CS3354 Software Engineering Final
Project Deliverable 2

Recipe Generator

Group #4

Rhea Aemireddy, Shannon Carter, Jay Chae, Roslyn Collings, Adair Gonzalez,

Hafa Kazi, Veom Nemade, Ayush Velhal

1. Deliverable 1

2. Project Task Delegation

Total Tasks

Deliverable # 1 [All Group tasks]

- **UX/UI Designing and Prototyping:** Develop wireframes on Figma and create a working prototype of the recipe generator.
    - Pages to develop:
        - Landing Page (with navbar: Saved Recipes, Start a Recipe)
            - Assigned: Shannon Carter rhea.a.reddy@gmail.com
        - Recipe Generator Flow:
            - Enter Ingredients (with submission button)
            - Resulting recipe
            - Assigned: ayush.velhal@gmail.com Adair Gonzalez
        - Saved Recipes:
            - Categories:
                - Appetizers
                - Breakfast
                - Lunch
                - Dinner
                - Dessert
                - Drinks
                - Snacks
            - Assigned: cotlqlc21@gmail.com Roslyn Collings hafakazi@gmail.com
- **Environment Setup:** Getting the appropriate technologies installed on everyone's machine. Connecting this project to a repository in GitHub.
    - Technologies Needed:
        - VSCode
        - Git/Github
        - Node.js
        - React.js
        - Flask
    - Packages we may use for Front-End:
        - TailwindCSS
        - Framer Motion
    - Packages we may use for Back-End:
        - FastAPI or Flask
        - Axios
    - Assigned: rhea.a.reddy@gmail.com Shannon Carter hafakazi@gmail.com cotlqlc21@gmail.com ayush.velhal@gmail.com veom2004@gmail.com axg230107@utdallas.edu Roslyn Collings

5.2. Deliverable # 2 [All Group tasks]

- **Implementing Front-End Pages**
    - Landing Page

        - Assigned: Shannon Carter rhea.a.reddy@gmail.com
    - Recipe Generator Flow:
        - Assigned: Adair Gonzalez ayush.velhal@gmail.com
    - Saved Recipes:
        - Assigned: cotlqlc21@gmail.com Roslyn Collings hafakazi@gmail.com
- **Implementing Backend Schema**
    - Assigned: veom2004@gmail.com ayush.velhal@gmail.com
- **Connecting Front-End pages to Backend with API Calls**
    - Assigned: Adair Gonzalez
- **Feature Testing**
    - Navbar functionality
        - Assigned: Shannon Carter
    - Recipe Generation:
        - Assigned: Roslyn Collings
    - Saved Recipes:
        - Assigned: cotlqlc21@gmail.com hafakazi@gmail.com

2. OUR REPOSITORY URL LINK:

https://github.com/254176/recipe-generator

2.1 Figma page design Link

https://www.figma.com/proto/07YaUxtu4BKF3OnapjvXzC/CS-3345%3A-Recipe-Generator?node-id=6-2&t=PQqDxRwvKj9Xod9q-1&scaling=scale-down&content-scaling=fixed

3. Delegation of tasks:

Hafa Kazi:	Outlining Software Process Model, Design “Saved Recipes” page, Set up environment
Shannon Carter:	Project Leader, Fill out delegation of tasks, Create and Share Figma Design File, Design and Prototype “Landing” Page, create Github Repository, Set up environment, Applying Architecture
Rhea Aemireddy:	Final Project Draft Description, Design “Landing” Page, Set up environment
Rosyln Collings :	Document Submission, Design “Saved Recipes” Page, Set up environment
Jay Chae:	Design “Saved Recipes” Page, Set up environment
Veom Nemade:	Creating Sequence Diagrams for Use Cases, Configure MongoDB/PostgreSQL, ensure all endpoints are functional
Adair Gonzalez:	Handling Software Requirements both functional and non-functional, design “Recipe Generator” Flow, planning for frontend and backend connections
Ayush Velhal:	Design “Recipe Generator” Flow, Creating Use Case Diagrams, Designing Class Diagrams, Document Revision

4. Software Process Model

We will be using an Agile methodology due to its adaptability and effectiveness on a small scale. We will be using the Scrum framework. This will allow us to be flexible, get continuous feedback, and make iterative development. Our requirements will evolve as we test, since it is an AI-based recipe generator, and we need to refine the model. Scrum will allow us to break the project into manageable sprints for regular progress and adapt to the feedback, and have good collaboration between team members. This will help us deliver a functional product faster because of continuous improvements with accuracy and usability based on real-world testing.

Database:

Architectural Design: We are using MongoDB for our Recipe Generator project because it offers the flexibility and scalability needed for handling the dynamic nature of recipe data. Each recipe can vary significantly in terms of the number of ingredients, preparation steps, and optional metadata such as tags or dietary preferences. MongoDB's schema-less structure allows us to store these recipes as self-contained documents without the constraints of a predefined relational schema, making it ideal for our use case. Additionally, our sequence diagram shows a simple flow where recipes are generated and saved in a single transaction, which MongoDB supports efficiently through its document-oriented approach. Given our microservices architecture, MongoDB enables independent scaling of the recipe-related services, ensuring performance and maintainability. Its seamless integration with our backend technologies (Flask or FastAPI) further streamlines development. Overall, MongoDB's document model, fast read/write operations, and alignment with agile development make it the most suitable choice for storing generated recipes in our system.

5. Software Requirements

5.a.) Functional Requirements:

Recipe Generation from Input

- The system shall allow users to input a list of ingredients and/or a dish name.
- The system shall generate a recipe based on the provided ingredients and/or dish name.
- The generated recipe shall include a list of ingredients, preparation steps, and cooking instructions.

Recipe Display

- The system shall display the generated recipe to the user in a clear and readable format via the frontend (React.js).

Recipe Saving

- The system shall provide an option for the user to save the generated recipe.
- If the user chooses to save, the system shall store the recipe in the database (MongoDB/PostgreSQL).
- The system shall allow users to retrieve previously saved recipes from the database.

User Interaction

- The system shall provide a user-friendly interface (via React.js) for entering inputs and viewing recipes.

- The system shall confirm successful saving of a recipe with a notification or message to the user.

5.b.) Non-functional requirements

Scalability

- The system shall support scaling individual services (frontend, backend, database) independently to handle increased user demand, as justified by the Microservices Architecture.

Performance

- The system shall generate and display a recipe within 5 seconds of receiving user input under normal load conditions.
- The system shall retrieve saved recipes from the database within 2 seconds.

Maintainability

- The codebase shall be well-documented to facilitate future development by team members.

Reliability

- The system shall ensure that saved recipes are accurately stored and retrievable with a 99% success rate.
- The system shall handle invalid inputs (e.g., empty ingredient list) gracefully by providing appropriate error messages.

Usability

- The frontend shall be intuitive and responsive, ensuring users can easily input data and view recipes on both desktop and mobile devices.
- The system shall support a consistent user experience across different browsers (e.g., Chrome, Firefox).

Security

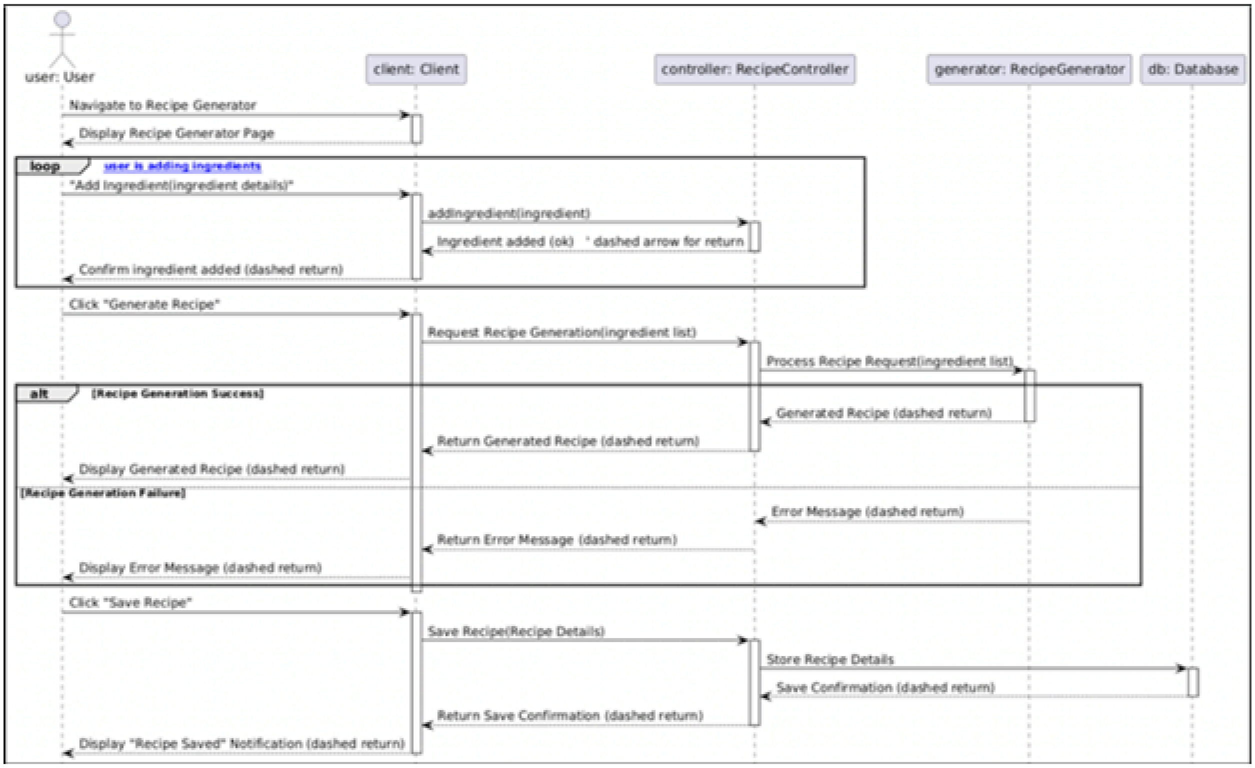
- The system shall protect user data (e.g., saved recipes) by implementing secure database access controls.
- The system shall validate user inputs to prevent injection attacks or malicious data entry.

Availability

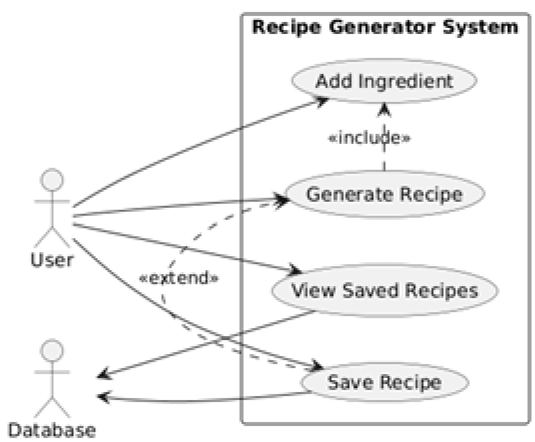
- The system shall be available 95% of the time, excluding planned maintenance, to ensure users can access the recipe generator when needed.

6. Diagrams (Use-Case/Sequence/Class/Activity)

Sequence Diagram:



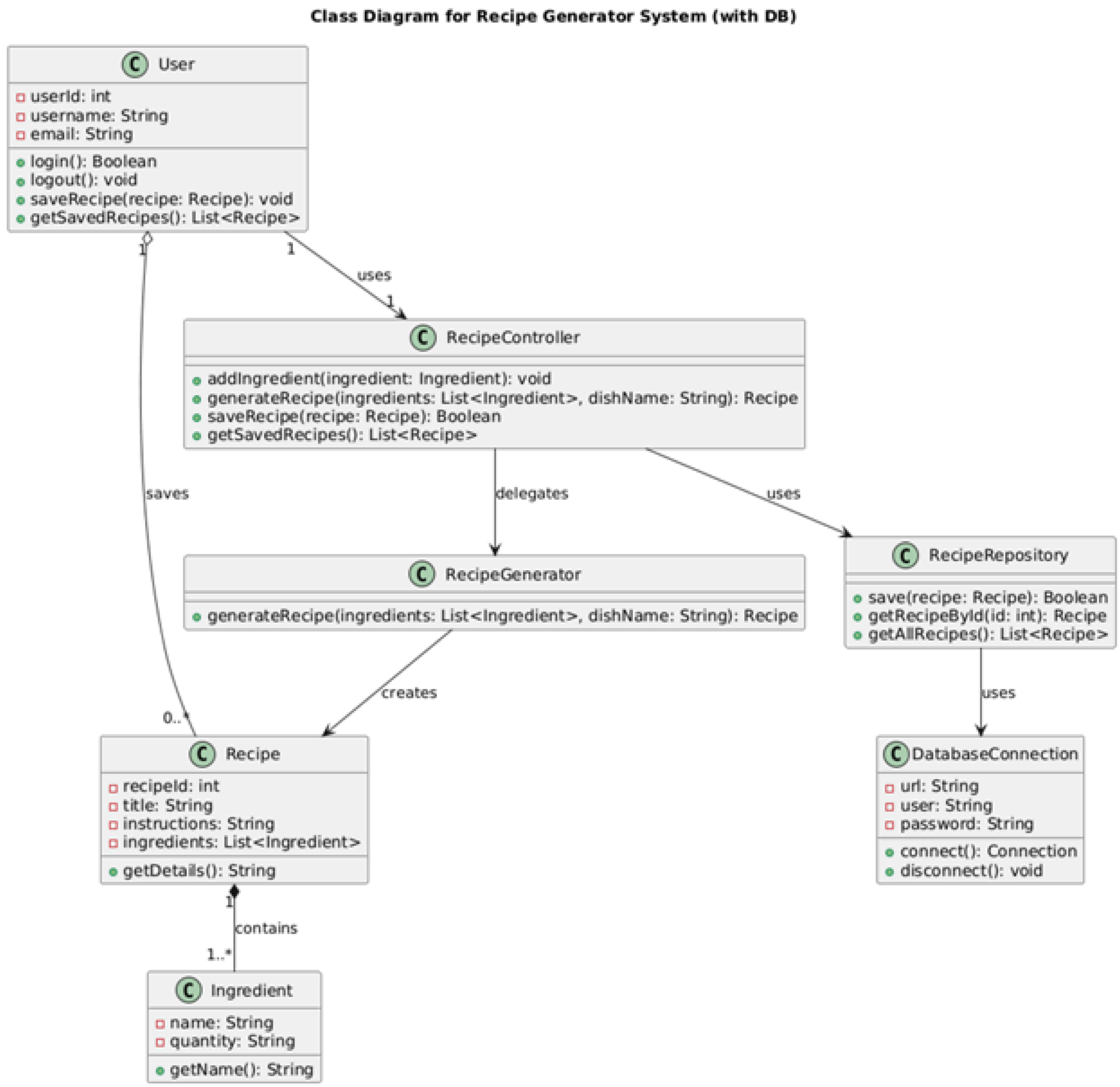
Use Case diagram:



Traceability Matrix:

Requirement ID	Priority Weight	UC1: Add Ingredient	UC2: Generate Recipe	UC3: Save Recipe	UC4: View Saved Recipes
FR1	3	X			
FR2	3		X		
FR3	2		X		
FR4	2		X		
FR5	2			X	
FR6	1			X	
FR7	1				X
FR8	1	X	X	X	X
FR9	1			X	
Score		4	8	5	2

Class diagram:



7. Architectural design

7.1. Describe why the pattern is selected

The **Microservices Architecture** pattern is chosen for scalability and maintainability. The system is divided into:

- **Frontend (React.js)**: Handles UI interactions.
- **Backend (FastAPI)**: Manages recipe generation logic and database interactions.
- **Database (MongoDB)**: Stores user-generated recipes.
- **External API (AI Model, if applicable)**: Fetches or generates recipe data.

Justification for Microservices Architecture:

- **Scalability**: Individual services can scale independently.
- **Flexibility**: Backend can support multiple frontends.
- **Maintainability**: Clear separation of concerns.

8. Final Project Draft Description

Project Overview

The **Recipe Generator** is a web-based application designed to help users generate recipes based on available ingredients or a dish name. The system allows users to **input ingredients, view generated recipes, and save them for future use**. Built with **React.js (frontend), FastAPI (backend), and MongoDB (database)**, it follows a **Microservices Architecture** to ensure scalability, flexibility, and maintainability.

Motivation & Goals

The project aims to assist users in **meal planning, reducing food waste, and saving time** by providing tailored recipe recommendations. As busy college students, our team recognizes the need for **a quick and efficient way to find recipes** based on available ingredients.

Key Features

- Users can **input ingredients or a dish name** to generate recipes.
- Recipes include a **list of ingredients, preparation steps, and cooking instructions**.
- Users can **save and retrieve recipes** from a database.
- The UI is **user-friendly, responsive, and accessible on multiple devices**.
- The system ensures **fast performance, security, and high availability**.

Technology Stack

- **Frontend:** React.js
- **Backend:** FastAPI
- **Database:** MongoDB
- **Version Control:** Git/GitHub

Development Approach

The project follows an **Agile development model using Scrum**, ensuring iterative improvements. Tasks were divided across two deliverables:

1. **Deliverable 1:** Prototyping UI (Figma), system design, and environment setup.
2. **Deliverable 2:** Frontend-backend implementation, database integration, and feature testing.

Expected Outcomes

By implementing a **scalable, efficient, and secure recipe generation system**, we aim to provide users with a **convenient tool for meal planning**, ultimately helping them make **better use of available ingredients and reduce food waste**.

Deliverable 2

Project Scheduling: Recipe Generator: Gantt chart

https://docs.google.com/spreadsheets/d/1Rux96Zyg8tQnRPaHcwHUANaqXXX97U6OcluaSlekRkc/edit?usp=sharing

Recipe Generator: GANTT CHART

PROJECT TITLE	Recipe Generator	Group 4	Rhea Aemireddy, Shannon Carter, Jay Chae, Roslyn Collings, Adair Gonzalez, Hafa Kazi, Veom Nemade, Ayush Velhal

TASK	Assigned	START DATE	WEEK 1	WEEK 2	WEEK 3	WEEK 4	WEEK 5	WEEK 6	WEEK 7	WEEK 8	WEEK 9	WEEK 10	WEEK 11
Deliverable 1													
Delegation of Tasks	Shannon	2/1/25											
Software Requirements	Adair	2/7/25											
Software Process Model	Safa	2/14/25											
Sequence Diagrams	Veom	2/28/25											
UI/UX Designing + Prototyping													
Landing Page	Shannon, Rhea	3/7/25											
Recipe Generator Flow	Ayush, Adair	3/7/25											
Saved Recipes	Jay, Roslyn, Hafa	3/7/25											
Environment Setup	Everyone	3/7/25											
Deliverable 2													
Deliverable 1 Corrections													
Formatting	Roslyn	3/28/25											
Class Diagram	Adair, Veom	3/28/25											
Sequence Diagrams	Veom	3/28/25											
Use Case Diagram	Ayush, Veom	3/28/25											
Database + Architectual Design	Rhea	3/28/25											
Taraceability Matrix	Shannon, Veom	3/28/25											
Project Code Implementation	Ayush, Veom	3/29/25											
Project Scheduling, Project duration and staffing	Shannon	4/4/25											
Cost, Effort and Pricing Estimation	Veom	4/4/25											
Test Plan	Adair	4/4/25											
Comparison to similar designs	Roslyn	4/4/25											
Conclusion	Rhea	4/4/25											
Github Repo Submission	Ayush	4/18/25											
References	Veom, Rosyln	4/18/25											

Project Duration:

Start Project Date: 2/1/2025

End Project Date: 4/29/2025

Total Duration: 11 Weeks

Staffing and Minutes:

- Shannon Carter
  - 2 Hours Per Week → 22 Hours Total
- Rhea Aemireddy
  - 2 Hours Per Week → 22 Hours Total
- Jay Chae
  - 2 Hours Per Week → 22 Hours Total
- Roslyn Collings
  - 2 Hours Per Week → 22 Hours Total
- Adair Gonzalez
  - 2 Hours Per Week → 22 Hours Total
- Hafa Kazi
  - 2 Hours Per Week → 22 Hours Total
- Veom Nemade
  - 2 Hours Per Week → 22 Hours Total
- Ayush Velhal
  - 2 Hours Per Week → 22 Hours Total

Meeting Minutes:

- Meeting 1: 2/1/2025
  - Attendance: Rhea, Shannon, Jay, Roslyn, Adair, Hafa, Veom, Ayush
- Meeting 2: 4/17/2025
  - Attendance: Rhea, Shannon, Roslyn, Veom
- Meeting 3: 4/24/2025
  - Attendance: Rhea, Veom, Roslyn, Jay, Adair, Shannon, Ayush
- Meeting 4: 4/29/2025
  - Attendance: Rhea, Veom, Roslyn, Ayush

3.2 Cost, Effort and Pricing Estimation (Functional Point)

Count:

# of user i/p= 10,

# of user o/p = 5,

# of user queries = 8,

# of data files = 30,

# of external interfaces = 4.

PC Calculation:

Q1 = 1 , Q2 = 3, Q3 = 2, Q4 = 2, Q5 = 1, Q6 = 4, Q7 = 1, Q8 = 3, Q9 = 4, Q10 = 5, Q11 = 2, Q12 = 0, Q13 = 0, Q14 = 4

PC = 32

PCA Calculation:

PCA = 0.65 + 0.01(32) **= 0.97**

From the count, PC, and PCA we can calculate the GFP and therefore the function point and estimated effort for the simple, average, and complex case.

Productivity = 60 (assumed from slides)

Complexity	**GFP**	**PC**	**PCA**	**FP = GFP * PCA**	**Effort = FP/ 60**	**Duration = Effort / 8 people**
Simple	304	32	0.97	294.9	4.92 weeks	0.62 weeks
Average	425	32	0.97	412.3	6.87 weeks	0.86 weeks
Complex	633	32	0.97	614.0	10.23 weeks	1.28 weeks

3.3 Estimated cost of hardware products

Item	**Qty**	**Unit Cost**	**Total (6 mo)**
Cloud VM (t3.small)	**1**	**$15.18 /mo [Amazon EC2 Instance Type Comparison]**	**$91.08 (15.18 × 6) [Amazon EC2 Instance Type Comparison]**

3.4 Estimated cost of software products

Item	Qty	Unit Cost	Total
Domain Registration (.app)	1 yr	$12.98 / yr Namecheap	$12.98
Figma Professional (Full Seat)	3 seats × 6 mo	$16 /mo per seat Figma	$288.00 (16 × 3 × 6)
MongoDB Atlas M2 Cluster	6 mo	$60 / mo MongoDB	$360.00 (60 × 6)
Total Software			$660.98

3.5 Estimated cost of personnel

Role	Est. Hours	Rate	Total
Front-end Development	80	$25 /hr Upwork	$2,000 (80 × 25)
Back-end Development	100	$25 /hr Upwork	$2,500 (100 × 25)
QA & Testing	40	$15 /hr Upwork	$600 (40 × 15)

Project Management	**20**	**$34 /hr** [**Salary.com**]	**$680 (20 × 34)**
Total Personnel			**$5,780**

Grand Total Project Cost

- **Hardware: $ 91.08**
- **Software: $ 660.98**
- **Personnel: $ 5,780**

Overall ≈ $ 6,532.06

---

4. Software Testing

Scope

The test plan targets the following functional requirements of the Recipe Generator:

- **Recipe Generation from Input**: Users input ingredients and/or a dish name, and the system generates a recipe with ingredients, preparation steps, and cooking instructions.
- **Recipe Display**: The system displays the generated recipe in a clear, readable format via the React.js frontend.
- **Recipe Saving**: Users can save generated recipes to a database (MongoDB/PostgreSQL) and retrieve them later.
- **User Interaction**: The system provides a user-friendly interface for inputting data, viewing recipes, and receiving confirmation of saved recipes.

Testing Strategies

The following black-box testing strategies will be applied:

- **Equivalence Partitioning (EP)**: Divides the input domain into disjoint subsets where inputs are expected to produce similar behavior. Test cases select one value from each partition.
- **Boundary Value Analysis (BVA)**: Selects test cases at the edges of equivalence partitions to detect errors at boundaries.
- **Cause-Effect Testing**: Analyzes input conditions and their outcomes to derive test cases using a decision table.

Test Case Specification

- **Recipe Generation from Input**
  Description: The system generates a recipe based on user-provided ingredients and/or a dish name.
  Input Domain:
  - **Ingredients**: A list of strings representing ingredients (e.g., "chicken, peppers, rice, onions, tomatoes, spices").
  - **Dish Name**: A string representing a dish (e.g., "chicken curry").
  - **Constraints**: Ingredients can be empty, single, or multiple; dish name can be empty or non-empty.

  Equivalence Partitioning:
  - **Ingredients**:
    - **Valid Partitions**:
      - Non-empty list of valid ingredients (e.g., "chicken, rice, peppers, rice, onions, spices").
      - Single ingredient (e.g., "chicken").
      - Empty ingredient list (allowed per requirements).
    - **Invalid Partitions**:
      - Malformed input (e.g., non-string characters like "123!@#").
      - Excessively long ingredient list (e.g., >100 ingredients, if system has a limit).
  - **Dish Name**:
    - **Valid Partitions**:
      - Non-empty dish name (e.g., "chicken curry").
      - Empty dish name (allowed per requirements).
    - **Invalid Partitions**:
      - Malformed dish name (e.g., special characters like "!@#").
  - Excessively long dish name (e.g., >100 characters)

  Boundary Value Analysis:
  Ingredients:
    - Empty list (0 ingredients).
    - Single ingredient (1 ingredient).
    - Maximum allowed ingredients (e.g., 100, if specified; otherwise, test with a large number like 1000).
    - One less than maximum (e.g., 99).
    - One more than maximum (e.g., 101).

  Dish Name:

- Empty string (0 characters).
- Single character (e.g., "a").
- Maximum allowed length (e.g., 100 characters).
- One less than maximum (e.g., 99 characters).
- One more than maximum (e.g., 101 characters).

Test Cases

#	Description	Input	Output	Criteria
1	Valid ingredients and dish name	Ingredients: "chicken, peppers, rice, onions, tomatoes, spices", Dish: "chicken curry"	Recipe with ingredients, steps, instructions	Recipe generated correctly
2	Valid ingredients only	Ingredients: "chicken, peppers, rice, onions, tomatoes, spices", Dish: ""	Recipe with ingredients, steps, instructions	Recipe generated correctly
3	Valid dish name only	Ingredients: "", Dish: "chicken curry"	Recipe with ingredients, steps, instructions	Recipe generated correctly
4	Empty input	Ingredients: "", Dish: ""	Error message	Error displayed
5	Invalid input format	Ingredients: "123456&*", Dish: "chicken curry"	Error message	Error displayed
6	Boundary: empty input	Ingredients: "", Dish: "chicken curry"	Recipe or error	Matches expected behavior
7	Boundary: single ingredient	Ingredients: "chicken", Dish: ""	Recipe with ingredients and steps	Recipe generated correctly
8	Boundary: Max ingredients	Ingredients: 100 valid ingredients	Recipe or error	Matches expected behavior
9	Boundary: Max dish name length	Ingredients: "chicken", Dish: 100-char string	Recipe or error	Matches expected behavior

- **Recipe Display**

  Description: The system displays the generated recipe in a clear, readable format via the frontend.
  Input Domain: Generated recipe (list of ingredients, preparation steps, cooking instructions).

  Equivalence Partitioning:
  - **Valid Partitions:**
    - Complete recipe (all components: ingredients, steps, instructions).
    - Partial recipe (e.g., only ingredients and steps).
  - **Invalid Partitions:**
    - Empty recipe (no components).
    - Malformed recipe (e.g., non-string data).

  Boundary Value Analysis:
  - Empty recipe (0 components).
  - Minimal recipe (1 ingredient, 1 step, 1 instruction).
  - Large recipe (e.g., 100 ingredients, 100 steps).

Test Cases

#	Description	Input	Output	Criteria
1	Valid complete recipe	Recipe with full ingredients, steps and instructions	Displayed clearly in UI	All components visible and readable
2	Partial recipe	Recipe with partial ingredients, steps and instructions	Displayed clearly in UI	All components visible and readable
3	Empty recipe	Empty recipe	Error message or empty UI	Matches expected behavior
4	Boundary: Minimal recipe	Recipe with 1 ingredient, 1 step, and 1 instruction	Displayed clearly in UI	All components visible and readable
5	Boundary: Large recipe	100 ingredients, 100 steps	Displayed clearly in UI or	Matches expected

			error	behavior

- **Recipe Saving**
  Description: Users can save a generated recipe to the database and retrieve it later.
  Input Domain:
  - Recipe: A generated recipe.
  - User Action: Save request (e.g., button click).
  - Constraints: Database must store and retrieve recipes accurately.

  Equivalence Partitioning:
  - **Valid Partitions:**
    - Valid recipe to save (non-empty).
    - Retrieve saved recipe (existing in database).
  - **Invalid Partitions:**
    - Empty recipe.
    - Retrieve non-existent recipe.

  Boundary Value Analysis:
  - Empty recipe (0 components).
  - Minimal recipe (1 ingredient, 1 step).
  - Maximum recipe size (e.g., 100 ingredients, if database has limits).
  - Database limits (e.g., maximum number of saved recipes, if specified).

  Cause-Effect Testing:

  - **Causes:**
    - C1: Valid recipe provided.
    - C2: Save action triggered.
    - C3: Recipe exists in database (for retrieval).
  - **Effects:**
    - E1: Recipe saved successfully.
    - E2: Recipe retrieved successfully.
    - E3: Error message (save/retrieve failure).
- **Decision Table:**

Rule	C1	C2	C3	E1	E2	E3
1	Y	Y	-	X		
2	N	Y	-			X

3	**Y**	**N**	**Y**		**X**	
4	**Y**	**N**	**N**			**X**

Test Cases

#	Description	Input	Output	Criteria
1	Save valid recipe	Valid recipe, click save	Confirmation message, recipe in DB	Recipe saved, retrievable
2	Save empty recipe	Empty recipe, click save	Error	Error
3	Retrieve saved recipe	Select saved recipe	Recipe displayed in UI	Matches saved recipe
4	Retrieve non-existent recipe	Select non-existent recipe	Error	Error
5	Boundary: Minimal recipe	1 ingredient, 1 step, save	Confirmation message, recipe in DB	Recipe saved, retrievable
6	Boundary: Maximum recipe	100 ingredients, save	Confirmation or error	Matches expected behavior

- **User Interaction**

  Description: The system provides a user-friendly interface for inputting data, viewing recipes, and receiving save confirmations.

  Input Domain:
  - UI Inputs: Text fields (ingredients, dish name), buttons (submit, save).
  - Constraints: Inputs must be intuitive, and confirmations must be clear**.**

  Equivalence Partitioning:
  - **Valid Partitions:**
    - Valid text input (e.g., "chicken").
    - Valid button clicks (e.g., submit, save).
  - **Invalid Partitions:**
    - Malformed text input (e.g., "!@#").

- Invalid button state (e.g., submit with empty fields).

Boundary Value Analysis:

- Empty text field (0 characters).
- Single character input (e.g., "a").
- Maximum input length (e.g., 100 characters).

Test Cases

#	Description	Input	Output	Criteria
1	Valid input	Ingredients: "chicken, rice", click submit	Recipe displayed	Recipe visible in UI
2	Invalid input	Ingredients: "123456&*", click submit	Error	Error
3	Save confirmation	Save valid recipe	Confirmation message	Message visible in UI
4	Boundary: Empty input	Empty ingredients, click submit	Error	Error
5	Boundary: Max input	100-char ingredient, click submit	Recipe or error	Matches expected behavior

<u>**Test Environment**</u>

- **Frontend:** React.js, tested on Chrome and Firefox browsers
- **Backend:** Flask/FastAPI, connected to MongoDB/PostgreSQL.
- **Tools:** Manual testing via UI, automated testing with Jest (for React.js) if implemented.
- **Setup:** GitHub repository for version control

5. Comparison with Similar Designs

The purpose of our Gen-AI Recipe Generator is to find convenient recipes. This could be accomplished through other technological means, most prominently by searching a recipe database or by using Gen-AI on its own. Our project aims to outperform these competitors in overall convenience.

Time-efficiency

Lower in recipe databases due to the nature of searching versus the nature of generating. Allrecipes.com reportedly has 51K original recipes (Allrecipes), a limited number. Gen-AI will always produce a result, which makes it more time-effective.

Recipe quality

Human-submitted recipes on a recipe database have been tested by the submitters or by other users. This makes the quality more consistent than a unique, generated recipe. However, we provide safeguards and boundaries that improve the quality of our output when compared to that of gen-AI on its own.

Recipe specificity

A database is not tailor-made to the user and doesn't always have a recipe meeting requirements. Again, because of its limits in size, it falls short in comparison to gen-AI.

Framework

The framework of Gen-AI on its own is not tailored for recipes. Our recipe generator, much like a recipe database, will provide a structured, user-friendly environment with necessary boundaries and safeguards.

6. Conclusion

Conclusion Statement:

The Recipe Generator project successfully integrated front-end development, backend functionality, database storage, and user-focused interface design to provide a smooth and responsive recipe-generation experience. With thorough software testing methods—such as equivalence partitioning, boundary value analysis, and cause-effect testing—the team ensured reliability, usability, and error handling across key functional areas: recipe input, display, saving, and user interaction.

Challenges and Changes in Project Management and Software Planning:

1. **Project Management Challenges:**

- *Team Coordination*: Managing availability across contributors, especially with freelance developers from platforms like Upwork, required frequent check-ins and progress tracking using task lists.
- *Scope Creep Prevention*: As ideas evolved, there was a temptation to expand features (e.g., advanced filtering, nutrition suggestions). The team learned to balance innovation with feasibility by prioritizing the MVP.
- *Budget Allocation*: Cost estimation had to be adjusted to accommodate higher-than-expected SaaS costs (e.g., MongoDB pricing tiers), leading to careful budget reallocation and a sharper focus on essential services.

2. **Software Planning Adjustments:**
   - *Technology Stack Changes*: Initial backend planning considered Node.js, but the team pivoted to Flask/FastAPI for quicker setup and better integration with MongoDB Atlas.
   - *Testing Strategy Expansion*: Originally, testing was limited to basic input validation. Midway through, structured black-box testing approaches were adopted for better coverage, resulting in more detailed and reliable test plans.
   - *Database Choice*: Although PostgreSQL was considered for relational integrity, MongoDB was ultimately chosen due to the document-based structure of recipes, which aligned better with flexible input fields and nested data.
3. **Development Learnings:**
   - *UI/UX Feedback*: Early user feedback led to interface simplifications—like clearer form validation and real-time feedback messages—that significantly improved user interaction.
   - *Version Control Discipline*: Versioning and branching strategies using GitHub helped reduce code merge conflicts and allowed parallel development of frontend and backend components.

WORKS CITED

Allrecipes Team. *Allrecipes*, Allrecipes, www.allrecipes.com. Accessed 22 Apr. 2025.

“T3.Small - Amazon EC2 Instance Type.” *T3.Small - Amazon EC2 Instance Type*, aws-pricing.com/t3.small.html?utm_source=chatgpt.com. Accessed 29 Apr. 2025.

Front-End Developer Hourly Rates | Cost to Hire Front-End Developer | Upwork, www.upwork.com/hire/front-end-developers/cost/. Accessed 29 Apr. 2025.

..Website Domain Registration | Buy .Website New gTLD for $0.98 - NAMECHEAP, www.namecheap.com/domains/registration/gtld/website/. Accessed 29 Apr. 2025.

“Billing Breakdown and Optimization - Atlas - Mongodb Docs.” *Billing Breakdown and Optimization - Atlas - MongoDB Docs*, www.mongodb.com/docs/atlas/billing/billing-breakdown-optimization/?utm_source=chatgpt.com. Accessed 29 Apr. 2025.

Salary.com, Site built by: “Hourly Wage for Project Management Manager.” *Salary.Com*, 1 Apr. 2025, www.salary.com/research/salary/benchmark/project-management-manager-hourly-wages.

Front-End Developer Hourly Rates | Cost to Hire Front-End Developer | Upwork, www.upwork.com/hire/front-end-developers/cost/. Accessed 29 Apr. 2025.

Back-End Developer Hourly Rates | Cost to Hire Back-End Developer | Upwork, www.upwork.com/hire/back-end-developers/cost/. Accessed 29 Apr. 2025.

Cost to Hire Software QA Tester, www.upwork.com/en-gb/hire/software-qa-testers/cost/. Accessed 29 Apr. 2025.