CFG Nanodegree Project Document

Easy Eats: The Recipe Generator

CFG Nanodegree Cohort **Autmn/Winter 2021-22**

Deeqa Mohamed, Maya-Vani Mudgal, Mary-Dawn Onwosorom, Suad Mohamed, Amran Muse

Table of Contents

NUTRITION PROJECT

1 INTRODUCTION

1.1 Aim

This project aims to build a nutrition-based program that allows users to input their dietary preferences, nutritional restrictions (i.e. gluten-free, vegan, vegetarian, pork-free etc) and primary ingredients they would like to incorporate; and return suitable recipes that fit their criteria.

1.2 Objective

Cooking meals is an essential task that people must do with busy schedules with little to no time to prepare their meals, hence the success of food delivery services such as UberEATS and Deliveroo.

The hardest step in anything is the first, this program aims to solve that; what to cook? It conveniently considers the primary ingredients that you have, your dietary requirements and preferences, which will inevitably reduce the time spent scouring for recipes that suit your needs. It solves the problem of planning meals, which is ideal for people with a busy schedule.

This program aims to help people save money on takeaways and food delivery services by providing them with suitable options they can cook. By using ingredients already in the fridge, this reduces the frequency of takeaways thus reducing waste such packaging. This is better for the environment, as less energy is used for transport which in turn lowers CO2 emissions and helps to improve overall sustainability.

It is also useful for suggesting recipes to people who cook often and need new ideas, in addition to special occasions; when at a loss for what to cook or if their guests have special dietary requirements. Essentially this program simplifies the process of cooking meals by saving the users time and consequently money.

2 BACKGROUND

The features of this website have been carefully curated to ensure the best user experience possible, this will be achieved through filter powered search. The search will allow you to input one or more ingredients of your choice as well as the ability to exclude ingredients from the recipe. To further tailor the recipe to the user's health and dietary requirements, users will also be able to input allergies for example, lactose intolerance and dietary preferences such as vegan, pescatarian, etc. Once the user selections have been made, a choice of 20 recipes will be presented along with cards that contain the nutritional values and calories of the recipe which will allow the user to select a recipe that aligns with their requirements. Finally, a choice of top five recipes will be returned from the 20 recipes, further simplifying the user's decision making.

3 SPECIFICATION AND DESIGN

3.1 Requirements

The functional and non-functional requirements of this project were analysed and documented in the backlog during the initial brainstorm and refined throughout the project.

3.1.1 Functional requirements

The functional requirements were as follows:

- For the user to be able to input key ingredients they would like the recipe to incorporate and exclude specific ingredients they do not want the recipe to include.
- For the system to take into consideration their allergies, for example lactose intolerance and return recipes that are suitable for that user.
- For the system to provide a selection of dietary preferences from a drop-down menu, for example, veganism and return recipes that are suitable for that user.
- For the system to return a selection of recipes for the user to choose from.
- For the user to be able to view the nutritional values of the recipes.
- For the system to display how long the recipe will take.
- A button that generates recipes containing the key ingredients.
- A suitable API with recipes and filtering capabilities, which Edamam proved to be the best option.
- A user interface that implements HTML to build the structure of the website, CSS to bring that structure to life and Flask as the framework to connect the back-end python code.
- A system that allows users to rate recipes out of 5 and to also see how many other users rated the recipe.
- A system that allows users to leave a review

3.1.2 Non-functional requirements

- **Usability**
  - The website was designed using Figma with an aim to provide an aesthetically pleasing interface that appeals to users with an efficient process to achieve the desired recipes. Figma wireframes can be seen in the appendix.
  - This website is designed to be used multiple times daily to cook meals therefore in terms of reliability and availability, it must run 24/7/365.
- **Documentation**
  - Well organised, readable python code will allow for modifications and changes throughout the project.
  - Well documented software development processes such as agile method using Trello board, etc.
- **Testability**
  - Thorough unit testing
  - Tailored user acceptance testing
- **Security**
  - The user does not need to input personal information about themselves which reduces security measures that need to be taken.

3.1.3 User requirements

A key stage of creating and assigning tasks to build the program required important user personas that would use this website. Once the user personas were created as shown in table 1 and 2, their user stories were written to establish what their needs are and whether they aligned with the website's functionality. The user requirements were recorded in the manner of user stories, in the following format: 'As <a user>, I want <goal>, for this <reason>'. This step was essential to determining the contents of the backlog and the subsequently assigned tasks. Listed below are a few examples of the user stories:

- As a user I want to be able to see how long a recipe takes before I commit.
- As a user I want to be able to see the nutritional values of the meal.

User Personas

Name:	**Maddison Green: Student**
Bio:	Maddison is an 18-year-old Gen Z student, living away from home for the first time. Outside of her studies she frequents freshers events, she's social and has many commitments to various social and sport clubs at university.
Needs:	● Unable to recreate complicated Tik Tok recipes. She wants to save money on UberEats expenses whilst also cooking meals that she enjoys. ● She likes tasty and fun meals that are easy to make. ● She has a severe nut allergy. ● She has recently adopted a vegan lifestyle and needs healthy and tasty options.
Dislikes:	● Doing one thing for too long (loses interest quickly). ● Rude people

Table 1: Maddison User Persona

Name:	**Scott Thompson: Gym Enthusiast**
Bio:	Tristan is a 31-year-old avid gym goer with a busy schedule as a Marketing Executive. In his free time, he enjoys spending time with his partner and taking his dog for a walk.
Needs:	● Between his busy work schedule and gym going, he would like to prep meals that do not consume his limited time. ● He would like to make meals that include his daily nutritional goals (macros, etc). ● He often cooks for his pescatarian partner and wants to be able to make tasty and creative meals for them both.
Dislikes:	● Newbies at the gym ● Frivolous activities (likes constant productivity).

Table 2: Scott User Persona

Name:	**Karen Hopkins**
Bio:	Karen is a 45-year-old housewife and mum of three, she her daily schedule involves school runs, preparing her children for a day at school with breakfast and packed lunches. She has recently been diagnosed with irritable bowel syndrome (ibs).
Needs:	● She is now dairy and gluten free therefore is looking for new recipes that avoid those ingredients. ● She would like her children to be healthier whilst still enjoying tasty meals

	• She would like to save money on cookbooks
Dislikes:	• People that honk • People that use family parking without children

Table 3: Karen User Persona

3.2 Architecture

For our project, we used the client-server architecture. A client inputs their requests on the website and that request is taken from the server. The server will respond to the request and return the information on a jinja2 html page to the client. If the request sent from the client is stored in the database, the server will send a request to the database and the database will respond returning the data on a jinja2 html template.

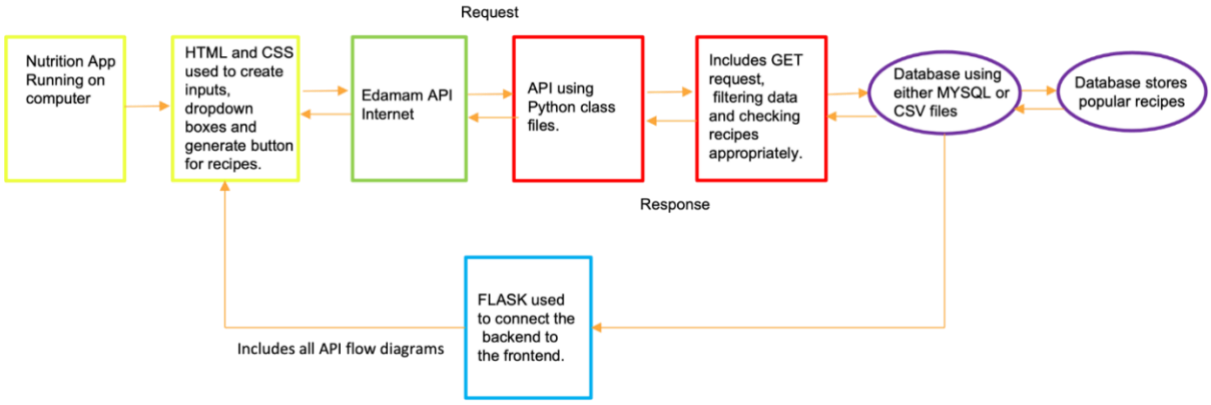

Figure 1: Graphical representation of the steps used to create the program in sequential order

3.3 Project structure

In the CFG-NUTRITION folder, we placed the GitHub repository, cfg-nutrition-project, and virtual environment. In the virtual environment we installed all the required libraries such as; flask, requests, pandas, unittest, and jinja2. Within the cfg-nutrition-project file, we created two folders: src and terminal_project. In the terminal_project folder, all the backend python files have been placed in there including a database which stores user's favourite recipes. The purpose of this folder is to show that we implemented a database in our project. However, it is in a separate folder to the src (website) folder as we have not been able to connect it to our front-end due to time restrictions. This is to be further discussed under the issues section. Initially, we wanted to. In the src folder we added an empty __init__.py file to mark directories on disk as Python package directories. We initially had issues importing files within each other but adding an empty __init__.py in every folder allowed us to call different files. After creating a __init__.py file, a __pycache__ folder was also created in the src folder. Within the src folder we also have an app folder. This folder contains all the files required for the website to run. An __init__.py file was added to the app folder and a __pycache__ was created by default as a result. The run.py is the Flask app file which connects the front-end to the backend. CALL_API.py, Recipe_class.py, and

main.py are python files which contain the recipes collected from the api. Test.py contains the tests for the server and api. Within the app folder, static and template folders are added. The static folder contains style.css and images. In the templates folder, we have all the jinja html templates required.

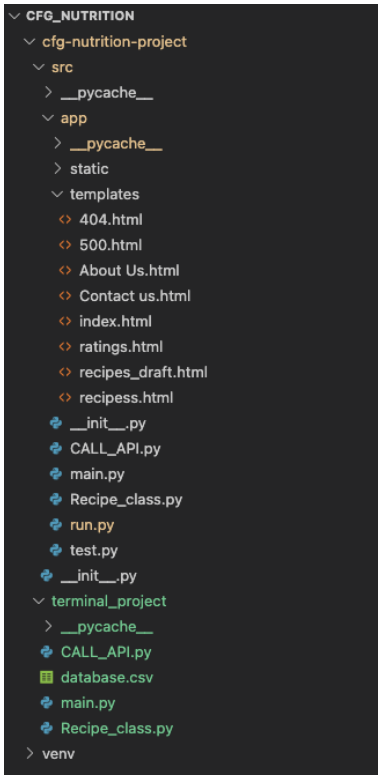

Figure 2: Project sub-directories

3.4 Database design

The design of this database was built on the concept that recipes should be returned for a particular ingredient, health, diet and allergy choice depending on the user's input which are directly appended to the database. The database is mocked using a CSV file which stores the recipe names, reviews and ratings for each user. We have provided the ability to add new user information to the database and overwrite existing user information for each recipe with the ask user function. We are also able to clear the database using the clear database function. By using the data frame initialised in the Call API class using the Pandas library this provided us with the ability to read and write data into the CSV file so we were able to filter out and use the columns we wanted to use. Through the use of the explore database function this will allow the user to view all the ratings and reviews for a particular recipe. By appending user information to the database all the ratings, reviews and average rating for a particular recipe can be calculated using the get average function and returned to the screen. The average data frame holds the following information for the recipe name, review and ratings and can be presented to the user when asked.

4 IMPLEMENTATION AND EXECUTION

4.1 Development method

This project utilises Scrum framework, an agile methodology to implement and execute this project. The initial scrum meeting established online tools that would assist this development approach and the organisation of the team which was crucial to reaching goals.

Firstly, a slack workspace was created for team members to communicate, send useful links, zoom meeting links etc. These zoom meetings were held daily in lieu of daily scrum meetings, lasting an average of 30 minutes, they reduced misunderstandings and allowed the team to come to conclusions promptly.

A notion page was created and shared with all the team members as a way of documenting meeting notes and creating easy access to additional documents. In addition, a scrum board was shared between all the team members on Trello; with information such the backlog, requirements and to-do which was regularly updated throughout the project and during sprint-planning meetings. This allowed us to divide the tasks and requirements between the team members and keep track of which tasks have been completed. Wireframes were also created on Figma to envision how the user interface would look.

4.1.1 Edamam api

The Edamam API proved to be a very useful tool as it allowed many of the program's filtering capabilities which saved ample time and code. The Edamam API allows certain ingredients to be excluded, specified ingredients to be incorporated, it shows the nutritional values of the recipes and it filters recipes based on allergies and dietary requirements. However, the Edamam API also provided a lot of information that was not useful to us, so the challenge was writing python code that only extracted the information we needed.

4.1.2 Recipe criteria

As aforementioned the website returns recipes based on the users input, this was made possible through OOP principles; by creating a recipe class although initial code was written in the form of a list. The recipe criteria consisted of four parts; what the ingredients the user wishes to exclude, what ingredients they would like to incorporate, their allergies and their dietary preferences (e.g. vegan).

4.1.3 Database

The database created in the terminal project collects ratings and allows users to leave a review using a drop-down menu. This data is then collected and presented to the user as an average figure of how many users have previously rated this recipe.

4.1.4 Flask, HTML, CSS

Bootstrap template files were used for all frontend pages. Bootstrap contains html and css, however additional html and css was also incorporated. A css stylesheet was placed in the <head> tag to allow the css component in bootstrap to load. A javascript file was also added just before the end of the </body> tag to allow the pages to be responsive.
Classes were used in the bootstrap html files to achieve different effects, however a css style file has been used to override some of these classes.

In Flask, functions for the server roots were created and jinja2 templates were used to print the information on the webpages. Error handler pages for 404 and 500 errors were created using Flask and jinja2 templates.

4.1.5 Team member roles

Each team member was assigned as scrum master each week so as to give everybody a fair chance at managing. The scrum master would host the daily zoom meetings and manage the projects in terms of reaching deadlines and managing the scrum board on Trello. Initially the team members were assigned as follows:

Amran: API, Testing

Deeqa: Front-end, Testing

Mary-Dawn: Database

Maya-Vani: API, Testing

Suad: Front-end, Design

These roles were decided after a discussion regarding the self-evaluated strengths and weaknesses of each team member. However, these roles overlapped throughout the project when we faced challenges during the implementation process.

4.2 Implementation procedure

4.2.1 Problems and solutions

One challenge we faced was changing the database design of the project towards the end. This last-minute change meant we were unable to connect our database to the flask app and therefore use the rating system on the website. Instead, we made a separate folder for the terminal project to show that the database is fully functioning and working. To improve, next time we would spend more time brainstorming database design ideas to prevent significant changes down the line.

When writing the tests in vscodes, using the run button to run the file was throwing errors because vscodes couldn't pick up the tests.py directory. Specifying the path root in the terminal solved this issue.

4.2.2 Adjustments

One of the changes made was the initial use of the CSV database because the data that was being collected was not valuable nor was it an efficient way to store data. Initially the team had planned to store the top 5 recipes from the 20 recipes returned. However, when we evaluated what decides that a recipe belongs in the top 5, we understood that there was no significant reason. So instead the database would be used to collect data from users in the manner of ratings. Users could instead rate recipes out of 5 and the ratings would be

displayed as well as how many customers in total had rated the recipe. However, due to the challenge of connecting the database to the flask app, the database was removed as a whole.

4.2.3 Accomplishments

One of the accomplishments included learning how the Edamam API works and using it to create features on the website. Another accomplishment was learning to use flask and jinja2 templates to connect the backend to the server.

5 TESTING AND EVALUATION

5.1 Unit testing

For each page on the website, a unit test was written to check the status code. A status code for the API URL was also tested. Another test was also written to check if the nutrients were being returned from the API.

5.2 User Acceptance Testing

Once the website was developed and the unit tests were executed, the next step involved unit acceptance testing as an end user. This method required the team members to use the website as each of the user personas and evaluate whether the website runs as it should and if it fits the user requirements.

5.3 System Limitations

One of the user requirements proposed that they be able to see how long each recipe takes, however this option was not successfully implemented using the API, this is certainly a feature that can be developed in the future. In addition, the users have the ability to leave a review, however these reviews are a selection from a drop-down menu which means that users are unable to express specific reviews about a recipe. Another future feature would be to have a comment section with limited characters so that users can leave concise reviews in their own words.

5.4 Evaluation

The UI of the Easy Eats website was evaluated using the Nielsen Heuristic Model which uses ten principles for interaction design. These principles were listed in Table 4 shown below and through using the findings from the user acceptance testing. Each principle was marked a pass or fail with explanations and the failed principles also give a suggestion for future updates that would allow it to pass the criteria.

Criteria	Pass/Fail	Explanation
Visibility of system status	Fail	The user is not aware of what is happening on the page other than their expectations from their actions. Future update: an icon that indicates their request is being returned.

Match between system and the real world	Pass	The sequence of steps follow a natural mapping. The recipes page includes and image of the meal.
User control and freedom	Pass	Clearly indicated button that will take the user to the desired page as well as an exit option.
Consistency and standards	Pass	Layout and design of the website is consistent with the norm. Navigation bar can be found where expected. Future update: The cards that show the recipes are not consistently four cards after the first row, so to make that consistent.
Error prevention	Pass	Error handler pages were created to handle any 404 and 500 errors. This navigates users to the homepage. The try and except method was used in python files and Flask to pick up any errors.
Recognition rather than recall	Pass	The users' selections are visible and navigation buttons remain throughout the process. Future update: when leaving a review/rating suggest recipe as user is typing it rather than typing full recipe name.
Flexibility and efficiency of use	Fail	There is no shortcut or alternative method of retrieving a previous recipe, users must go through the steps. Future update: for users to be able to create an account and have a record of their favourite or recently searched recipe.
Aesthetic and minimalist design	Pass	Every page only contains essential and relevant information. Uses minimal,

		aesthetic background designs, logo and icons.
Help users recognise, diagnose and recover from errors	Fail	There is a 500 internal server error message that indicates to users that they haven't selected a health or dietary requirement. Future update: An error message in plain language not error codes to tell users what went wrong. Also the option to select 'none'.
Help and documentation	Pass	Help icons next to each step that users can hover over that state 'Please fill in this field'. Future update: users must type 'none' if they don't wish to exclude any ingredients so a prompt that tells them to select 'none'.

Table 4: Nielsen Heuristic Model

6 CONCLUSION

Overall, the aim of the project was achieved by building a website that generates recipes based on the user's input. The criteria of the users' input being one or more key ingredients, the exclusion of specific ingredients, their health and dietary preferences and their allergies. The terminal project allows users to rate recipes, leave reviews and see the average rating of other users. The website operates using an Edamam API and the front-end was created using bootstrap template files with the incorporation of HTML, CSS and Javascript and finally the Flask framework. Easy Eats is a useful website to everyone from people that are cooking for the first time to skilled cooks who are looking for new and exciting recipes. This project eliminates the laborious task of searching for suitable recipes daily.

7 ACKNOWLEDGEMENTS

We would like to thank Code First Girls for facilitating this project and Fatma, Ali and Anete for their guidance, support and lessons throughout the Nanodegree that allowed us to gain the knowledge to carry out this project. Finally thank you to all the members of this team, the meetings and multiple perspectives were vital to achieving the final outcome.