients are always present in the FatSecret recipes and are added to the recipedata dictionary. If there is an image in the recipe, the image is added otherwise the clearmyfridge logo is added. If the cooking time is present in the recipe data, it is also added. The recipe servings is added as a JSON value and the dictionary is returned to be sent to the frontend.

4.4 React

4.4.1 File Structure

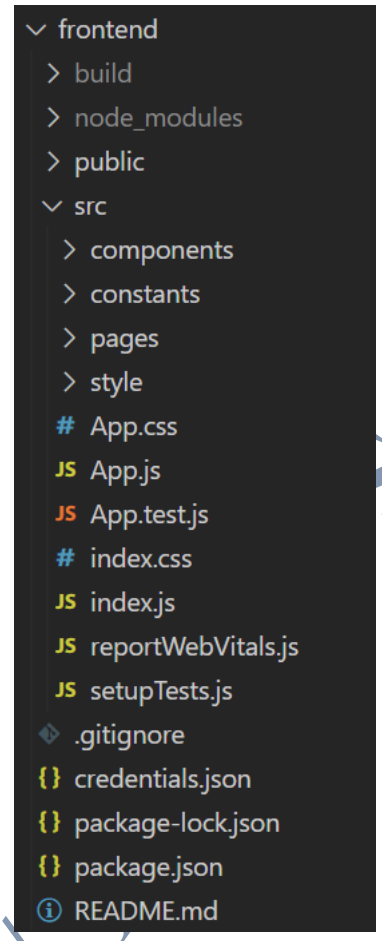


Figure 24 React Frontend File Structure

When a React app is created, files and directories are automatically created. The package.json file is used to manage the dependencies needed for the application and contains scripts need to run the application. Within the src folder, index.js is the main JavaScript file for the application. This file is responsible for rendering the React components to the Document Object Model, used to represent the web page.

Constants folder contains an index.js file that declares all the constants needed for this project. These consist of APIs to the Django backend, Cloudinary URL connection and Edanam API connection.

```
function App() {
  return (
    <BrowserRouter>
      <Switch>
        <Route exact path="/" component={withRouter(Home)} />
        <Route exact path="/about" component={withRouter(About)} />
        <Route exact path="/login" component={withRouter(Login)} />
        <Route exact path="/savedRecipes" component={withRouter(SavedRecipes)}/>
        <Route exact path="/recipes" component={withRouter(Recipes)} />
        <Route exact path="/:id" component={withRouter(RecipeDetails)} />
      </Switch>
    </BrowserRouter>
  );
}

export default App;
```

Figure 25 Page Navigation - App.js

App.js is the main component of the application that controls the user navigating between pages within ClearMyFridge. React Router library is used for routing in React, it enables navigation between views from different components, browser URL change and keeps the UI in sync with the URL. BrowserRouter is used to enable routing within this application. BrowserRouter uses HTML5 history API to keep the UI in sync with the URL. Within this application there is six different Route components. Each Route component defines the routes of the web application and the pages that should be rendered when a user visits a specific route.

4.4.2 Pages

For each page on ClearMyFridge a new JavaScript file is created. When a user navigates to a new page, BrowserRouter in App.js handles the change and displays components within each page to the user.

```
export default class Home extends Component {
  render() {
    return (
      <div className="app">
        <Header />
        <div className="container">
          <div className="search">
            <div className="heading">
              <SearchComponent />
              <WebcamComponent />
            </div>
          </div>
        </div>
        <div className="ingredients">
          <IngredientComponent />
        </div>
        <Footer />
      </div>
    );
  }
}
```

Figure 26 Home page - Home.js

The home page is the first page the user will see when they visit the web application. Header, SearchComponent, WebcamComponent, IngredientComponent and Footer components are imported into Home.js and displayed to the user. These components are discussed next in Section 4.4.3 Components.

Each page follow the same structure, header, components and footer. No functionality is carried out within the page files, only within the component files. Header and Footer components are imported and rendered for each page in the web application, allowing for a flow in the web application.

4.4.3 Components

Component files make up this web application. Each feature within a page has its own component associated with it. ClearMyFridge has the following features:

- Login/Register
- Search Ingredient
- Scan Ingredient Barcode
- Display Ingredients List
- Search Recipes
- View and Save Recipes

Each page within this web application contains one or more these features. Each page and their features are discussed within this section.

Login Page

The Login.js page consists of four components; LoginComponent, RegisterComponent, Header and Footer.

```
const [username, setUsername] = useState("");
const [error, setError] = useState(false);
const [password, setPassword] = useState("");

const handleSubmit = async (event) => {
  event.preventDefault();

  const user = {
    username: username,
    password: password,
  };

  // SEND LOGIN INFO TO DJANGO BACKEND
  axios
    .post(LOGIN_API_URL, user)
    .then((res) => {
      console.log(res);
      console.log(res.data);

      if (res.data["success"] == false) {
        setError("Invalid username or password");
      } else {
        localStorage.setItem("token", res.data.token);
        localStorage.setItem("username", username);
        window.location = "/";
      }
    })
    .catch((err) => {
      console.log(err);
      setError("Invalid username or password");
    });
};
```

Figure 27 Login Component

Figure 27 displays the Login Component functionality for the Login feature. A form is displayed to the user in which they can enter their username and password to sign in. State variables username and password use the useState hook to manage the user inputs.

HandleSubmit function handles the form submission event. This function creates a user object with the users username and password. A POST Request to the Django backend Login API endpoint using axios with the user object as the data payload. Axios is an open-source, promise-based HTTP client that uses JavaScript to send HTTP requests and manage responses. Axios post, get, put and delete requests are used throughout this web application to communicate with the Django backend and Edamam Food API.

If the backend responds with a success message, the user token and username is stored in the browsers localStorage and the user is redirected to the home page using window.location.

Otherwise, an error message is set in the error state variable and displayed to the user within the form.

```
const [email, setEmail] = useState("");
const [error, setError] = useState("");
const [password, setPassword] = useState("");
const [lastName, setLastName] = useState("");
const [firstName, setFirstName] = useState("");

const handleSubmit = async (e) => {
  e.preventDefault();

  // SEND REGISTER INFO TO DJANGO BACKEND
  axios
    .post(REGISTER_API_URL, {
      firstName: firstName,
      lastName: lastName,
      username: firstName + lastName,
      password: password,
      email: email,
    })
    .then((res) => {
      console.log(res);
      console.log(res.data);
      localStorage.setItem("token", res.data.token);
      localStorage.setItem("username", firstName + lastName);
      window.location = "/";
    })
    .catch((err) => {
      console.log(err);
      setError("Failed to register user");
    });
};
```

Figure 28 Register Component

Figure 28 displays the register component functionality for registering a new account for ClearMyFridge. RegisterComponent renders a registration form that includes firstname, lastname, password and email fields. The username is automatically generated by combining the users firstname and lastname together. As the user enters their details, onChange handler updates the corresponding state variables.

When a user submits the form, handleSubmit function is called. This function sends a POST request to the Django backend register API (see section 4.3.4 Views for more details) using axios. If the registration is successful, the users authentication token is stored in the browsers local storage along with the username and the user is redirected to the home page. If the registration fails, an error message is displayed to the user.

Home Page

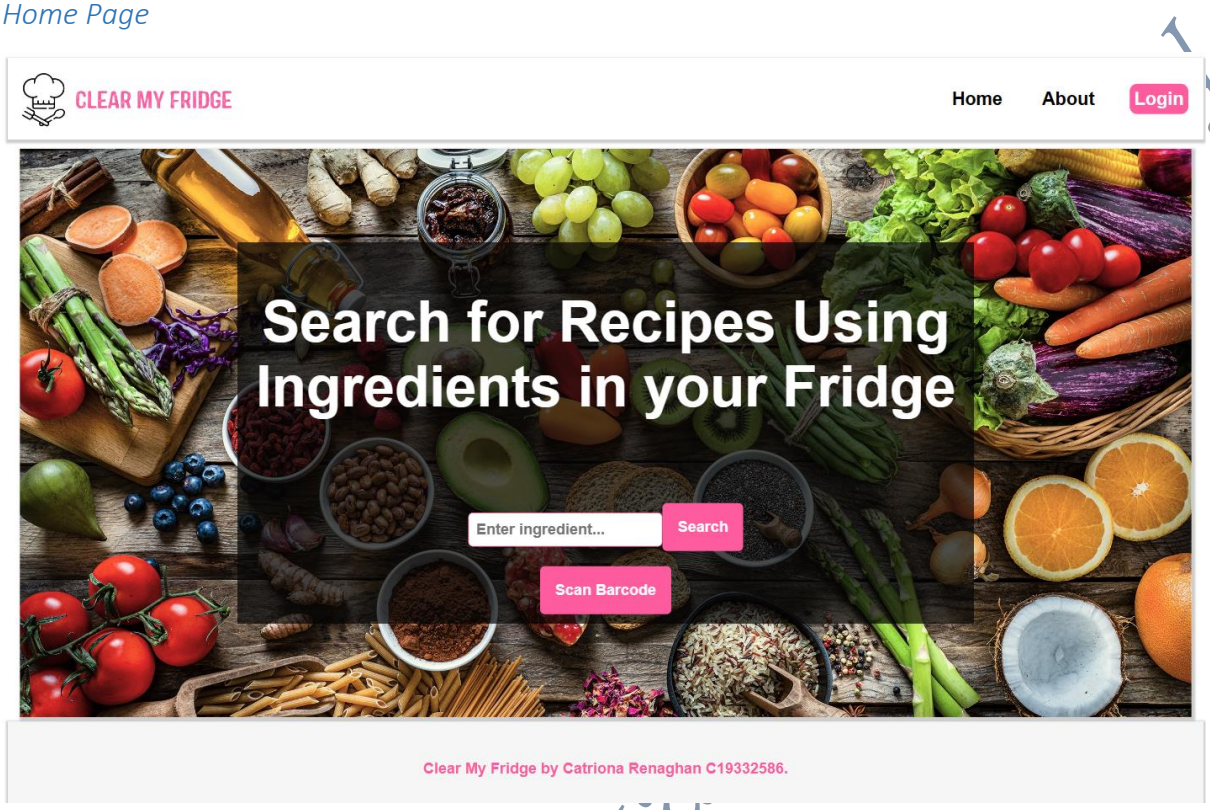


Figure 29 ClearMyFridge Home Page

ClearMyFridge Home page consists of five different components; Header, SearchComponent, WebcamComponent, IngredientComponent and Footer. The display of the home page depends on if the ingredient list is populated or not. Figure 29 displays the home page when the ingredient list is unpopulated.

```
// SEARCH INGREDIENT NAME IN EDAMAM API
const searchIngredient = async () => {
  setClicked(true);
  const api_url =
    FOOD_API_URL + "&ingr=" + ingredientSearch + "&nutrition-type=cooking";
  await axios
    .get(api_url)
    .then((res) => {
      console.log(res);
      console.log(res.data);
      setProduct(res.data);
      displayProducts(res.data);
    })
    .catch((err) => {
      console.log(err);
      setError("No Product Found in Database");
    });

};

// SEND INGREDIENT TO DJANGO BACKEND
const onClickHandler = (event, product) => {
  const ing_url = INGREDIENT_API_URL;
  var product = product.replaceAll('"', "");

  let storedUsername = localStorage.getItem("username");
  console.log("storedUsername=" + storedUsername);

  if (storedUsername == null) {
    storedUsername = "Guest";
  }

  // SEND INGREDIENT DETAILS TO INGREDIENT API ENDPOINT
  axios.post(ing_url, { productName: product, user: storedUsername }).then(
    (response) => {
      console.log(response);
      setClicked(false);
    },
    (error) => {
      alert(error);
    }
  );
};
```

```
const handleSubmit = (event) => {
  event.preventDefault();
  searchIngredient();
};

// DISPLAY EDAMAM RESPONSE INGREDIENTS
function displayProducts(products, error) {
  if (error === "No Product Found in Database") {
    return (
      <div className="products">
        <button>No Product Found in Database</button>
      </div>
    );
  }

  if (!products || !products.hints || products.hints.length === 0) {
    return null;
  }

  return (
    <>
      <div className="products">
        {products.hints.slice(0, 5).map((hint) => (
          <button
            key={hint.food.foodId}
            onClick={(event) => onClickHandler(event, hint.food.label)}>
            {hint.food.label}
          </button>
        ))}
      </div>
    </>
  );
}
```

Figure 30 Search Component

Search Component, figure 30, searches Edamam Food API for food items based on the product name entered by the user. The component renders a form with a search bar that the user can enter an ingredient name into. When the user clicks the Search button, handleSubmit calls the searchIngredient function to search the ingredient name in the Edamam API.

The searchIngredient function makes a HTTP GET request to the Edamam API URL that consists of an Edamam ID, key and ingredient entered by the user. If the request is successful the displayProducts function is called, otherwise the error state variable is set to display an error message.

The displayProducts function displays the response data from the Edamam Food API. The first five ingredients in the response are displayed as buttons. If there is no products, an error message is displayed to the user. When a user clicks on a product button, onClickHandler event handler is called.

onClickHandler function retrieves the users username from local storage, if the user has not logged in the username is set to guest. The product name and the username is then sent to the Django backend using axios POST request to the Ingredient API endpoint (see section 4.3.4 Views for details). If the POST request fails, the user is alerted with the error.

WebcamComponent is rendered next within the Home page. React webcam and image uploaded is discussed later in section 4.5 Image Processing. WebcamComponent renders a Scan Barcode button that opens the users camera. After the image is captured and uploaded to Cloudinary and sent to Django backend. Using the barcode returned from the Image API endpoint, getIngredients function sends a GET request to the Edamam Food API with the barcode as a parameter. If the API returns an error, the barcode variable is set to an error messge, otherwise, product is set to the response data.

displayProduct function renders a button with the product name from the Edamam Food API. When the user clicks on the button onClickHandler function is called and the ingredient is added to the Ingredient database as explained above in the SearchComponent onClickHandler function.

```
// GET INGREDIENTS FROM DJANGO BACKEND
useEffect(() => {
  let storedUsername = localStorage.getItem("username");
  if (storedUsername == null) {
    storedUsername = "Guest";
  }
  console.log("storedUsername=" + storedUsername);
  axios
    .get(`${INGREDIENT_API_URL}?user=${storedUsername}`)
    .then((res) => {
      console.log(res);
      console.log(res.data);
      setIngredients(res.data);
    })
    .catch((err) => {
      console.log(err);
    });
}, [ingredients]);

// DELETE INGREDIENT FROM DATABASE
const removeIngredient = (event, ingredient) => {
  const ing_url = INGREDIENT_API_URL + ingredient;
  console.log(ing_url);

  axios.delete(ing_url).then(
    (response) => {
      console.log(response);

      axios
        .get(INGREDIENT_API_URL)
        .then((res) => {
          console.log(res);
          console.log(res.data);
          setIngredients(res.data);
          if (res.data.length == 0) {
            const search = document.querySelector(".search");
            const heading = document.querySelector(".heading");
            search.classList.remove("search--with-ingredients");
            heading.classList.remove("heading--with-ingredients");
          }
        })
        .catch((err) => {
          console.log(err);
        });
    },
    (error) => {
      alert(error);
    }
  );
};
```

```
// DISPLAY LIST OF INGREDIENTS
const displayIngredients = (ingredients) => {
  const search = document.querySelector(".search");
  const heading = document.querySelector(".heading");
  if (ingredients == 0) return null;

  if (ingredients.length > 0) {
    search.classList.add("search--with-ingredients");
    heading.classList.add("heading--with-ingredients");
    return (
      <>
        <link
          rel="stylesheet"
          href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/4.7.0/css/font-awesome.min.css"
        ></link>
        <div className="ingredient-container">
          <div className="ingredient-list">
            <ul>
              <h1>You're Searching for Recipes with:</h1>
              {ingredients.map((item) => (
                <li className="products-list">
                  {item.productName}
                  <i
                    class="fa fa-minus minus"
                    onClick={(event) => removeIngredient(event, item.id)}
                  ></i>
                </li>
              ))}
              <br></br>
              <Link className="search-recipe-btn" to="/recipes">
                Find Recipes
              </Link>
            </ul>
          </div>
        </div>
      </>
    );
  } else {
    search.classList.remove("search--with-ingredients");
    heading.classList.remove("heading--with-ingredients");
    return null;
  }
};
```

Figure 31 Ingredient Component

Ingredient Component displays the lists of ingredients to the user when database is populated. The useEffect hook retrieves the username stored in local storage, if the username is null it is set to guest and fetches the ingredients from the Django Ingredient API when the

component is rendered or when the ingredients state variable changes. The response data is then set to the ingredients state variable using setIngredients.

When the developer was hosting the application in Heroku, a cloud based MySQL database had to be used instead of a local MySQL database. This meant that the requests to the database was limited for each user. To overcome this issue, instead of a GET request fetching the ingredients list each time the component renders, a 15 second wait was added. This ensures that the user can continue to use the web application without being blocked.

DisplayIngredients function returns JSX code to display a list of the ingredients. If the ingredient list is populated CSS classes are added to adjust the display of the home page accordingly, if the ingredients lists is empty, the CSS classes are removed to display the home page in Figure 29. A minus symbol is displayed beside each ingredient, when the user clicks the symbol, removeIngredient function is called. Below the ingredient list, a link to the Recipes page redirects the user to this page. This page is discussed next within this section.

removeIngredient takes event and ingredient object as arguments. The ingredient ID is added to the Django Ingredient API URL and a HTTP DELETE request is sent to Ingredient API endpoint (see section 4.3.4 Views for details). The updated ingredients list is then fetched again using a GET request and display is updated.

Recipes Page

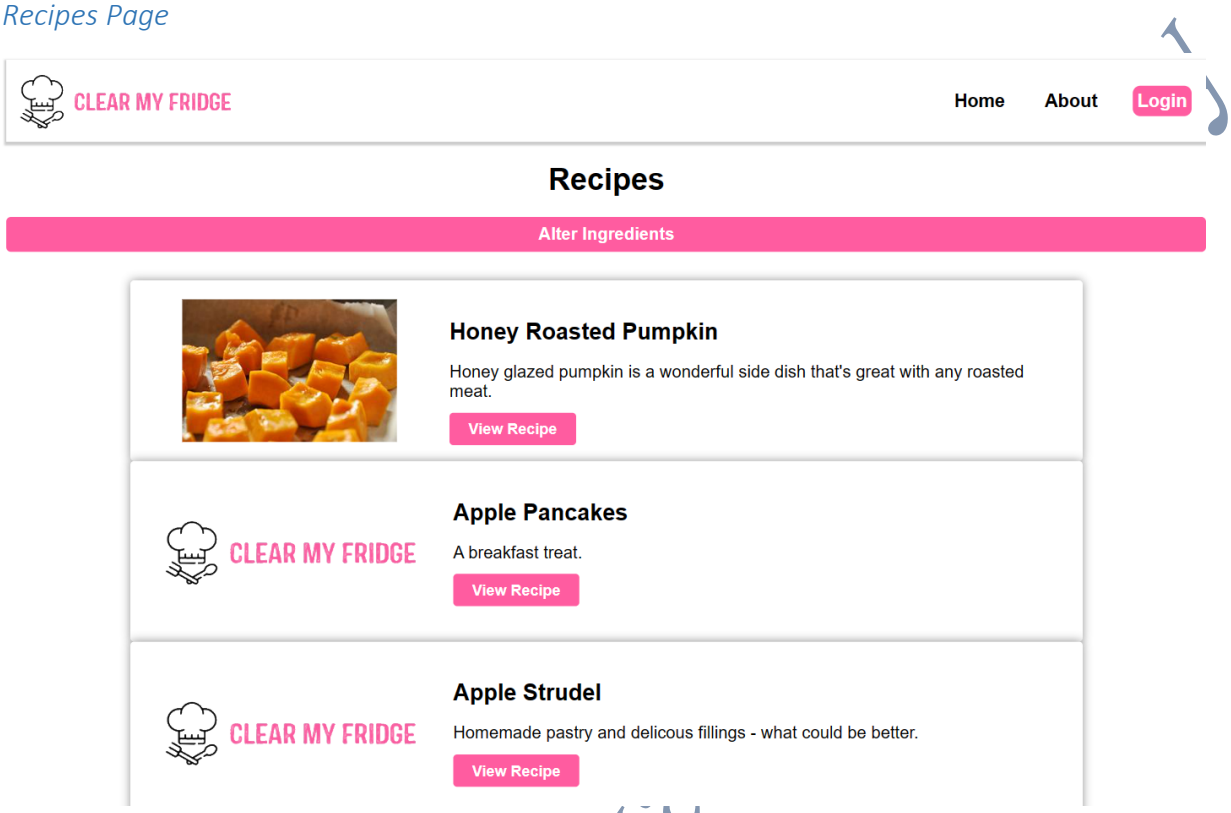


Figure 32 ClearMyFridge Recipes Page

ClearMyFridge Recipes page, Figure 32, contains three components; Header, RecipeComponent and Footer. This page contains the search recipes feature.

```
const [recipes, setRecipes] = useState("");

// GET RECIPES FROM DJANGO BACKEND
const getRecipes = async () => {
  try {
    let storedUsername = localStorage.getItem("username");
    if (!storedUsername) {
      storedUsername = "Guest";
    }

    const res = await axios.get(`${RECIPES_API_URL}?user=${storedUsername}`);
    setRecipes(res.data);
  } catch (err) {
    console.log(err);
  }
};

useEffect(() => {
  getRecipes();
}, []);
```

```
// DISPLAY RECIPES RETURNED FROM BACKEND API
const displayRecipes = (recipes) => {
  if (!recipes) return null;
  return (
    <div className="recipe-container">
      <div className="recipe-heading">
        <h1>Recipes</h1>
        <Link className="recipe-btn" to="/">
          Alter Ingredients
        </Link>
      </div>
      <div className="recipe-display">
        <ul className="recipe-list">
          {recipes.map((recipe) => (
            <li key={recipe.id} className="recipe-details">
              <img
                className="recipe-image"
                src={recipe.image}
                alt={recipe.title}
              />
              <div className="recipe-text">
                <h2 className="recipe-title">{recipe.title}</h2>
                <p className="recipe-desc">{recipe.description}</p>
                <Link
                  className="view-btn"
                  to={`${recipe.title.replace(/\s+/g, "")}?id=${recipe.id}`}
                >
                  View Recipe
                </Link>
              </div>
            </li>
          ))}
        </ul>
      </div>
    </div>
  );
};
```

Figure 33 Recipe Component

RecipeComponent in Figure 33 displays recipes containing the ingredients the user entered on the Home page. The useEffect hook is used to call the getRecipes function when the component is mounted. getRecipes fetches the recipes from the recipe Django API and stores the recipes within the recipes state variable using setRecipes function.

displayRecipes function takes the recipes as an input and loops through each recipe using map function. For each recipe the image, title and description is displayed. A button "View Recipe" is displayed beside each recipe linking to RecipeDetails page with the recipe ID passed in the URL.

Recipe Details Page

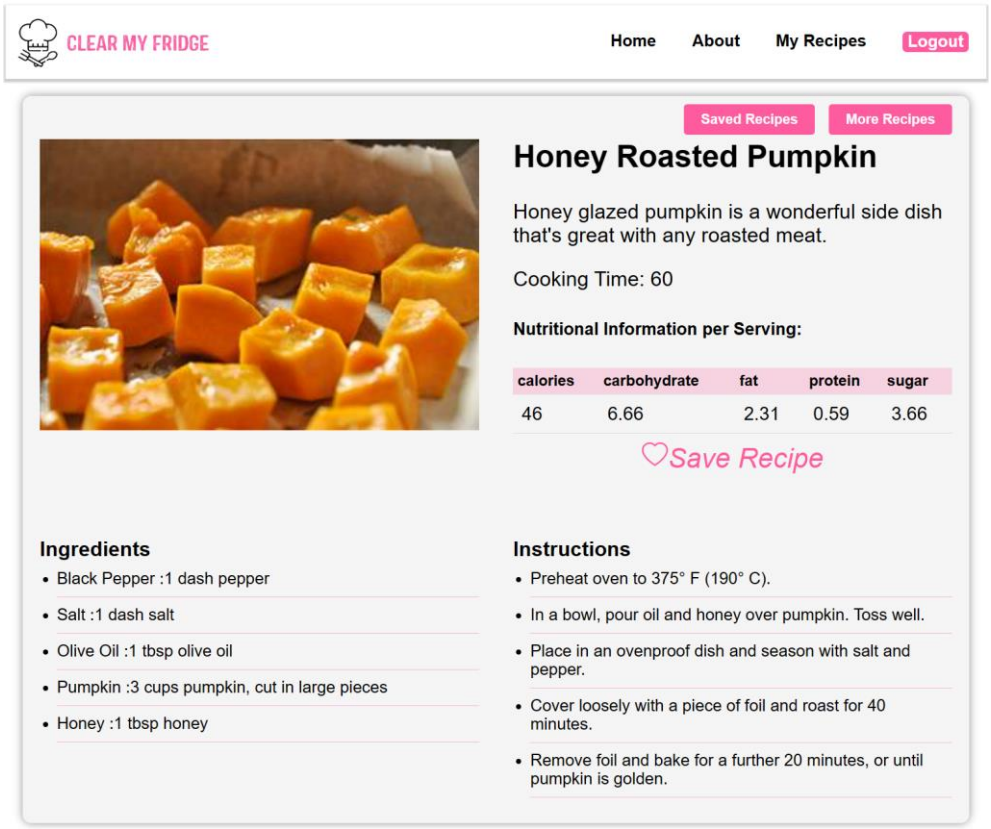


Figure 34 Recipe Details Page

RecipeDetails page contains three components; Header, ViewRecipeComponent and Footer. ViewRecipeComponent illustrates the View Recipe Feature.

Figure 35 View Recipe Component

Display Recipe Component displays a recipe and allows users who have logged in to save and unsave the recipe to their account. The recipe ID is extracted from the URL using URLSearchParams. useEffect hook calls sendRecipe() function which sends a POST HTTP Request to the View Recipes API (discussed in section 4.3.4 Views) within the Django backend with the recipe ID extracted. Upon success, the recipe data is then retrieved using axios GET request to the same API and displayRecipeDetails function is called.

displayRecipeDetails parses the recipe data from the backend API response that is of JSON format and then displays the recipe image, title, description, cooking time if it exists, nutritional information, ingredients and instructions. If the user has logged in, a save recipe heart icon and text is displayed, if the user has not logged in this is not displayed. useEfeect checks if the recipe has been saved for the user logged in by sending a GET request to the Save Recipe API and looping through each recipe to check if any recipe ID matches the recipe ID of the recipe being displayed. If the recipe is within the database, the heart icon is filled and the text changes to “Unsave Recipe”.

handleClick() function handles saving recipe feature. If the user saves the recipe a HTTP POST request is made to the Save Recipe API to save it. If the recipe is already saved and the user

wished to unsave the recipe, a HTTP DELETE request to the API is sent to remove the recipe from the database. The liked stated is toggled accordingly to update the display of the button.

Saved Recipes Page

Saved Recipes page contains three components; Header, SavedRecipeComponent and Footer. Only users who have logged in have access to this page. Users can navigate to this page by clicking “My Recipes” in the navigation bar or by clicking “Saved Recipes” button when viewing a recipes details.

SavedRecipeComponent is very similar to RecipesComponent discussed above under Recipes Page. When the component is mounted, the saved recipes are retrieved by sending a HTTP GET request to the Save Recipe API. These recipes are displayed in the same way as seen in Figure 32.

4.4.4 Style

Five different CSS files are used to style ClearMyFridge web application in order to make it user friendly; Header-Footer, Home, Login-Register, Recipe and RecipeDetails. The same colour palette is used throughout all the pages and similar features in different pages use the same styling, for example displaying recipes and saved recipes both use Recipe.css.

CSS is also used to make the web application responsive to screen size and for different features. As discussed earlier for webcam feature and ingredients list, CSS classes are added to adjust the home screen. These CSS classes are added and removed according to if the users camera is open and if there is ingredients within the database.

When the screen width is less than 768px, ingredients list on the home page is no longer displayed to the right of the search ingredients component, instead it is displayed below, ensuring components on the screen are not overlapping. Similarly within Recipes and Saved Recipes pages, recipe cards adjust to display images above recipe titles, description and view recipe buttons. Recipe Details page displays all recipe items in a column when the screen size is adjusted.

The navigation bar also adjusts when the screen size is reduced. Instead of each page tab displaying in a row, an icon appears that creates a drop down menu when the user clicks on it.

4.5 Image Processing

The image processing part of this web application consists of capturing the users image of the product, uploading the image to a Cloudinary database and extracting the barcode from the image. Capturing and uploading the image is completed within the frontend React application whereas extracting the barcode is completed within the Django backend.

4.5.1 Capture Image

React Webcam

```
// OPEN CAMERA
const startCam = () => {
  console.log("startCam");
  const search = document.querySelector(".search");
  const heading = document.querySelector(".heading");
  const container = document.querySelector(".container");
  search.classList.add("search--with-camera");
  heading.classList.add("heading--with-camera");
  container.classList.add("container--with-camera");
  setCamera(true);
};

// CLOSE CAMERA
const stopCam = () => {
  console.log("stopCam");
  let stream = videoElement.current.stream;
  const tracks = stream.getTracks();
  tracks.forEach((track) => track.stop());
  removeCSS();
  setCamera(false);
};

// TAKE PICTURE OF PRODUCT
const capture = useCallback(() => {
  console.log("capture");
  const capImg = videoElement.current.getScreenshot();
  removeCSS();
  setImage(capImg);
  uploadImage(capImg);
  setCamera(false);
}, [videoElement]);
```

```
return (
  <div>
    <div>
      {camera && (
        <div className="popup">
          <div className="popup-data">
            <Webcam
              className="webcam"
              audio={false}
              ref={videoElement}
              mirrored={true}
              videoConstraints={videoConstraints}
            />
            <div className="btn-container">
              <button className="search-child btn" onClick={capture}>
                Capture
              </button>
              <button className="search-child btn" onClick={stopCam}>
                Close Camera
              </button>
            </div>
          </div>
        </div>
      )}
    </div>
    {!camera && (
      <button className="search-child btn" onClick={startCam}>
        Scan Barcode
      </button>
    )}
    <div>
      {displayError()}
      {displayProduct(product)}
    </div>
  </div>
);
```

Figure 36 React Webcam - WebcamComponent.js

React webcam is a JavaScript library that is used to include the features; open camera, capture image and close camera. The user clicks “Scan Barcode” button on the home page, startCam() function is called. This function adds CSS classes to the webcam elements to adjust the display when the camera is open and the camera state is set to true. The uploadImage is also called in this function which is discussed next in this section.

When the camera is true, a pop up with the camera feed is displayed with two buttons: Capture and Close. When the Capture button is clicked the capture() function is called and takes a screenshot of the current frame within the camera footage and it is saved as an image. The CSS classes are removed to return the home page back to its original state and the camera

state is set to false, resulting in the camera closing. The camera can also be closed by clicking the close button, which stops the camera stream, removes CSS classes and sets camera to false.

Upload Image

```
const uploadImage = async (img) => {
  if (img == null) {
    return;
  }
  fileData.append("file", img);
  fileData.append("upload_preset", "unsigned");

  await fetch(CLOUDINARY_URL, {
    method: "POST",
    body: fileData,
  })
    .then((res) => res.json())
    .then((data) => {
      console.log("Success:", data);
      console.log("secure url: " + data.secure_url);

      // SEND IMAGE URL TO DJANGO BACKEND
      axios
        .post(IMAGE_API_URL, {
          title: "temp",
          image: data.secure_url,
        })
        .then((res) => {
          console.log(res);
          console.log(res.data);
          setBarcode(res.data["barcode"]);
        })
        .catch((err) => {
          console.log(err);
        });
    })
    .catch((err) => console.error("Error:", err));
};
```

Figure 37 Upload Image - WebcamComponent.js

The uploadImage() function takes the image file saved within the capture() function above as an argument appends it to a FormData, along with an upload present needed for uploading to Cloudinary. A POST Request sends the FormData object as the request body to the Cloudinary API. If the response is successful, the secure URL of the uploaded image is retrieved from the response data and it is sent the Django backend Image API using an axios POST request. The backend API response then contains the barcode (discussed next) and is

set to the barcode state variable and is used to carry out other features within the WebcamComponent.

4.5.2 Extract Barcode from Image

Barcode extracting functionality is carried out within the extract_barcode_nums function within the imgprocessing.py file. This file is within the Django backend and is called upon within the imageView POST request.

```
def extract_barcode_nums(img, url):

    if url:
        # READ IMAGE FROM CLOUDINARY URL
        req = urllib.request.urlopen(img)
        arr = np.asarray(bytearray(req.read()), dtype=np.uint8)
        img = cv2.imdecode(arr, cv2.IMREAD_COLOR)

        img = cv2.flip(img, 1)
    else:
        img = cv2.imread(img)

    # CONVERT TO GRAYSCALE
    greyImg = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    # APPLY GRADIENT FILTER
    depth = cv2.cv.CV_32F if imutils.is_cv2() else cv2.CV_32F
    gradX = cv2.Sobel(greyImg, ddepth=depth, dx=1, dy=0, ksize=-1)
    gradY = cv2.Sobel(greyImg, ddepth=depth, dx=0, dy=1, ksize=-1)

    gradient = cv2.subtract(gradX, gradY)
    gradient = cv2.convertScaleAbs(gradient)

    # APPLY BLUR AND THRESHOLD
    blurImg = cv2.blur(gradient, (9, 9))
    (_, thresh) = cv2.threshold(blurImg, 225, 255, cv2.THRESH_BINARY)

    # APPLY MORPHOLOGY
    kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (21, 7))
    closed = cv2.morphologyEx(thresh, cv2.MORPH_CLOSE, kernel)
```

Figure 38 Image Preparation - imgprocessing.py

In order to read the barcode within the image, the image has to be prepped and transformed.

If URL is not None the URL is read. As the image is stored and passed a Cloudinary URL, urllib package is used. Urllib package is a URL handling module used for python. Urllib.request.urlopen() is used to read the image from the Cloudinary URL. Numpy as array()

converts the image into an array and OpenCV imdecode() function reads the array data and converts it into an image. The image is then flipped, allowing the numbers to appear correctly and not backwards. If the URL is None, OpenCV uses imread() to read the image from a local file. The URL is only None when running tests.

Figure 39 Gradient Filtering and Morphology

After the image is successfully read, the image is converted to grayscale using OpenCV cvtColor and gradient filtering is applied. OpenCV Sobel operator is an edge detection filter than is applied to both horizontal and vertical directions of the grayscale image resulting in gradX and gradY images that represent the gradient. The depth used within the Sobel operation represents the gradient magnitude values. Gradient is then calculated by subtracting gradY from gradX, resulting in an image that highlights the edges. As seen in Figure 39, after a gradient filter is applied to the image, any objects within the image have a highlighted edge.

Next image blurring, thresholding and morphology operations are applied to the gradient image. First Gaussian blur is applied to the image to smooth and reduce noise and detail. Next a binary threshold is applied to the blurred image, pixels equal to 225 are set to 225 (white) and any value less than 225 are set to 0 (black). This creates a binary image with edges and features highlighted in white and background in black. Morphology closing operation is applied to the binary image to fill in small gaps and smooth the edges. The closed image is a more complete segmentation of the objects within the image. As seen in Figure 39, when

blurring, thresholding and morphology is applied to the gradient image, the barcode is highlighted.

```
# FIND CONTOURS
cnts = cv2.findContours(
    closed.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
cnts = imutils.grab_contours(cnts)
sortedCNTS = sorted(cnts, key=cv2.contourArea, reverse=True)[0]

# FIND BOUNDING BOX
(x, y, w, h) = cv2.boundingRect(sortedCNTS)
(x, y, w, h) = (x - 30, y - 30, w + 60, h + 60)
cropImg = img[y:y + h, x:x + w]
cv2.rectangle(img, (x, y), (x + w, y + h), (0, 0, 255), 2)

# READ BARCODE
readBarcode = pyzbar.decode(cropImg)

if not readBarcode:
    print("No barcode detected")
    barcodeNum = "No barcode detected"
else:
    for barcode in readBarcode:
        barcodeNum = barcode.data.decode("utf-8")
        text = "{}".format(barcodeNum)
        print("Barcode Number: {} ".format(barcodeNum))

return barcodeNum
```

Figure 40 Barcode Detection and Reading

Finally contour detection, bounding box calculation and barcode reading is completed. OpenCV findContours() function is used to detect the contours within the morphed image. RETR_EXTERNAL flag retrieves only the external contours of the objects and CHAIN_APPROX_SIMPLE flag compresses the contours by removing unnecessary points. The result is a list of contours as numpy arrays. Grab_contours() function extracts the counters so that they can be used. The contours are then sorted by area from largest to smallest and the largest contour, barcode area, is stored in sortedCNTS.

The bounding box of the largest contour is calculated and the x-coordinate, y-coordinate, width and height of the bounding box are stored. The values are adjusted to make the box slightly larger, ensuring the barcode numbers are within the bounding box. The image is then cropped to the region of the bounding box.

Pyzbar decode() function reads the barcode from the cropped image. If no barcode is detected, barcodeNum is assigned “No barcode detected” otherwise the barcode data extracted from the decoded image is stored in barcodeNum. BarcodeNum is then returned to the imageView where the extracte_barcode_num function is a called. With the example image in Figure 39, the barcode number 80177173 is returned.

4.6 Conclusion

In conclusion, ClearMyFridge is a web application hosted using Heroku, with a Django backend and React.js frontend. Heroku uses a dyno, container, with the Django backend and react build, and uses Gunicorn to start the webserver.

The Django backend consists of models and views. Models are used to create tables within the MySQL database and views are used to interact with the database. Connecting to the FatSecret API also happens in the Django backend, along with image processing techniques to extract the barcodes.

The React frontend consists of components, each containing a feature for the web application. Within these components, HTTP requests are made to Django backend to add and receive data from the database.

5. Testing and Evaluation

5.1. Introduction

Software testing involves evaluating and verifying that the program does what it is meant to. Testing within this project helped prevent bugs and improve performance. In this section, different types of testing are discussed in relation to this project.

5.2. System Testing

There is two categories of testing; Functional Testing and Non-Functional Testing.

5.2.1 Functional Testing

Unit Testing

Unit testing is a type of white box testing that involves testing individual units or components of the system. This ensures that each component of the system performs as expected. Unit tests are written throughout development, preventing bugs when changes are made. When unit tests are written for each function, when changes are made the unit tests can identify any problems quickly.

```python
class food_api_test(TestCase):
    # Test Edamam API Connection
    def test_api(self):
        url = settings.EDAMAM_API_URL + '&ingr=apple&nutrition-type=cooking'
        response = request.urlopen(url)
        self.assertEqual(response.status, 200)

    # Test Edamam API Response - Food Name Search
    def test_api_name_results(self):
        url = settings.EDAMAM_API_URL + '&ingr=apple&nutrition-type=cooking'
        response = request.urlopen(url)
        responseText = response.read()
        self.assertIn(b'apple', responseText)

    # Test Edamam API Response - Barcode Search
    def test_api_barcode_results(self):
        url = settings.EDAMAM_API_URL + '&upc=80177173&nutrition-type=cooking'
        response = request.urlopen(url)
        responseText = response.read()
        self.assertIn(b'Nutella', responseText)


class recipe_api_test(TestCase):
    # Test Fatsecret API Response - Search Ingredient
    def test_api(self):
        fs = Fatsecret(settings.FATSECRET_API_KEY,
                       settings.FATSECRET_API_SECRET)
        foods = fs.recipes_search("apple")
        self.assertTrue(foods)

    # Test Fatsecret API Response - Recipe ID
    def test_api_recipe(self):
        fs = Fatsecret(settings.FATSECRET_API_KEY,
                       settings.FATSECRET_API_SECRET)
        recipe = fs.recipe_get(13172)
        recipe_name = recipe['recipe_name']
        self.assertEqual(recipe_name, "Lemon Mousse")


class barcode_api_test(TestCase):
    # Test barcode Processing
    def test_barcode(self):
        img = "./media/images/nutella.jpg"
        barcode = ExtractBarcode.extract_barcode_nums(img, url=False)
        self.assertEqual(barcode, "80177173")
```

Figure 41 Unit Tests - tests.py

As features were developed within this project, a unit test was implemented to ensure it is working as expected. Figure 41 displays 3 test classes for testing different functionalities within this project.

The food_api_test class contains three different test methods for testing the Edamam Food API. The first method, test_api, makes a GET request to the Edamam API URL that is stored in the settings file containing the secret key and ID and with the ingredient apple. It asserts that the response status is 200, indicating a successful connection. The second method, test_api_name_results, tests the same API URL as test_api, but this time it checks the response contains the word “apple”. This checks that the API returns the correct data and the products that are expected to be returned. The third method, test_api_barcode_results tests the Edamam API again, but this time it searches for a product using the Universal Product Code (UPC) of a Nutella jar barcode. It asserts that the response contains the word “Nutella”, ensuring the correct product is returned.

The recipe_api_test class contains two test methods that test the FatSecret Recipe API. The first method, test_api, makes an instance of the FatSecret class with the API key and secret from the settings file and calls the recipe_search method with “apple” as the ingredient. It asserts that the result is not empty, ensuring that at least one recipe was found. The second method, test_api_recipe, also makes an instance of the FatSecret class and calls the recipe_get method with the recipe ID 13172. It then asserts that the response recipe name is equal to “Lemon Mousse”, indicating the correct recipe is returned.

The barcode_api_test class contains one test method, test_barcode that tests the barcode extracting feature. It calls the extract_barcode_nums function from imgprocessing.py file with the image path to a locally stored image and the URL set to false, ensuring the image is read correctly from the file. It asserts that the result is expected to equal the barcode number 80177173. This test checks that the image processing feature correctly extracts the barcode number from the image.

System Testing

System testing is a type of black box testing that is the process of evaluating how application components interact together. It ensures the application works as it is designed to. As features were developed, pervious features were tested to ensure that the new feature did not "break" any previously implemented features. For example, when Login requirement was implemented, the developer tested the add ingredient and scan barcode features if the user had logged in and logged out. This ensured any new code did not have a negative effect on the previous features. When bugs appeared, they were fixed there and then, allowing for a fully functional system at all times.

Acceptance Testing

Acceptance testing involves the customer testing the application. User stories were provided to a friend and user story was carried out on the system. If the user story could be completed with the expected result showing, it passes. As there is no customer for this project, a friend tested the system with guidelines provided. User Acceptance testing ensures bugs are caught and fixed and functionality works as it should.

Prior to giving the friend the user stories to follow, the friend was asked to "play around" on the website to familiarise themselves with the application. Upon doing so, an error with recipes displaying with correct ingredients displayed to the tester. This bug was corrected before the tester was given the user stories to test the features correctly.

UAT Testing						
Test Cases to test Features of Clear My Fridge						
No.	Feature	Page	Step	Description	Expected Results	Status
1	Register	Login	1	Click Login Button in navigation bar	User is on Login page with login and register form displaying	Pass
			2	Enter First name, Last name, Email and Password	Username is auto filled with first and last name combined	Pass
			3	Click Register	User is redirected to home page	Pass
					Logout and MyRecipes Pages in Navigation bar displayed	Pass
2	Logout	Home	1	Click logout button in navigation bar	Login button appears in navigation bar and MyRecipes page appears	Pass
3	Register	Login	1	Follow same steps as test case No. 2	Error message "Failed to register user" is displayed	Pass
4	Login	Login	1	Click Login Button in navigation bar	User is on Login page with login and register form displaying	Pass
			2	Enter username (firstname + lastname) and password	User is redirected to home page	Pass
					Logout and MyRecipes Pages in Navigation bar displayed	Pass
5	Home Display	Home	1	Navigate to Home page	Background image is size of screen size	Pass
6	Add Product by Name	Home	1	Click on text box and enter apple	Text can be entered into text box	Pass
			2	Click Search	5 different types of apples appear to choose from	Pass
			3	Select "apple"	Image size adjusts	Pass
					Products buttons disappear	Pass
					Ingredients list displays on right of screen or below (depends on screen size)	
					Apple appears in Ingredient list	Pass
7	Add Product by Barcode	Home	1	Navigate to Home page	Scan Barcode button seen	Pass
			2	Click Scan Barcode	Camera opens	Pass
					Capture and Close Camera buttons appear	Pass
			3	Hold barcode in front of camera and click "Capture" button	Camera closes and displays adjusts	Pass
					Honey button appears	Pass
			4	Click Honey	Honey is added to the ingredient list	Pass
8	Remove Ingredient	Home	1	Click - sign beside Apple	Apple ingredient is removed from list	Pass
9	Search Recipes	Home	1	Navigate to Home page	Search Recipe button Appears under Ingredients List	Pass
			2	Click Search Recipes	User is redirected to Recipes Page	Pass
					Recipes with "Honey" in the title appear	Pass

Figure 42 User Acceptance Testing

Figure 42 displays the first 9 of 16 User Stories that were presented to the friend that tested the system for the developer. Each user story contains step by step instructions and expected result for each step, making it easy for the tester to follow and use the system. When the tester completed each step, the status is set to either pass or fail. Fortunately, each user story passed and the tester was presented with the expected result.

5.2.2 Non-functional Testing

Usability Testing

Usability testing is the process of testing the feel and user-friendliness of the application. This consists of cross browser testing and ensures the application is simple and easy to use. [36]

The system was tested across three different browsers; Edge, Google Chrome and Opera. The web application worked as expected in each browser, allowing the application to reach a wider audience.

5.3. System Evaluation

Evaluation the web application consists of verification and validation.

Verification

Verification is determined whether the system has no technical errors and that the requirements are met. Verification proves that all requirements have been met and anything out of scope is not considered is not verified. System testing is also a form of verification. When all system unit tests pass and functionality is working as expected, the system is verified.

All requirements stated before and during development were met within this application. Whilst the application was not hosted using Digital Ocean as planned due to permission and access issues, Heroku was researched and used instead allowing this requirement to still be met. With these requirements met and the developer's unit-test passed the application is sufficiently passed the verification stage.

Validation

Validation consists of the system being user-friendly and acceptable. Adaptability is an important part of validation. The system must meet the users' needs and must be developed so that future development can be made. The program must work in different environments. User Interface should be friendly and simple to use and follow. [37]

The user interface promotes and 'ease of learning' format whereby, the user can intuitively interact with the application with little to no knowledge of the system prior to using it. The

acceptance testing illustrates success in system validation and that the quality of the project and that the system features are robust and meet the users expectations.

5.4. Conclusion

Testing is an important process of the development stage of the project. Unit, system and acceptance testing ensured the application features work as expected and usability testing ensured the web application is accessible for all users. After completing the testing, it is fair to say that the system behaves as expected in a real world environment.

6. Conclusions and Future Work

6.1 Limitations

Edamam Food API contains nearly 900,000 food within its database, however most of these foods do not contain UPC codes. When searching Edamam Food Database API using the barcode, foods sold worldwide proved to be more successful than food items sold locally. This makes searching for food products using the barcode limited.

FatSecret Recipe API has its limitations when returning searching for recipes by ingredients and by recipe ID. When searching for recipes using the ingredients the user has entered, FatSecret only returns 20 recipes by default, this can be changed but is limited to a maximum of 50. This restricts the number of recipes displayed to the user. Alongside this, not all recipes contain the same structure. Not every recipe contains an image or cooking time. The ClearMyFridge logo has been added to the recipes with no image.

JawsDB MySQL database is used to store data within the webserver. This database has a limited number of user requests forcing the developer to change how ingredients are shown to the user.

6.2 Conclusion

After months of researching, project planning and developing, the project has proven successful. This has made the completion of the project a rewarding experience. Despite encountering complications along the way, all the components in the system work as intended with confirmation from the testing evaluation.

Researching skills have enhanced after researching different technologies, evaluating and choosing technologies based on what would work best for the project idea. Each technology was researched in depth and chosen based on the advantages it would add to the project.

Following an agile development as a methodology improved the efficiency of the development of the project. Having a structured plan of the requirements and weekly logs, helped keep the project on track. Developing the features by following the requirements table forced the developer to deal with issues before moving on, allowing all requirements to be completed in full.

New skills were attained whilst developing this project. A framework was never used before when developing an end to end system. Django and React frameworks taught the developer how to program a backend and frontend separately and in a structured manor. Whilst it was difficult learning these frameworks from scratch, the skills acquired will be helpful when developing in the future. Working with APIs was also a new skill learnt throughout the communicating the frontend with the backend.

6.3. Future Work

As not all recipes contain an image, an AI image could be generated with the ingredients in the recipe. An AI art generator, like Midjourney, could be used to implement this. This would improve the recipe display, allowing recipe data to be uniform.

A recommended recipe section within the saved recipe page for logged in users could be an added feature. A recommendation algorithm such as content-based filtering could be implemented to suggest recipes based on the recipes a user has saved already.

To make the barcode scanning feature more robust, a product detection feature before allowing the user from capturing an image could be implemented. This would prevent images with no barcodes being uploaded to Cloudinary and from being processed within the webserver.

7. Bibliography

[1] Explain that Stuff. (2018). How do barcodes and barcode scanners work? [online] Available at: https://www.explainthatstuff.com/barcodescanners.html.

[2] Postman Blog. (2020). What Is an API and How Does It Work? [online] Available at: https://blog.postman.com/intro-to-apis-what-is-an-api/.

[3] Tesco Real Food. (n.d.). *Recipe Search Engine : What Can I Make With...* [online] Available at: https://realfood.tesco.com/what-can-i-make-with.html.

[4] myfridgefood.com. (n.d.). *MyFridgeFood - Home*. [online] Available at: https://myfridgefood.com/.

[5] www.supercook.com. (n.d.). *Supercook: recipe search by ingredients you have at home*. [online] Available at: https://www.supercook.com/#/desktop.

[6] Dauzon, S., Bendoraitis, A. and Ravindran, A. (2016). Django: Web Development with Python. [online] Google Books. Packt Publishing Ltd. Available at: https://books.google.ie/books?hl=en&lr=&id=vKjWDQAAQBAJ&oi=fnd&pg=PP1&dq=django+python&ots=2nTFzyoZyJ&sig=xVEW_-ZWViGG6IOSYUWrvmxJ6YQ&redir_esc=y#v=onepage&q=django%20python&f=false

[7] www.djangoproject.com. (n.d.). *Getting started with Django | Django*. [online] Available at: https://www.djangoproject.com/start/. [Accessed 12 Oct. 2022]

[8] careerfoundry.com. (n.d.). *The Flask Web Framework: A Beginner's Guide*. [online] Available at: https://careerfoundry.com/en/blog/web-development/what-is-flask/.

[9] GeeksforGeeks. (2021). *Connect Flask to a Database with Flask-SQLAlchemy*. [online] Available at: https://www.geeksforgeeks.org/connect-flask-to-a-database-with-flask-sqlalchemy/.

[10] Mindbowser. (2021). *What Is Ruby on Rails & Why You Should Use it for Your Web Application*. [online] Available at: https://www.mindbowser.com/what-is-ruby-on-rails/.

[11] admin (2021). *Django Vs React : which one is best for web development?* [online] Available at: https://www.javaassignmenthelp.com/blog/django-vs-react/#:~:text=being%20SEO%20friendly.- [Accessed 13 Oct. 2022].

[12] Fylaktopoulos, G., Goumas, G., Skolarikis, M., Sotiropoulos, A. and Maglogiannis, I. (2016). An overview of platforms for cloud based development. *SpringerPlus*, 5(1). Available at: https://link.springer.com/article/10.1186/s40064-016-1688-5

[13] DigitalOcean. (n.d.). *DigitalOcean Documentation*. [online] Available at: https://docs.digitalocean.com/.

[14] Amazon (2015). *Amazon Web Services (AWS) - Cloud Computing Services*. [online] Amazon Web Services, Inc. Available at: https://aws.amazon.com/?nc2=h_lg.

[15] Google (2019). *Cloud Computing Services | Google Cloud*. [online] Google Cloud. Available at: https://cloud.google.com/.

[16] devcenter.heroku.com. (n.d.). Heroku Dev Center. [online] Available at: https://devcenter.heroku.com/.

[17] Talend (n.d.). *What is MySQL? Everything You Need to Know | Talend*. [online] Talend Real-Time Open Source Data Integration Software. Available at: https://www.talend.com/resources/what-is-mysql/.

[18] www.jobsity.com. (n.d.). *5 Reasons Why MySQL Is Still the Go-to Database Management System*. [online] Available at: https://www.jobsity.com/blog/5-reasons-why-mysql-is-still-the-go-to-database-management-system.

[19] PostgreSQL (2019). *PostgreSQL: About*. [online] Postgresql.org. Available at: https://www.postgresql.org/about/.

[20] MongoDB. (n.d.). *Why Use MongoDB And When To Use It?* [online] Available at: https://www.mongodb.com/why-use-mongodb#:~:text=Using%20MongoDB%20can%20provide%20many.

[21] Readthedocs.io. (2011). Pillow — Pillow (PIL Fork) 6.2.1 documentation. [online] Available at: https://pillow.readthedocs.io/en/stable/.

[22] OpenCV (2018). *About*. [online] Opencv.org. Available at: https://opencv.org/about/.

[23] docs.opencv.org. (n.d.). OpenCV: Smoothing Images. [online] Available at: https://docs.opencv.org/4.x/d4/d13/tutorial_py_filtering.html.

[24] docs.opencv.org. (n.d.). OpenCV: Histogram Equalization. [online] Available at: https://docs.opencv.org/3.4/d4/d1b/tutorial_histogram_equalization.html.

[25] www.tutorialspoint.com. (n.d.). OpenCV - Adaptive Threshold. [online] Available at: https://www.tutorialspoint.com/opencv/opencv_adaptive_threshold.htm.

[26] Hudson, L. (n.d.). pyzbar: Read one-dimensional barcodes and QR codes from Python 2 and 3. [online] PyPI. Available at: https://pypi.org/project/pyzbar/.

[27] Cloudinary. (n.d.). Image and Video Upload, Storage, Optimization and CDN. [online] Available at: https://cloudinary.com/.

[28] Edamam.com. (2019). Edamam - Eat better! [online] Available at: https://www.edamam.com/.

[29] platform.fatsecret.com. (n.d.). FatSecret Platform API - Verified Food and Nutrition Data. [online] Available at: https://platform.fatsecret.com/api/.

[30] Tang, E. (2018). Event Management System Using Image Authentication. Available at: https://library-cc.tudublin.ie/search?/XDT228&SORT=D/XDT228&SORT=D&SUBKEY=DT228/1%2C151%2C151%2CB/frameset&FF=XDT228&SORT=D&29%2C29%2C

[31] Corson, C. (2014). FindMyImage –Estimating the location of an image. Available at: https://library-cc.tudublin.ie/search?/Xfind&SORT=DZ/Xfind&SORT=DZ&extended=0&SUBKEY=find/1%2C458%2C458%2CB/frameset&FF=Xfind&SORT=DZ&8%2C8%2C

[32] Synopsys Editorial Team (2019). Top 4 software development methodologies | Synopsys. [online] Software Integrity Blog. Available at: https://www.synopsys.com/blogs/software-security/top-4-software-development-methodologies/.

[33] Anon, (n.d.). Top 7 Software Development Methodologies | Laneways Software. [online] Available at: https://www.laneways.agency/top-7-software-development-methodologies/.

[34] Kumar, G. and Kumar Bhatia, P. (2012). Impact of Agile Methodology on Software Development Process. International Journal of Computer Technology and Electronics Engineering (IJCTEE), [online] 2(4), pp.1–5. Available at: https://www.researchgate.net/profile/Gaurav-Kumar-175/publication/255707851_Impact_of_Agile_Methodology_on_Software_Development_Process/links/00b49520489442e12d000000/Impact-of-Agile-Methodology-on-Software-Development-Process.pdf.

[35] Nishadha (2012). UML Diagram Types | Learn About All 14 Types of UML Diagrams. [online] Creately Blog. Available at: https://creately.com/blog/diagrams/uml-diagram-types-examples/.

[36] Softwaretestinghelp.com. (2017). Types of Software Testing: Different Testing Types with Details. [online] Available at: https://www.softwaretestinghelp.com/types-of-software-testing/.

[37] Forcada Matheu, N. (2005). Life Cycle Document Management System for Construction. pp.200–215. Available at: https://www.tdx.cat/bitstream/handle/10803/6160/10Nfm10de12.pdf?sequence=34.xml

8. Appendix

8.1 Django Models

```python
# RECIPE MODEL
class Recipes(models.Model):
    """Model representing a recipe."""
    id = models.IntegerField(primary_key=True)
    title = models.CharField(max_length=100)
    description = models.CharField(max_length=200, default='description')
    image = models.URLField(max_length=200, default='image')

    def __str__(self):
        """String for representing the Model object."""
        return self.title

#RECIPE DETAILS MODEL
class RecipeDetails(models.Model):
    """Model representing a recipe."""
    id = models.IntegerField(primary_key=True)
    title = models.CharField(max_length=100)
    description = models.CharField(max_length=200, default='description')
    instructions = models.JSONField(default='instructions')
    ingredients = models.JSONField(default='ingredients')
    image = models.URLField(max_length=200, default='image')
    cooking_time_min = models.IntegerField(default=0)
    servings = models.JSONField(default=0)

    def __str__(self):
        """String for representing the Model object."""
        return self.title

# SAVED RECIPE MODEL
class SavedRecipes(models.Model):
    """Model representing a saved recipe."""
    id = models.IntegerField(primary_key=True)
    user = models.CharField(max_length=100)
    recipe_info = models.JSONField(default='recipe_info')

    def __str__(self):
        """String for representing the Model object."""
        return self.id
```

Figure 43 Django Models - models.py

Figure 43 displays the Recipes, RecipeDetails and SavedRecipes models. Recipes model contains id (Integer Field), title and description (Character Fields) and image (URL Field) . These fields store recipe data from the FatSecret API.

RecipeDetails model contains the fields; id (Integer Field), title and description (Character Fields), instructions, ingredients and servings (JSON Fields), image (URL Field) and cooking_time_min (Integer Field). These fields are used to store recipe details from the FatSecret API.

SavedRecipes model contains the id (Integer Field), user (Character Field) and recipe_info (JSON Field) to store the recipes a logged in user has saved.

8.2 Django Serializers

```
# RECIPE SERIALIZER
class RecipesSerializer(serializers.ModelSerializer):
    class Meta:
        model = Recipes
        fields = ('id', 'title', 'description', 'image',)


# RECIPE DETAILS SERIALIZER
class RecipeDetailsSerializer(serializers.ModelSerializer):
    class Meta:
        model = RecipeDetails
        fields = ('id', 'title', 'description', 'instructions',
                  'ingredients', 'image', 'cooking_time_min', 'servings',)

# SAVED RECIPE SERIALIZER
class SavedRecipeSerializer(serializers.ModelSerializer):
    class Meta:
        model = SavedRecipes
        fields = ('id', 'user', 'recipe_info',)
```

Figure 44 Django Serializers - serializers.py

As seen in Figure 44, RecipesSerializer, RecipeDetailsSerializer and SavedRecipeSerializer inherit the fields of the Recipes, RecipeDetails and SavedRecipe models.

8.3 Django Views

```python
# INGREDIENT DETAIL PUT/DELETE
@api_view(['PUT', 'DELETE'])
def ingredient_detail(request, pk):
    try:
        ingredient = Ingredient.objects.get(pk=pk)
    except Ingredient.DoesNotExist:
        return Response(status=status.HTTP_404_NOT_FOUND)

    if request.method == 'PUT':
        serializer = IngredientSerializer(ingredient, data=request.data)
        if serializer.is_valid():
            serializer.save()
            return Response(serializer.data)
        return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)

    elif request.method == 'DELETE':
        ingredient.delete()
        return Response(status=status.HTTP_204_NO_CONTENT)
```

Figure 45 Ingredient Detail - views.py

When the user wishes to change ingredient details or delete an ingredient, the ingredient object is retrieved using the ingredient ID. If the request is PUT, the ingredient is serialized and the ingredient is updated with the request data and a positive response is returned. If the request method is DELETE the ingredient object is deleted and the status is returned in the response.

```python
# RECIPE LIST GET
class recipesView(APIView):
    def get(self, request, *args):
        username = request.GET.get('user')
        ingredients = Ingredient.objects.filter(user=username)

        serializer = IngredientSerializer(ingredients, many=True)
        myIngredients = ""

        for ingredient in serializer.data:
            myIngredients += ingredient['productName'] + ","

        myrecipes = SearchRecipes.search(myIngredients)

        recipes = Recipes.objects.all()
        recipes.delete()
        print('Recipes deleted successfully')

        serializer = RecipesSerializer(data=myrecipes, many=True)
        if serializer.is_valid():
            serializer.save()
            print('Recipes saved successfully')
        else:
            print('Recipes not saved')
            print(serializer.errors)

        recipes = Recipes.objects.all()
        serializer = RecipesSerializer(recipes, many=True)
        return Response(serializer.data)
```

Figure 46 Recipes View - views.py

When the user wishes to view a list of recipes that contains the ingredients entered, recipesView is used with the GET request. The user logged in is retrieved and the ingredient objects are filtered by the user. The ingredients are then serialized and the productName for each ingredient is added to a list with a comma separating them. The list of ingredients is passed to the search function within the SearchRecipes.py which is discussed in section 4.3.5 Recipe Search . All the recipe objects are retrieved and deleted before the recipes returned from search function are serialized and saved to the database. All the new recipes within the database are retrieved from the database and serialized before returning to the frontend.