Technical Documentation

1. Introduction

Name : Kitchen Assistant
Created by: Amber King, Ruth Muleya, Simone Proctor, Kam Mounsey

This technical documentation for Group 2's final project details the design and implementation of a recipe search website.

1.1 Roadmap Of Report

1.2 Aims and Objectives

The overall aim is to create a locally hosted cooking support website that references an SQL database and the API EDAMAM (*https://developer.edamam.com/recipe-demo/.)*. It includes: recipe suggester, recipe links, an SQL DB for ingredient substitution, a converter function for units (metric<>imperial) and an individual recipe unit conversion.

2. Background

The user must input at least one ingredient to pull up recipes including this ingredient. When searching for substitutions a user must search by a food name. The unit conversion can convert the following units by inputting positive numerical data: Cups, Tbsp, Tsp, G, Oz, Fl oz and Ml.

3. Specifications and Design

3.1 Functional Requirements

This section will outline the functional requirements of our system.

3.1.1 Flask website with python search

Functionality: a website using Flask that integrates a search feature that uses Flask-request to send user inputted data to the python program. The website has a search bar to query data and display the results clearly.

Use Case: A user visits the Flask website, enters a search query (an ingredient) into the search bar, and views the returned results on the same page.

User story: As a home cook I always have leftover ingredients like shallots in my fridge,I want to search for recipes containing shallots so that I can use them before they go to waste.

Inputs: Search query (text)

Outputs: Recipe(s)

3.1.2 API search results displayed clearly on the website

Functionality: Format and display the search results obtained from an API in a user-friendly manner on the website.

Use Case: After a user uses the API to perform a search, the system processes the results and presents the results clearly so that the user can utilise the website efficiently.

Input: API Response Data (json)

Output: HTML/CSS-rendered Search Results

3.1.3 Conversion function

Functionality: Implement a function to convert data from one format to another (e.g., from cups to ml).

Use Case: A user requests data in a different format the system processes the conversion and provides the result.

User story: As a UK home cook, I want to easily convert recipe measurements from units like cups to units like millilitres/grams so that I can accurately follow recipes using units I'm familiar with.

Input: Original data, Desired format/unit

Output: Converted data

3.1.4 SQL DB Search functionality

Functionality: Enable the website to perform search queries directly against a SQL database, retrieving relevant data based on user input.

Use Case: A user enters an ingredient into the search bar, and the system executes an SQL query against the database to find matching records, a matching record indicates that there is an alternative ingredient that can be used as a substitute.

User story: As a user with dietary restrictions, I want to find appropriate and common substitutions for certain ingredients in a recipe so that I can make the recipe suitable for my diet.

Input: Search Query (text), SQL Query Parameters (Mapped from user input)

Output: Search Results (Rows of data from the database)

3.1.5 SQL results displayed clearly on website

Functionality: Display the search results retrieved from the SQL database in a clear and organised manner on the website.

Use Case: After a user performs a search query against the SQL database, the system presents the results in an easily navigable and readable format.

Input:SQL Query results

Output: HTML/CSS-rendered Search Results

3.2 Non Functional Requirements

This section will outline the non functional requirements of the website.

Usability: The system is designed simply with the user in mind. It should be intuitive to use. There is an ongoing joke with online recipes, that you must first read the creator's life story before reaching the recipe. We wanted our website to be purely functional.

We chose not to use a login system, we wanted to cater to the spontaneity of starting cooking, realising there is a missing ingredient and making adjustments on the fly.

Compatibility. The system would require the user to have a functioning browser, access to the API and the SQL database.

Size : Initially, if we were to go live with the project, we would anticipate several hundred users but would expect this to grow monthly and would want the system to cope with much higher traffic.

Scalability : The current system is designed with a website at the front end, utilising an API and SQL database. Flask's lightweight nature allows for easy scaling horizontally by adding more instances of the application.

However, currently we only take in american recipes which can be accommodated by the regex standardisation - this would be an additional challenge if we scaled up the recipe data source.

With an expected growth in variation of users, there may be more units required for recipes.

Local storage is the main data save for recipes, if a user does not clear their cache, the size of the data would continue to grow every query - therefore another method of storage is needed.

Maintainability: The separation of concerns among components (web application, database, and external APIs) simplifies maintenance and updates. Each component can in theory be modified or upgraded with minimal impact on the others. However, even though the system is easy to maintain, it could be hard to manage input errors, which may be not accounted for with our exceptions.

3.3 Future Non Functional Considerations

3.3.1. Log In System

We plan to implement a user log-in system to provide personalised results and features. This would necessitate enhanced security measures and robust data protection to handle user credentials and sensitive information. We would therefore prioritise compliance with relevant regulations and ensure the system is easy to maintain, facilitating updates and addressing issues like forgotten passwords.

3.3.2. Enhanced Features

Future enhancements include shopping list generation, an expanded substitution database, and advanced search functionalities for dietary needs. The addition of user areas will enable personalised recipe suggestions and improved save functions. These developments will require careful attention to system maintainability and a seamless user experience.

3.3.3. Partnering with existing companies/Apps

Partnering with existing companies, such as gusto, would allow us to utilise not only their customer base but their functionalities, allowing for greater growth and scalability.

3.4 Design and Architecture

3.4.1 Overview

Our system is a website built using Flask, a lightweight and flexible Python web framework. It integrates with both a database and external API.

3.4.2 Key Components

Web Application

- Framework : Flask- Jinja(html templating)
- Language : Python/Javascript/Css
- Role : Serves as the core of the application, managing HTTP requests and responses, rendering web pages, and handling user interactions.

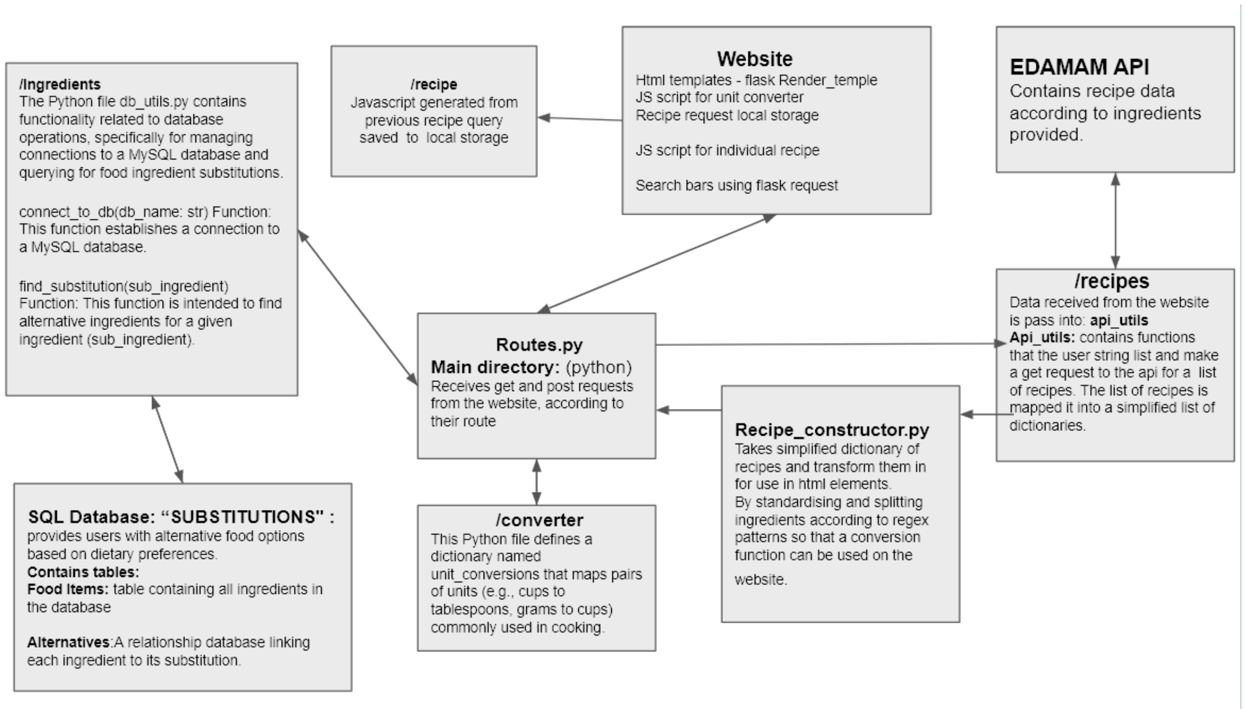
Database

- Type : MySQL
- Role : Stores and manages substitution data

External API

- Purpose : provides recipe information, integral to our website.
- Integration : The system communicates with these APIs using HTTP requests, processing the responses to enhance or extend its functionality.

3.4.3 Architectural Diagram



/Ingredients
The Python file db_utils.py contains functionality related to database operations, specifically for managing connections to a MySQL database and querying for food ingredient substitutions.
connect_to_db(db_name: str) Function: This function establishes a connection to a MySQL database.
find_substitution(sub_ingredient) Function: This function is intended to find alternative ingredients for a given ingredient (sub_ingredient).
/recipe
Javascript generated from previous recipe query saved to local storage
Website
Html templates - flask Render_temple
JS script for unit converter
Recipe request local storage
JS script for individual recipe
Search bars using flask request
EDAMAM API
Contains recipe data according to ingredients provided.
/recipes
Data received from the website is pass into: api_utils
Api_utils: contains functions that the user string list and make a get request to the api for a list of recipes. The list of recipes is mapped it into a simplified list of dictionaries.
Routes.py
Main directory: (python)
Receives get and post requests from the website, according to their route
Recipe_constructor.py
Takes simplified dictionary of recipes and transform them in for use in html elements.
By standardising and splitting ingredients according to regex patterns so that a conversion function can be used on the website.
/converter
This Python file defines a dictionary named unit_conversions that maps pairs of units (e.g., cups to tablespoons, grams to cups) commonly used in cooking.
SQL Database: "SUBSTITUTIONS" :
provides users with alternative food options based on dietary preferences.
Contains tables:
Food Items: table containing all ingredients in the database
Alternatives:A relationship database linking each ingredient to its substitution.

4. Implementation and Execution

This section will outline the implementation and execution of our project, with a focus on development approach.

4.1.Development Approach

We took an agile approach to our project. This feedback driven approach allowed us to communicate openly and effectively whilst continuously evaluating our needs and goals for the project.

4.2 Team Member Roles

Simone : Python, Web templating, JS
Ruth: Python (API), liaising with Programmes Team
Amber: Python (unit conversion), CSS
Kam : SQL DB, documentation

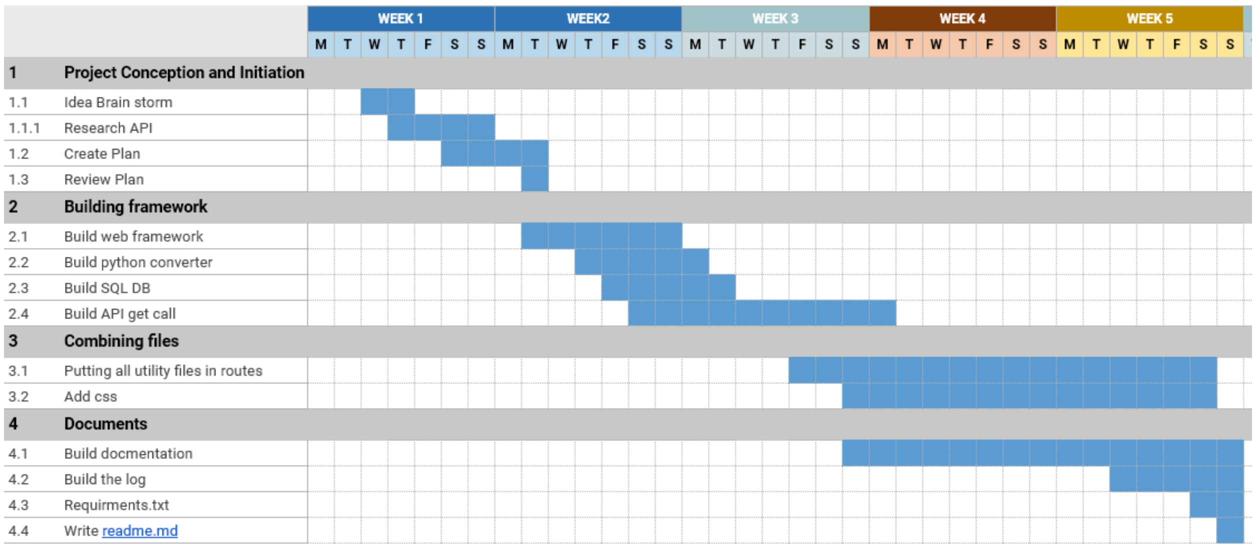
4.3 Tools and Libraries

This section details the tools and libraries utilised to build our project. We used Python, SQL, Flask, EDAMAM API, CSS, My.SQL, Iter Tools3, Jinja, Javascript, Regex and os.

4.4 Agile Development

As stated previously, we took an agile approach to our project. Below is an outline of how we implemented an agile approach.

Gantt Chart:



		WEEK 1							WEEK2							WEEK 3							WEEK 4							WEEK 5						
		M	T	W	T	F	S	S	M	T	W	T	F	S	S	M	T	W	T	F	S	S	M	T	W	T	F	S	S	M	T	W	T	F	S	S
1	**Project Conception and Initiation**																																			
1.1	Idea Brain storm			🟦	🟦																															
1.1.1	Research API				🟦	🟦	🟦	🟦																												
1.2	Create Plan						🟦	🟦	🟦	🟦																										
1.3	Review Plan									🟦																										
2	**Building framework**																																			
2.1	Build web framework									🟦	🟦	🟦	🟦	🟦	🟦																					
2.2	Build python converter											🟦	🟦	🟦	🟦	🟦																				
2.3	Build SQL DB												🟦	🟦	🟦	🟦	🟦																			
2.4	Build API get call													🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦													
3	**Combining files**																																			
3.1	Putting all utility files in routes																		🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	
3.2	Add css																				🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	
4	**Documents**																																			
4.1	Build docmentation																				🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦
4.2	Build the log																														🟦	🟦	🟦	🟦	🟦
4.3	Requirments.txt																																	🟦	🟦
4.4	Write readme.md																																		🟦

- Communication and collaboration
- Asana
- Less focus on heavy documentation
- Transparency on what is and isn't working
- Feedback driven development
- Testing
- Iterative development
- Refactoring

4.5 Achievements

The project overall signifies a huge achievement for all involved. This section summarises a few key achievements that we feel stand out.

- Utilising and collating a range of skills from across the course. Further deepening our understanding and knowledge of key topics.
- Working with github and git as a team, representing real life situations.

- Successful project management and utilising tools such as asana. Identifying strengths (and weaknesses) allowed efficient work.
- Improved knowledge of additional tools such as Regex and standardisation based on input/output.

4.6 Challenges

Overall the project represented many challenges and tested our knowledge and abilities. Some key challenges that would be important to highlight are listed below.

- Researching new ideas and implementing frameworks that were initially foreign to us.
- Troubleshooting difficulties, for example when using git and github
- Effective communication issues, particularly with an absent member, making our team a group of 4.
- Fixing errors and exception handling
- Resolving merge conflicts

5. Testing and Evaluation

This section will outline our testing strategies and evaluate our system progress so far.

5.1 Testing strategies

User 1: *As a UK home cook, I want to easily convert recipe measurements from units like cups to units like millilitres/grams so that I can accurately follow recipes using units I'm familiar with.* Using User 1 as a test case, starting from the homepage and navigating to the recipes tab, clicking on a recipe and being able to convert units of the recipe directing into another, fulfils this.

Unit testing for Recipe class ingredient standardisation static method. The purpose of the unit tests are to see if all ingredients are correctly standardised and mapped using regex for web templates. Tests include unicode characters, additional units and units that do not follow naming conventions in the converter function.

Unit testing for the converter function in particular included testing a conversion from the same units, different units, zero cases and negative cases as well as an invalid conversion to ensure we get the desired output

Unit testing for the API included testing to see if we got the desired output when inputting a valid ingredient and trying again with invalid ingredients (non-ingredients) and zero cases. As well as that, ensuring that a json, that mirrored the input given, was returned in the instance of valid.

5.2 System Limitations

- Lack of advanced searching and filtering
- Limited ingredient search scope
- Database and api dependency
  - Cannot control API maintenance times
- Basic unit conversion support

    - Limited to keys
- Non personalised user experience
- Scalability challenges
- User interface simplicity
    - Limited error response
- Error handling and user input validation
- Cannot account for all recipe ingredient formats
    - Dependent on building regex patterns
- Limited recipe data scope
- Data storage limited to user computer memory

6. Conclusion

Our project culminated in a recipe search website that integrates the EDAMAM API and SQL database. The site allows users to search for recipes based on ingredients, find substitutions and convert measurement units between metric and imperial. Built using Flask, the website offers a straightforward and efficient user experience.

Key achievements include the successful implementation of core features and the use of an agile development approach, which enabled continuous feedback and iterative improvements. While the project meets its primary objectives, it has limitations such as lack of advanced searching and filtering as well as a narrow search scope.

These limitations offer clear pathways for future enhancements, such as more advanced search capabilities and improved scalability. Overall, the project demonstrates strong teamwork and technical integration, providing a robust platform for further development.