TASTE HUB
CULINARY SHARING

Project Final Report

Authors: Jean-Nicolas Rouette (30121189), Edward An (30142179), Jad Khalil (30132370), Jacob Lever (30147405), Jori Duguid (30145690), Fardin Aryan (30161450), Noshin Zion (30163372).
Contributors: No outside contributions.

Schulich School of Engineering, University of Calgary
Software Architecture - SENG401 - B03 - Winter 2024

Last Updated: 27th of March 2024
Status: Final Report

Github Link: https://github.com/JadKhalil/TasteHub
Website Link: https://main--thetastehub.netlify.app/

Architecture Diagram

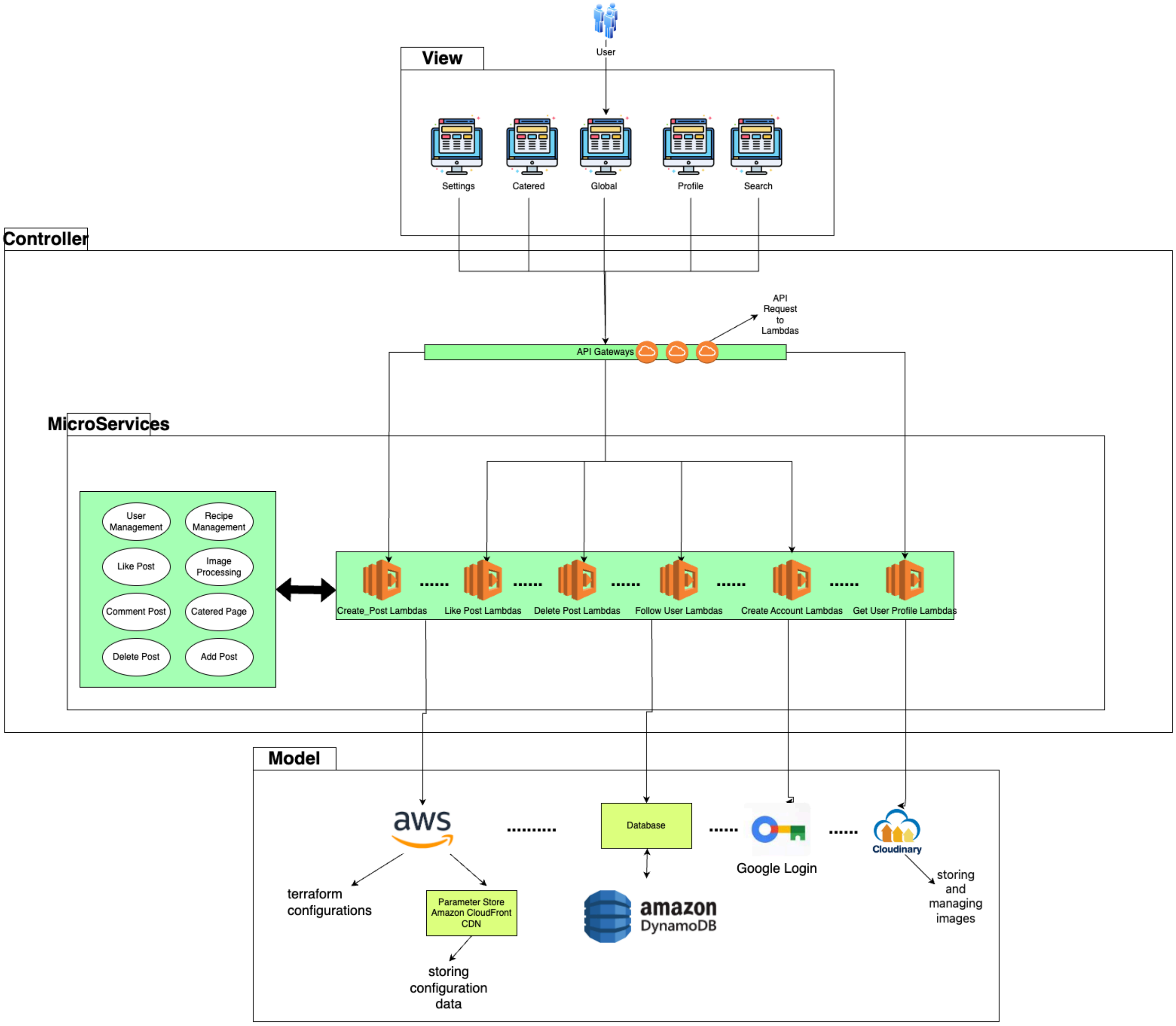
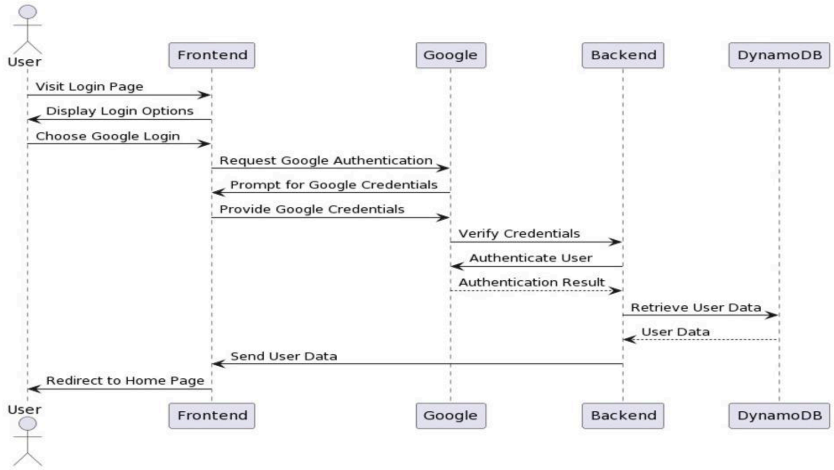


Figure 1: Architecture Diagram

The architecture diagram is a hybrid of MVC and microservices. The view and the model are completely separate from each other; they can only communicate with the controller. The controller stores/fetches data to/from the model based on the user interaction with the view. The controller is further divided into many microservices that are separate from one another. Because each AWS Lambda function is a serverless, event-driven compute service, they operate completely independently from one another. Even when one lambda function fails, the other lambda functions can still operate. Each AWS Lambda function communicates with different components within the model depending on its use case. Although all of them interact with the AWS DynamoDB, some lambda functions also send requests to Cloudinary.

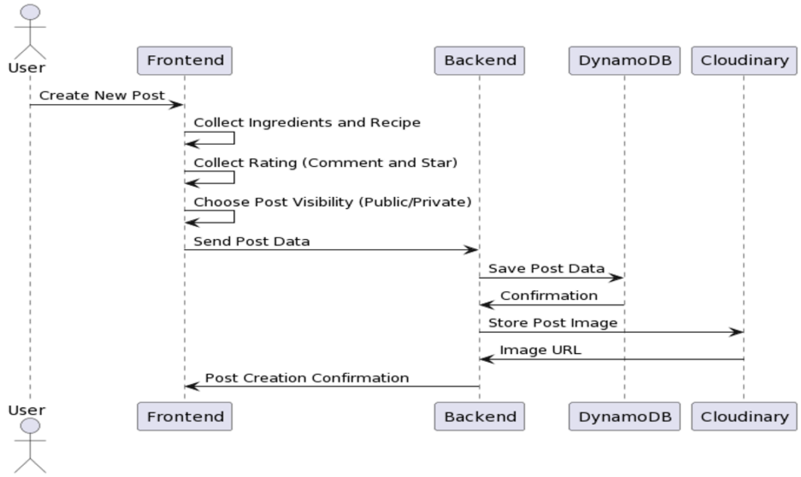
Sequence Diagrams

Login Process



The sequence diagram outlines the login process for TasteHub, a web application for sharing recipes. It begins with the user visiting the login page and choosing to sign in via Google. The frontend requests authentication from Google, prompting the user to enter their credentials, which Google then verifies. Upon successful verification, Google authenticates the user and sends the result back to the application's backend. The backend retrieves the user's data from DynamoDB and forwards it to the frontend, which then redirects the user to TasteHub's home page, completing the login process.

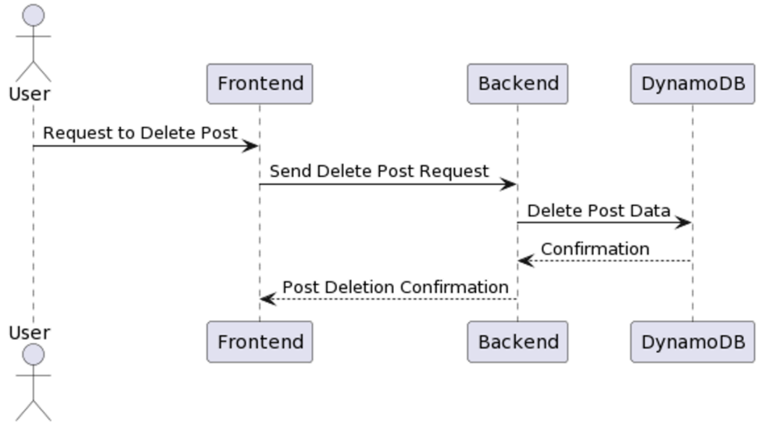
Creating Post



The sequence diagram represents the "Creating Post" process in TasteHub: A user initiates the process by creating a new post via the frontend, where they enters details such as ingredients, the recipe, and a rating that includes comments and stars, along with choosing whether the post will be public or private.

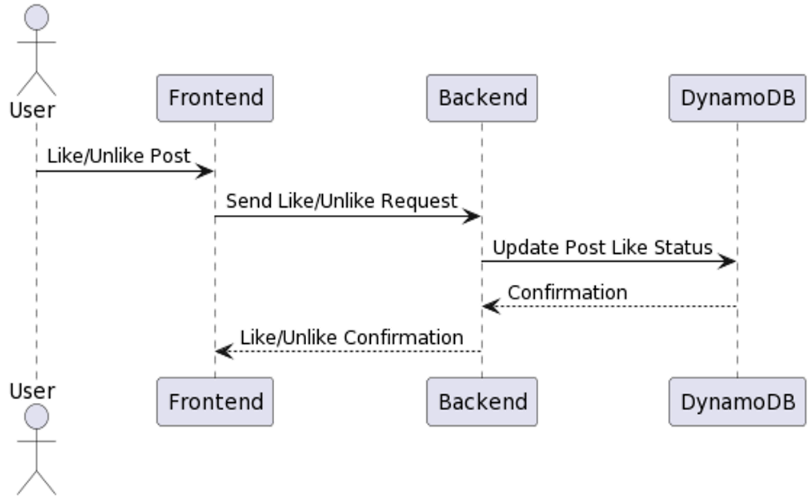
After the user submits this data, the frontend sends it to the backend, which then saves the post data to DynamoDB and stores the post image on Cloudinary, a cloud image service, which responds with the image URL. Finally, the backend confirms the successful creation of the post to the frontend, which in turn confirms to the user that their post has been created.

Deleting Post



The sequence diagram depicts the "Deleting Post" workflow for TasteHub: A user sends a request to the frontend to delete a specific post, which in turn sends a delete request to the backend. The backend then processes this request by deleting the post data from DynamoDB. Upon successful deletion, the backend sends a confirmation back to the frontend, which subsequently confirms to the user that the post has been deleted. This completes the post deletion process from the user's initiation to the confirmation of removal.

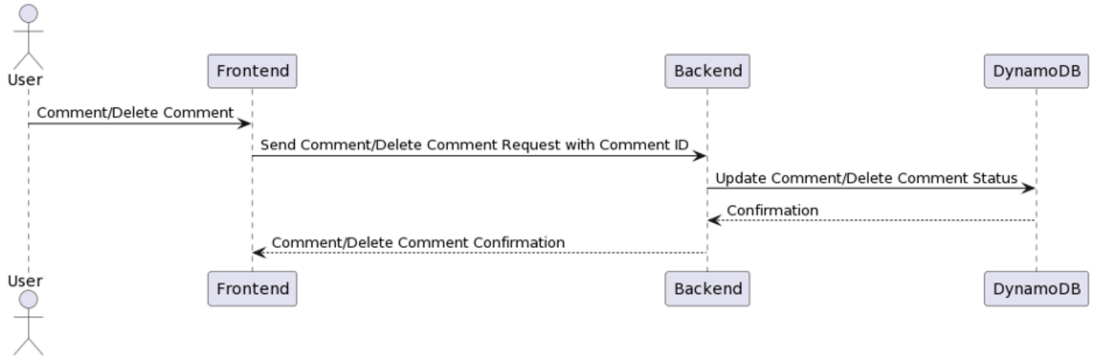
Liking/Unliking Post



The above sequence diagram depicts the process of liking and unliking a post in our application.

The process begins with the user liking or unliking a post on the frontend of the social media application. This action sends a request to the backend servers. The backend acknowledges the request and updates the like status of the post in the DynamoDB database, most likely the social media app’s database. Once the update is complete, a confirmation message is sent back to the backend servers. The frontend then receives confirmation and updates the user interface accordingly.

Commenting/Deleting Comment


User
Frontend
Backend
DynamoDB
Comment/Delete Comment
Send Comment/Delete Comment Request with Comment ID
Update Comment/Delete Comment Status
Confirmation
Comment/Delete Comment Confirmation
User
Frontend
Backend
DynamoDB

The above sequence diagram illustrates the process of making and deleting a comment in a post.

The process starts with the user creating a comment or selecting to delete a comment. This action triggers the frontend to send a request to the backend. The backend communicates with DynamoDB to update the comment’s status or delete it altogether. Finally, the backend sends a confirmation message back to the frontend, which then relays the information back to the user.

RTM Evolution

LEGEND:

✅ = Complete

❌ = Incomplete

🔜 = Yet to start

🔄 = In-progress

Version No. 1.0

Summary:

The design phase for most application requirements is complete, and development has commenced with some requirements 'ready to deploy.' Testing activities have not yet started.

Requirement	Description	Design	Development	Testing	Deployment
RQ1	User login	✅	✅	🔜	✅
RQ2	User Signup	❌	🔜	🔜	❌
RQ3	User Feed	✅	✅	🔜	✅
RQ4	Catered Page	✅	🔄	🔜	❌
RQ5	Creating New Post	✅	✅	🔜	✅
RQ6	Searching Post	✅	🔄	🔜	✅
RQ7	Accessing Profile	✅	🔄	🔜	✅
RQ8	Reacting To Post	✅	🔄	🔜	✅
RQ9	Commenting In A Post	❌	🔜	🔜	❌
RQ10	Post Description	✅	✅	🔜	✅
RQ11	User Logout	✅	✅	🔜	✅

Version No. 2.0

Summary:

Req ID	Functional Requirements				Status	Design	Testing	Non-functional requirements
	Main Requirements	Sub Requirements	Category	Description				
1	Overall Project	- User Management - Recipe Management - Social Interaction	Required	Defines functionalities for a social recipe sharing platform, enabling users to post and browse recipes, ensuring secure user management	🔄	🔄	🔄	Performance, security, usability, Support, availability, localizability
2	User Management	- Sign up - Sign In - Editing profile - Deleting profile	Required	Enables users to create, access, and modify their accounts.	🔄	🔄	🔜	Performance, security, usability, availability, maintainability
3	User signs up	N/A	Required	Creates an account using email and an auto-generated username based on first and last name	✅	✅	🔜	Performance, security, usability, maintainability
4	User signs in	N/A	Required	Logs in using Gmail	✅	✅	🔜	Performance, security, usability, maintainability
5	User editing profile	N/A	Expected	Modifies account information	🔄	🔄	🔜	Performance, security, usability, maintainability
6	User deleting profile	N/A	Expected	Permanently deletes the account from the database system	🔜	🔜	🔜	Performance, security, usability, maintainability
7	User logs out	N/A	Expected	Logging out of the account from the online system	✅	✅	🔜	Performance, security, usability, maintainability
8	Recipe Management	- Create recipes - Add images to recipe - Edit existing recipe - Delete recipes - Categorize recipe - Search recipes	Required	Enables users to create, edit, share, and browse recipes with ingredients and instructions.	✅	✅	🔄	Performance, security, usability, availability, maintainability

9	Create recipes	N/A	Required	Enabling the user to create a recipe in a friendly UI	✅	✅	🔜	Performance, security, usability, maintainability
10	Add images to recipe	N/A	Expected	Adding images to the recipe, making it more user friendly	✅	✅	🔜	Performance, security, usability, maintainability
11	Edit existing recipe	N/A	Expected	Edit recipes easily with a user-friendly interface.	❌	❌	❌	Performance, security, usability, maintainability
12	Delete recipes	N/A	Expected	Deleting the recipe from the database and the cloud too	✅	✅	🔜	Performance, security, usability, maintainability
13	Categorize recipes	N/A	Expected	Enabling the user to filter the recipes based on recipe type, user information etc.	✅	✅	🔜	Performance, security, usability, maintainability
14	Search recipes	N/A	Expected	Find recipes quickly (search & filter) with a user-friendly experience.	✅	✅	🔜	Performance, security, usability, maintainability
15	Social Interaction	-Follow/unfollow users -User feed -Liking/unliking user recipes -Commenting on a post - Deleting comments	Required	Enables users to connect and engage with each other through following, commenting, and liking recipes.	✅	✅	🔄	Performance, scalability, availability
16	Follow user	N/A	Expected	Easy user following with optional privacy control.	✅	✅	🔄	Usability, security
17	Unfollow user	N/A	Expected	Easy user unfollowing with optional privacy control.	✅	✅	🔄	Usability, security
18	Get Followers and Following	N/A	Expected	Allows users to see a list of their followers and followings	🔜	🔄	🔄	Usability, performance, security
19	User feed	N/A	Expected	Displays recent recipe updates from followed users and all users quickly and efficiently.	✅	✅	✅	Usability, performance, moderation
20	Liking user recipes	N/A	Expected	Simple recipe liking with instant UI update.	✅	✅	🔄	Usability, performance
21	Uniking user recipe	N/A	Expected	Simple recipe unliking with instant UI update.	✅	✅	🔄	Usability, performance
22	Get user's liked recipes	N/A	Expected	Displays a list of all user's liked recipes	🔜	🔄	🔄	Usability, performance

23	Show list of likes on post	N/A	Expected	Displays a list of all users who liked a post	🔜	🔄	🔄	Usability, performance
24	Commenting in a post	N/A	Expected	User-friendly commenting with fast submission and optional moderation.	✅	✅	🔄	Usability, performance, moderation
25	Get Comments on a Post	N/A	Expected	Displays the comment section under each post in a user-friendly format	✅	✅	🔄	Usability, performance
26	Deleting a Comment	N/A	Expected	Allows the user to remove their comment	🔄	✅	🔄	Usability, performance

Our software has its core functionality established. We are currently working on adding minor components, while backend lambda function testing is done. Other testing phases are yet to start.**Version No. 2.5**

Summary:

The core functionality of the software is currently undergoing testing. Less crucial components have been implemented and completed. Additionally, exploratory testing is currently underway.

Req ID	Functional Requirements				Status	Design	Testing	Non-functional requirements
	Main Requirements	Sub Requirements	Category	Description				
1	Overall Project	- User Management - Recipe Management - Social Interaction	Required	Defines functionalities for a social recipe sharing platform, enabling users to post and browse recipes, ensuring secure user management	🔄	🔄	🔄	Performance, security, usability, Support, availability, localizability
2	User Management	- Sign up - Sign In - Editing profile - Deleting profile	Required	Enables users to create, access, and modify their accounts.	🔄	🔄	✅	Performance, security, usability, availability, maintainability
3	User signs up	N/A	Required	Creates an account using email and an auto-generated username based on first and last name	✅	✅	✅	Performance, security, usability, maintainability
4	User signs in	N/A	Required	Logs in using Gmail	✅	✅	✅	Performance, security, usability, maintainability
5	User editing profile	N/A	Expected	Modifies account information	✅	✅	🔄	Performance, security, usability, maintainability
6	User deleting profile	N/A	Expected	Permanently deletes the account from the database system	❌	❌	❌	Performance, security, usability, maintainability
7	User logs out	N/A	Expected	Logging out of the account from the online system	✅	✅	✅	Performance, security, usability, maintainability
8	Recipe Management	- Create recipes	Required	Enables users to create, edit, share, and browse recipes	✅	✅	✅	Performance, security, usability, availability, maintainability

		- Add images to recipe - Edit existing recipe - Delete recipes - Categorize recipe - Search recipes		with ingredients and instructions.				
9	Create recipes	N/A	Required	Enabling the user to create a recipe in a friendly UI	✅	✅	✅	Performance, security, usability, maintainability
10	Add images to recipe	N/A	Expected	Adding images to the recipe, making it more user friendly	✅	✅	✅	Performance, security, usability, maintainability
11	Edit existing recipe	N/A	Expected	Edit recipes easily with a user-friendly interface.	❌	❌	❌	Performance, security, usability, maintainability
12	Delete recipes	N/A	Expected	Deleting the recipe from the database and the cloud too	✅	✅	✅	Performance, security, usability, maintainability
13	Categorize recipes	N/A	Expected	Enabling the user to filter the recipes based on recipe type, user information etc.	✅	✅	🔄	Performance, security, usability, maintainability
14	Search recipes	N/A	Expected	Find recipes quickly (search & filter) with a user-friendly experience.	✅	✅	✅	Performance, security, usability, maintainability
15	Social Interaction	-Follow/unfollow users -User feed -Liking/unliking user recipes -Commenting on a post - Deleting comments	Required	Enables users to connect and engage with each other through following, commenting, and liking recipes.	✅	✅	🔄	Performance, scalability, availability
16	Follow user	N/A	Expected	Easy user following with optional privacy control.	✅	✅	🔄	Usability, security
17	Unfollow user	N/A	Expected	Easy user unfollowing with optional privacy control.	✅	✅	🔄	Usability, security
18	Get Followers and Following	N/A	Expected	Allows users to see a list of their followers and followings	❌	❌	❌	Usability, performance, security
19	User feed	N/A	Expected	Displays recent recipe updates from followed users and all users quickly and efficiently.	✅	✅	✅	Usability, performance, moderation
20	Liking user recipes	N/A	Expected	Simple recipe liking with instant UI update.	✅	✅	🔄	Usability, performance
21	Unliking user recipes	N/A	Expected	Simple recipe unliking with instant UI update.	✅	✅	🔄	Usability, performance
22	Get user's liked recipes	N/A	Expected	Displays a list of all user's liked recipes	❌	❌	❌	Usability, performance
23	Show list of likes on post	N/A	Expected	Displays a list of all users who liked a post	❌	❌	❌	Usability, performance
24	Commenting in a post	N/A	Expected	User-friendly commenting with fast submission and optional moderation.	✅	✅	🔄	Usability, performance, moderation

Version No. 3.0

Summary: The core functionality of the software has undergone full testing. Exploratory Testing Is also complete.

Req ID	Functional Requirements				Status	Design	Testing	Non-functional requirements
	Main Requirements	Sub Requirements	Category	Description				
1	Overall Project	- User Management - Recipe Management - Social Interaction	Required	Defines functionalities for a social recipe sharing platform, enabling users to post and browse recipes, ensuring secure user management	🔄	🔄	✅	Performance, security, usability, Support, availability, localizability
2	User Management	- Sign up - Sign In - Editing profile - Deleting profile	Required	Enables users to create, access, and modify their accounts.	🔄	🔄	✅	Performance, security, usability, availability, maintainability
3	User signs up	N/A	Required	Creates an account using email and an auto-generated username based on first and last name	✅	✅	✅	Performance, security, usability, maintainability
4	User signs in	N/A	Required	Logs in using Gmail	✅	✅	✅	Performance, security, usability, maintainability
5	User editing profile	N/A	Expected	Modifies account information	✅	✅	✅	Performance, security, usability, maintainability
6	User deleting profile	N/A	Expected	Permanently deletes the account from the database system	❌	❌	❌	Performance, security, usability, maintainability
7	User logs out	N/A	Expected	Logging out of the account from the online system	✅	✅	✅	Performance, security, usability, maintainability
8	Recipe Management	- Create recipes - Add images to recipe - Edit existing recipe - Delete recipes - Categorize recipe - Search recipes	Required	Enables users to create, edit, share, and browse recipes with ingredients and instructions.	✅	✅	✅	Performance, security, usability, availability, maintainability
9	Create recipes	N/A	Required	Enabling the user to create a recipe in a friendly UI	✅	✅	✅	Performance, security, usability, maintainability
10	Add images to recipe	N/A	Expected	Adding images to the recipe, making it more user friendly	✅	✅	✅	Performance, security, usability, maintainability
11	Edit existing recipe	N/A	Expected	Edit recipes easily with a user-friendly interface.	❌	❌	❌	Performance, security, usability, maintainability
12	Delete recipes	N/A	Expected	Deleting the recipe from the database and the cloud too	✅	✅	✅	Performance, security, usability, maintainability

13	Categorize recipes	N/A	Expected	Enabling the user to filter the recipes based on recipe type, user information etc.	✅	✅	✅	Performance, security, usability, maintainability
14	Search recipes	N/A	Expected	Find recipes quickly (search & filter) with a user-friendly experience.	✅	✅	✅	Performance, security, usability, maintainability
15	Social Interaction	-Follow/unfollow users -User feed -Liking/unliking user recipes -Commenting on a post - Deleting comments	Required	Enables users to connect and engage with each other through following, commenting, and liking recipes.	✅	✅	✅	Performance, scalability, availability
16	Follow user	N/A	Expected	Easy user following with optional privacy control.	✅	✅	✅	Usability, security
17	Unfollow user	N/A	Expected	Easy user unfollowing with optional privacy control.	✅	✅	✅	Usability, security
18	Get Followers and Following	N/A	Expected	Allows users to see a list of their followers and followings	❌	❌	❌	Usability, performance, security
19	User feed	N/A	Expected	Displays recent recipe updates from followed users and all users quickly and efficiently.	✅	✅	✅	Usability, performance, moderation
20	Liking user recipes	N/A	Expected	Simple recipe liking with instant UI update.	✅	✅	✅	Usability, performance
21	Unliking user recipes	N/A	Expected	Simple recipe unliking with instant UI update.	✅	✅	✅	Usability, performance
22	Get user’s liked recipes	N/A	Expected	Displays a list of all user’s liked recipes	❌	❌	❌	Usability, performance
23	Show list of likes on post	N/A	Expected	Displays a list of all users who liked a post	❌	❌	❌	Usability, performance
24	Commenting in a post	N/A	Expected	User-friendly commenting with fast submission and optional moderation.	✅	✅	✅	Usability, performance, moderation
25	Get Comments on a Post	N/A	Expected	Displays the comment section under each post in a user-friendly format	✅	✅	✅	Usability, performance
26	Deleting a comment	N/A	Expected	Allows the user to remove their comment	✅	✅	✅	Usability, performance

Object Oriented Programming

We chose not to implement Object-Oriented Programming (OOP) in our application, which utilizes AWS DynamoDB, AWS Lambda, React, and adopts the MVC and microservices architecture, as a strategic alignment with design patterns better suited to our project's goals. OOP emphasis on encapsulating data and behavior into objects, would not have been a good choice with our technological architectural patterns. In short, the services that we used, such as AWS Lambda and React.js all prompt and encourage a stateless programming pattern and are better suited for functional programming principles.

Testing

We used Postman to perform API tests of lambda functions. These API tests were performed:

- Create User Profile
- Get User profile
- Create Post
- Get Personal Posts
- Create Comment
- Get Comments on Post
- Delete Comment
- Like Post
- Get Likes on Post
- Get User's Liked Posts
- Unlike Post
- Delete Post
- Follow User
- Get Followers
- Get Following
- Unfollow User

These tests were performed in order because the outcome of some of the later test cases depended on the existence of the mock data in the database.

Create User Profile Test Case

- This test case was developed to ensure the fulfillment of requirement IDs #3 and #5 in the RTM.
- Executed by sending a HTTP POST request to the "Create User Profile" lambda function using a mock binary form data passed into the body of the request.
- Status code: 200 and success message are returned. A new row is added to the "tastehub-users" database table and a profile picture is uploaded to the Cloudinary storage.

Get User Profile Test Case

- This test case was developed to ensure the fulfillment of requirement ID #4 in the RTM.

- Executed by sending a HTTP GET request to the “Get User Profile” lambda function using a mock query parameter.
- Status code: 200, success message, and a JSON object of the user profile information are returned.

Create Post Test Case

- This test case was developed to ensure the fulfillment of requirement IDs #9 and #10 in the RTM.
- Executed by sending a HTTP POST request to the “Create Post” lambda function using a mock binary form data passed into the body of the request.
- Status code: 200, success message, and a JSON object of the new post are returned. The JSON object contains a Cloudinary URL of the image instead of the image file that was passed in the request. A new row is added to the “tastehub-posts” database table, the numberOfPosts column is updated on the “tastehub-users” database table, and an image is uploaded to the Cloudinary storage.

Get Personal Posts Test Case

- This test case was developed to ensure the fulfillment of requirement ID #19 in the RTM.
- Executed by sending a HTTP GET request to the “Get Personal Posts” lambda function using a mock query parameter.
- Status code: 200, success message, and a JSON object of a list containing all posts belonging to the mock user are returned.

Create Comment Test Case

- This test case was developed to ensure the fulfillment of requirement ID #24 in the RTM.
- Executed by sending a HTTP POST request to the “Create Comment” lambda function using a mock JSON data passed into the body of the request.
- Status code: 200 and success message are returned. A new row is added to the “tastehub-comments” database table and the numberOfComments column is updated on the “tastehub-posts” table.

Get Comments on Post Test Case

- This test case was developed to ensure the fulfillment of requirement ID #25 in the RTM.
- Executed by sending a HTTP GET request to the “Get Comments” lambda function using a mock query parameter.
- Status code: 200, success message, and a JSON object of a list containing all the comments belonging to a post are returned.

Delete Comment Test Case

- This test case was developed to ensure the fulfillment of requirement ID #26 in the RTM.

- Executed by sending a HTTP DELETE request to the “Delete Comment” lambda function using a mock query parameter.
- Status code: 200 and success message are returned. An existing row is removed from the “tastehub-comments” database table and the numberOfComments column is updated on the “tastehub-posts” table.

Like Post Test Case

- This test case was developed to ensure the fulfillment of requirement ID #20 in the RTM.
- Executed by sending a HTTP POST request to the “Like Post” lambda function using a mock JSON data passed into the body of the request.
- Status code: 200 and success message are returned. A new row is added to the “tastehub-likes” database table and the numberOfLikes column is updated on the “tastehub-posts” table.

Get Likes on Post Test Case

- This test case was developed to ensure the fulfillment of requirement ID #23 in the RTM.
- Executed by sending a HTTP GET request to the “Get Likes on Post” lambda function using a mock query parameter.
- Status code: 200, success message, and a JSON object of a list containing all the users who liked the post are returned.

Get User’s Liked Posts Test Case

- This test case was developed to ensure the fulfillment of requirement ID #22 in the RTM.
- Executed by sending a HTTP GET request to the “Get Users Liked Post” lambda function using a mock query parameter.
- Status code: 200, success message, and a JSON object of all the posts the user has liked are returned.

Unlike Post Test Case

- This test case was developed to ensure the fulfillment of requirement ID #21 in the RTM.
- Executed by sending a HTTP DELETE request to the “Unlike Post” lambda function using a mock query parameter.
- Status code: 200 and success message are returned. An existing row is removed from the “tastehub-likes” database table and the numberOfLikes column is updated on the “tastehub-posts” table.

Delete Post Test Case

- This test case was developed to ensure the fulfillment of requirement ID #12 in the RTM.
- Executed by sending a HTTP DELETE request to the “Delete Post” lambda function using a mock query parameter.

- Status code: 200, success message, and the delete cloudinary public key are returned. An existing row is removed from the “tastehub-posts” database table, the numberOfPosts column is updated on the “tastehub-users” database table, and an image is deleted from the Cloudinary storage. In addition, any existing rows that reference the deleted post from the “tastehub-comments” and “tastehub-likes” table are removed.

Follow User Test Case

- This test case was developed to ensure the fulfillment of requirement ID #16 in the RTM.
- Executed by sending a HTTP POST request to the “Follow User” lambda function using a mock JSON data passed into the body of the request.
- Status code: 200 and success message are returned. A new row is added to the “tastehub-follows” database table and the numberOfFollowing and numberOfFollowers columns are updated on the “tastehub-users” table.

Get Followers Test Case

- This test case was developed to ensure the fulfillment of requirement ID #18 in the RTM.
- Executed by sending a HTTP GET request to the “Get Followers” lambda function using a mock query parameter.
- Status code: 200, success message, and a JSON object of a list of followers are returned.

Get Following Test Case

- This test case was developed to ensure the fulfillment of requirement ID #18 in the RTM.
- Executed by sending a HTTP GET request to the “Get Following” lambda function using a mock query parameter.
- Status code: 200, success message, and a JSON object of a list of following users are returned.

Unfollow User Test Case

- This test case was developed to ensure the fulfillment of requirement ID #17 in the RTM.
- Executed by sending a HTTP DELETE request to the “Unfollow User” lambda function using a mock query parameter.
- Status code: 200 and success message are returned. An existing row is removed from the “tastehub-follows” database table, and the numberOfFollowing and numberOfFollowers columns are updated on the “tastehub-users” table.