Waste To Taste

Product Design Specification

Version 1.0

February 26th 2024

Delbert Li
Lucas Prifti
Saadman Choudhury

Revision History

Date	Description	Author	Comments
2/26/2024	Version 1.0	- Delbert Li - Lucas Prifti - Saadman Choudhury	First Draft of Design document submitted

Document Approval

The following Product Design Specification has been accepted and approved by the following:

Signature	Printed Name	Title	Date
DL	Delbert Li	• Team Lead • Presentation Lead • Assistant Documentation Lead • UI/Frontend Lead	4/10
LP	Lucas Prifti	• Database/Backend Lead • Presentation Lead • Note-Taker • QA Lead	4/10
SC	Saadman Choudhury	• Documentation Lead • UI/Frontend Lead, Assistant Database/Backend Lead	4/10
	Dr. Seyed Ziae Mousavi Mojab	• Professor	

Table of Contents

1. Introduction

The purpose of the Product Design Specification (PDS) Document is to comprehensively detail the design aspects of the Waste To Taste web application. This document acts as a strategic guide for the development team, stakeholders, and future contributors to ensure that the design is in alignment with the project's goals, user needs, and technical requirements as outlined in the Software Requirements Specification (SRS) document.

The PDS document aims to:

- Define the architectural framework, including technology choices, database schema, user interface design, and integration strategies, ensuring the application is scalable, maintainable, and user-friendly.
- Establish design standards and guidelines that promote consistency throughout the application's user interface, improving the user experience and facilitating ease of use.
- Detail the specific design choices for the application's features and functionalities, including screen layouts, navigation flows, and interaction models, to ensure they address the defined user needs and project objectives effectively.
- Provide a clear and comprehensive reference for the development team to implement the application as envisioned, minimizing ambiguity and ensuring that development efforts align with the project's goals.
- Facilitate effective communication and collaboration among project team members, stakeholders, and any external designers or developers involved in the project by providing a clear and shared understanding of the application's design.
- Serve as a foundational document for future iterations of the application, enabling informed decision-making when the application is updated or expanded upon.

This document is designed to be dynamic and will be updated throughout the development process as design decisions are refined and new requirements are identified. It is essential for ensuring that the Waste To Taste application not only meets the current needs of its users but is also adaptable to future enhancements and growth.

2. General Overview and Design Guidelines/Approach

2.1 General Overview of Design Philosophy

The design philosophy of the Waste To Taste revolves around user-centered design principles, focusing on creating an intuitive, engaging, and seamless user experience. Our goal is to empower users with the knowledge and tools they need to minimize food waste, save money, and enjoy cooking through a platform that is both informative and easy to navigate. The design emphasizes simplicity, accessibility, and functionality, ensuring that users of all technological and cooking skill levels can benefit from the application.

2.2 Design Guidelines and Overall Approach

- Simplicity: Maintain a clean and straightforward interface that highlights the application's core features, such as recipe management, culinary techniques, and strategic savings. The navigation should be intuitive, with a minimal learning curve for new users.
- Consistency: Use a consistent design language throughout the application, including consistent color schemes, typography, and layout structures. This consistency extends to interactive elements like buttons and forms, ensuring a coherent user experience.
- Responsiveness: Ensure the design is fully responsive, providing an optimal viewing experience across a wide range of devices (from desktop computers to mobile phones), screen sizes, and orientations.
- User Feedback and Interaction: Incorporate elements that provide immediate feedback to users, such as form validation messages and loading indicators. Interactive elements should be clearly indicated to encourage user interaction.
- Performance and Efficiency: Design with performance in mind, optimizing images and assets to ensure fast load times and smooth interactions, enhancing the overall user experience.

2.3 Assumptions, Constraints, and Standards

- **Assumptions**: It is assumed that users have a basic level of digital literacy and access to a stable internet connection. The design assumes users have varying levels of cooking expertise, from beginners to more experienced home cooks.
- **Constraints**: The project is constrained by the technological choices made (e.g., MongoDB, React.js, Node.js), which influence the design's scalability and performance. Time constraints and resource availability also impact the depth and breadth of features that can be implemented in the initial release.
- **Technological Standards**: The application will use modern web development standards, including HTML5, CSS, and JavaScript ES6+, ensuring compatibility across major web browsers.
- **Design and UI Standards**: Follow established UI design standards and best practices for web applications, ensuring a user-friendly and aesthetically pleasing interface.

This design approach, guided by these principles, guidelines, and standards, aims to create a product that is not only functional and informative but also a pleasure to use, encouraging users to return and engage with the content regularly.

3. Architecture Design

This section outlines the architecture design for the Waste To Taste web application, focusing on the system's hardware, software, security, communication, and performance architectures. It provides a blueprint for how the application is structured and operates within its ecosystem, detailing the interactions between its components and external entities.

3.1 Hardware Architecture

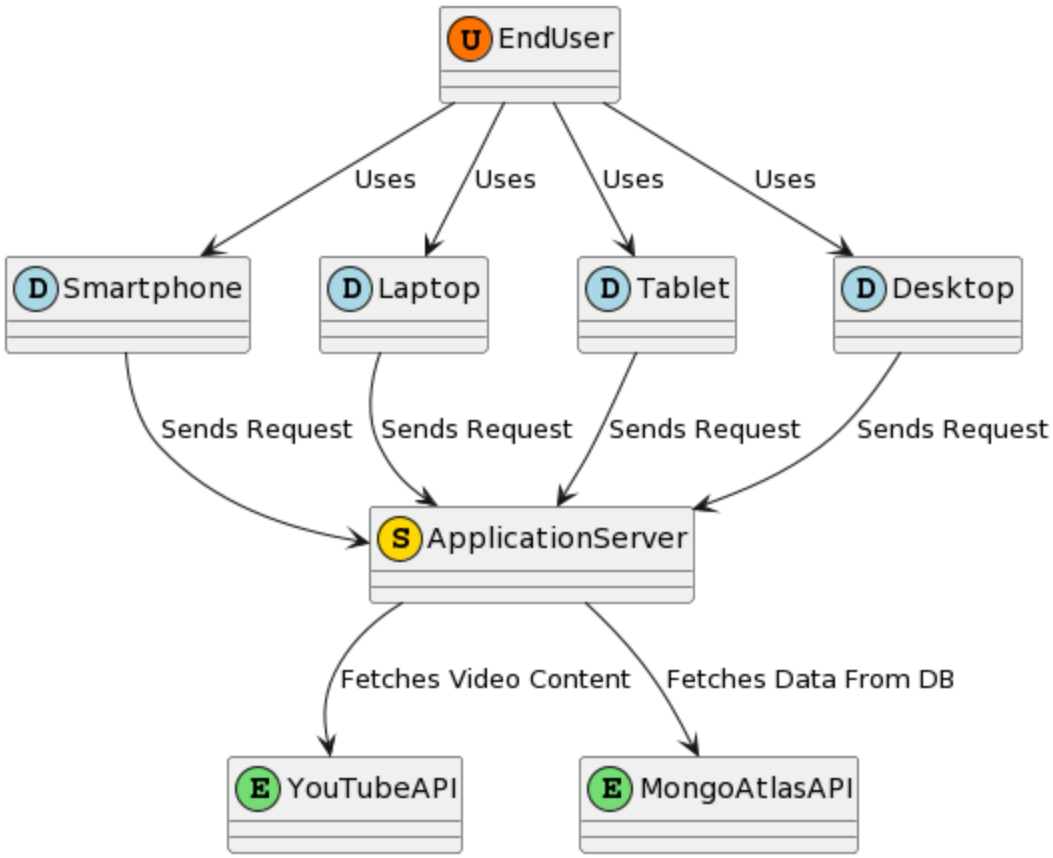
- **Computing Resources:**
    - The application leverages cloud-based infrastructure (AWS EC2 instances) optimized for compute performance and scalability. These instances are configured with multi-core processors and scalable RAM options through the cloud, to efficiently handle concurrent user requests and background processing tasks.

- **Peripheral Devices:**
    - Users interact with the application via standard computing devices equipped with modern web browsers. These include:
        - Smartphones,
        - Tablets,
        - Laptops,
        - Desktops,
    - Our goal being to ensure wide accessibility without the need for specialized hardware.

- **Data Storage:**
    - MongoDB Atlas is used for cloud-based data storage, offering high availability, automatic scaling, and data redundancy. This NoSQL database service is chosen for its flexibility in handling schema-less data, facilitating rapid development and iteration of data models.

- **Connection Overview:**
    - Devices connect to the application through secure, encrypted channels over the internet. The cloud infrastructure is designed for high availability, with data centers spread across multiple locations to ensure close proximity to users worldwide.

Hardware Architecture Diagram:



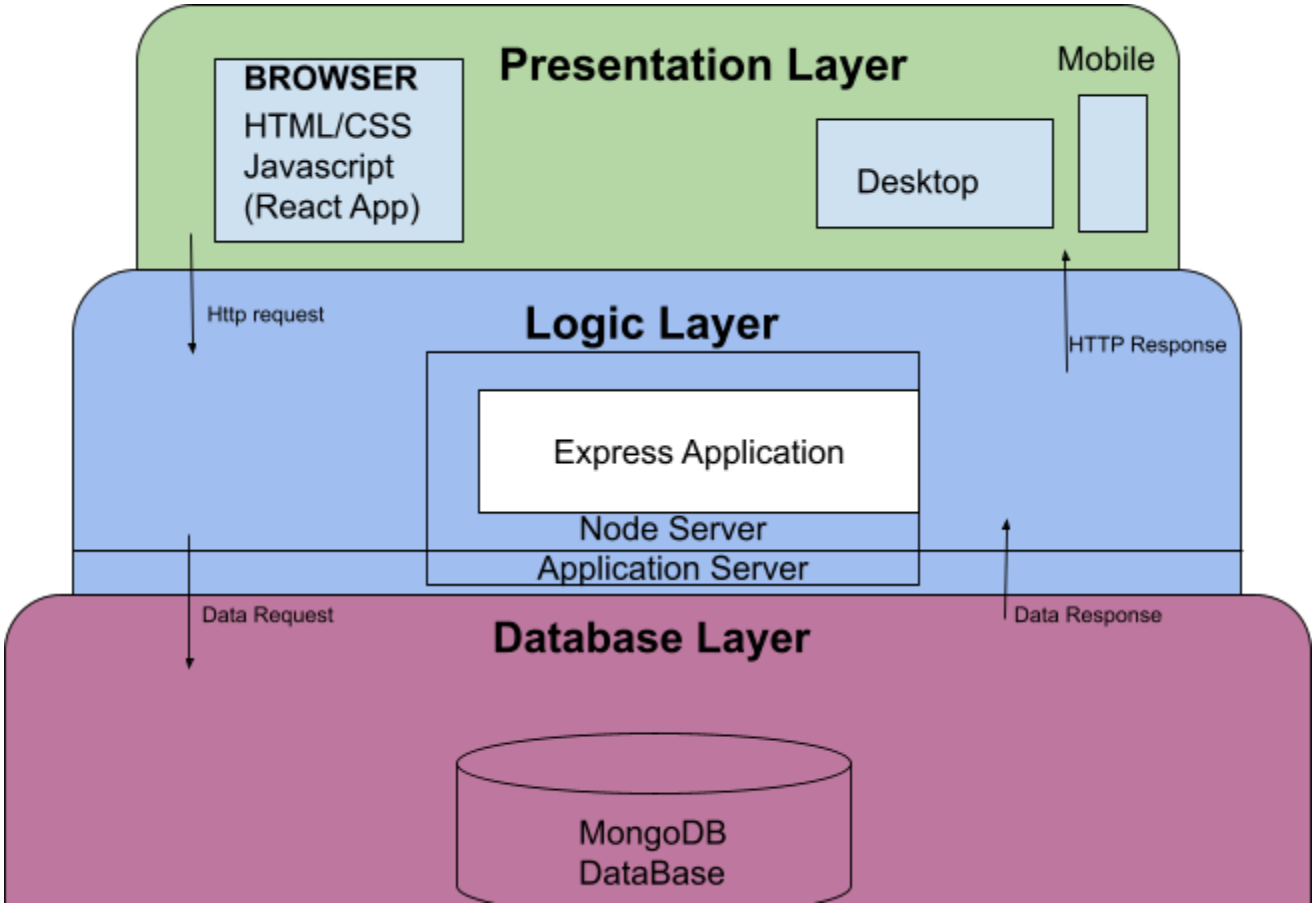
3.2 Software Architecture

- **Architectural Structure: 3 - Tier Architecture Model**
    - Our web application adopts a 3-tier architectural model, designed to enhance modularity, scalability, and maintainability. The 3-Tier architecture divides the application into three distinct layers: the Presentation Layer, the Server Logic Layer, and the Database Layer. Each layer has its own set of responsibilities, maintaining a clear separation of concerns.
- **Presentation Layer (React):**
    - This layer is the front end of the application, handling all user interactions within the browser. It's developed using React, a JavaScript library renowned for its efficiency and flexibility in building dynamic user interfaces. React's component-based approach allows for modular UI development, making the application more maintainable and scalable. The presentation layer communicates with the business logic layer via HTTP requests to fetch or manipulate data.

- **Server Logic Layer (Node.js & Express.js):**
    - Serving as the application's foundation, this layer processes user commands, performs server logic, and handles data exchange between the presentation and database layers. Built on Express, a web application framework for Node.js, it facilitates the creation of robust APIs and web servers. This layer includes:

        - **API Endpoints:** Defined using Express to handle client requests and responses.
        - **Middleware:** For processing requests, handling authentication, and interacting with the database.
        - **Server Logic:** The core functionalities like user registration, authentication, and recipe management are implemented here.
- **Database Layer (MongoDB Atlas):**
    - This layer is responsible for data persistence and management. MongoDB, a NoSQL database, is used for its scalability and flexibility in handling schema-less data, making it ideal for storing user profiles, authentication tokens, recipe details, and more. The business logic layer interacts with the database through Mongoose, an ODM (Object Data Modeling) library for MongoDB and Node.js, facilitating data validation, query building, and business logic execution.

Software Architecture Diagram:



3.3 Security Architecture

- **Data Protection:**
    - Implements TLS 1.3 for all communications, ensuring data is encrypted in transit. Data at rest, including sensitive user information and authentication tokens, is encrypted using AES-256 encryption standards within MongoDB Atlas.

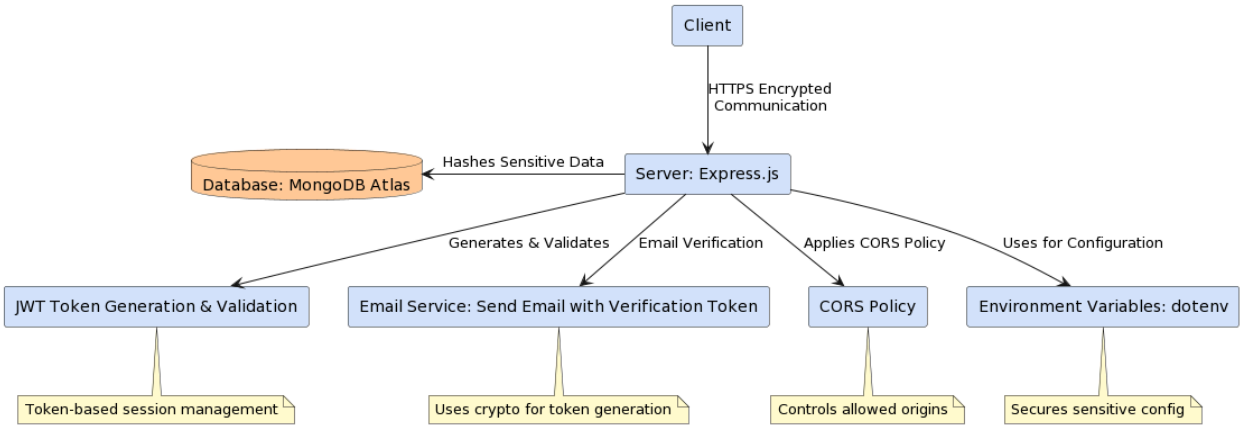
- **Key Security Components:**
    - **Access Control:** Leverages JSON Web Tokens (JWT) for managing user sessions and access permissions.

    - **Data Protection:** Utilizes field-level encryption for sensitive user data in the database. HTTPS Everywhere is enforced to secure data exchange between the client and server.

    - **Network Security:** Employs AWS WAF (Web Application Firewall) to protect against common web exploits and DDoS attacks, ensuring the application's availability and security.

    - **Application Security:** Incorporates input validation frameworks to sanitize and validate user input, preventing SQL injection and XSS attacks. Regular code audits and dependency checks shall be conducted to identify and remediate vulnerabilities.

Security Architecture Diagram:



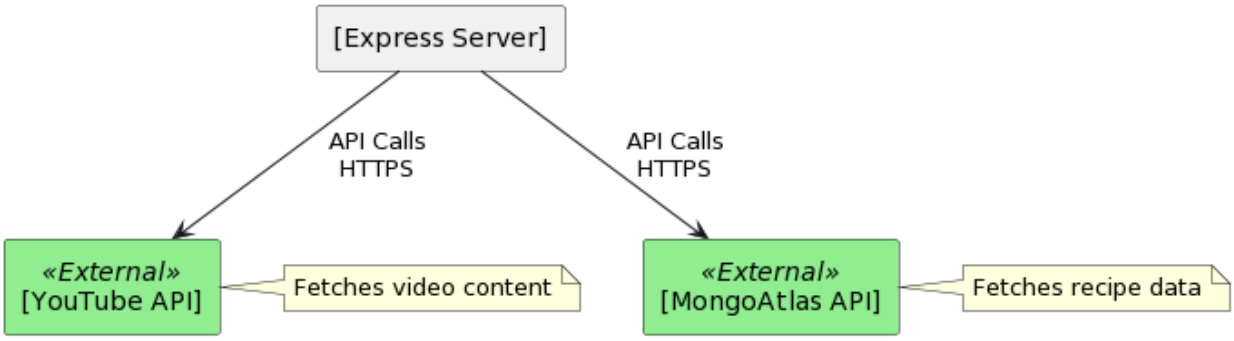
3.4 Communication Architecture

- **Communication Protocols:**

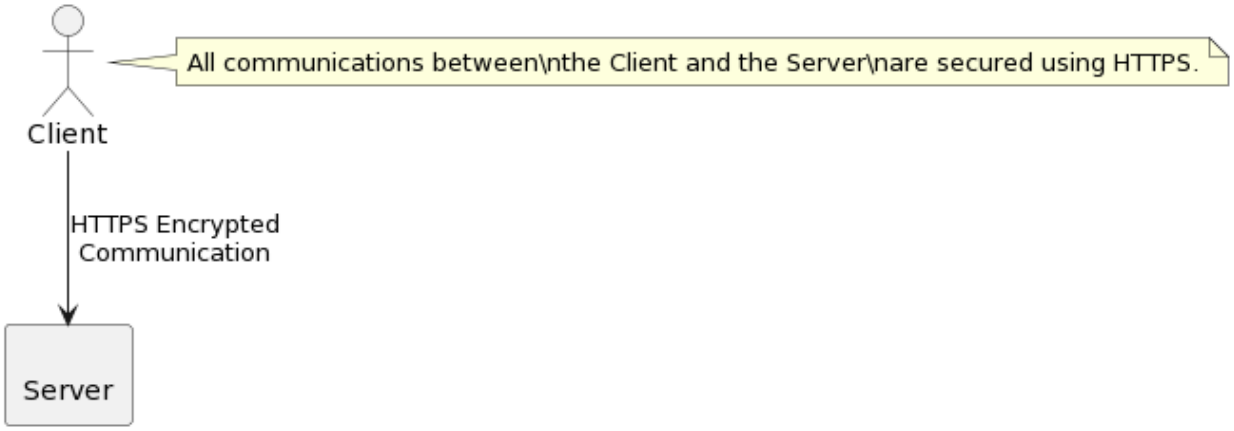
- Adopts HTTPS as the primary protocol for secure web communication, enforcing strict transport security. RESTful APIs facilitate microservices communication, utilizing JSON for lightweight data interchange.

- **Data Exchange Standards:**
  - JSON is chosen for its simplicity and efficiency in data representation, supporting a wide range of data types and structures. API communication standards follow the OpenAPI Specification for clear documentation and client generation.

External API integration



Client-Server Communication Diagram



3.5 Performance

The performance of the Waste To Taste web application is critical for ensuring a seamless user experience. The performance architecture is designed to efficiently manage user requests and data processing, ensuring high responsiveness and stability under various load conditions. The key components of our performance strategy include:

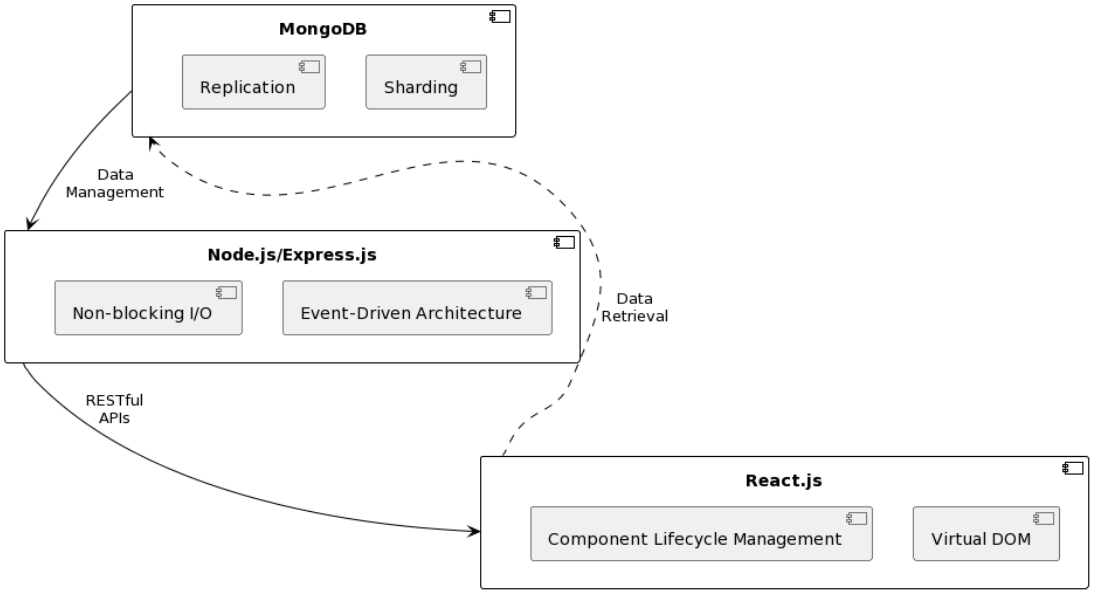
Scalability and Responsiveness:

- The application leverages MongoDB's built-in scalability features, including sharding and replication, to distribute data across multiple nodes, ensuring quick data access and high availability.
- React's virtual DOM minimizes the number of DOM manipulations required, significantly improving front-end rendering performance and ensuring smooth user interactions even on lower-end devices.
- The application's backend, built with Node.js and Express, utilizes an asynchronous, non-blocking I/O model that efficiently handles concurrent user requests, maintaining a high level of responsiveness as the user base grows.

Concurrency Handling:

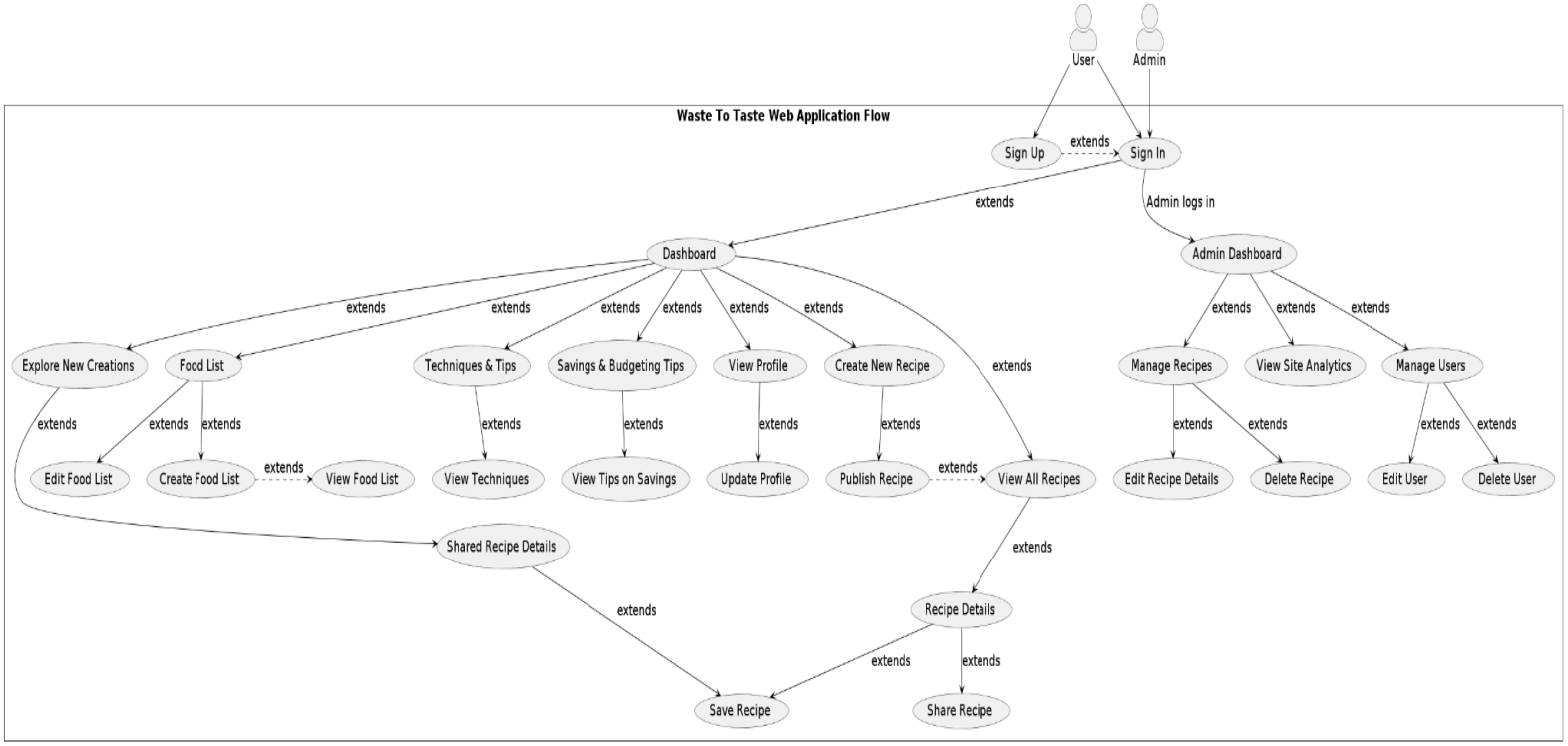
- Node.js's event-driven architecture is at the core of our application's ability to handle multiple simultaneous connections efficiently. This approach minimizes overhead and maximizes throughput, ensuring that user actions receive quick and consistent responses.
- The application's design includes thorough testing of concurrent operations, identifying and resolving potential bottlenecks in both the frontend and backend. This ensures that the application can maintain optimal performance even during peak usage periods.

Performance Architecture Diagram:



4. System Design

4.1 Use-Case Diagram



4.2 Use-Cases

USE CASE ID:	UC-01		
USE CASE NAME:	User Registration		
Created by:	Delbert Li	**Updated by:**	N/A
Date Created:	2/23/2024	**Revision Date:**	N/A
Actors:	New User		
Description:	This use case describes what happens when the user first accesses the site and goes through the process of creating an account. This allows for new users to register for an account by providing their personal information ensuring that they can access and utilize all of the application's features.		
Trigger:	User accesses the website and clicks on sign up to create an account.		
Preconditions:	● User does not have an account ● User must have valid email ● System servers must be operational		
Post Conditions:	● The user's account is created ● Verification will be sent to the users email ○ User verifies email ● Users can log in to the Web App and then can utilize all features of the Waste To Taste.		

Normal Flow:	1. New Users navigate to the website's main page.
	2. User then clicks on the sign up button.
	3. They enter their required information(There are criteria for each input).
	  a. Email
	  b. First name, last name
	  c. Password
	4. The Web App will then validate the user input for sign-up
	  a. After sign up is successful, an email verification will be sent to the new user's email to be verified.
	5. User clicks on the verification and validates the email, and then the web app system creates the account.
	6. User logs in to the system successfully and can utilize all features.		

Alternative Flows:	None		
Exceptions:	● If the email is already in use, the system informs the user and asks for a new email.		

	○ Users can then utilize a “forgot password” function to make a new password with a link sent to their email. ● If the email doesn’t meet the proper criteria, it will inform the user to fix the format of the email. ● If the password doesn’t meet the proper criteria it will inform the user to create a new password or to modify the current one to fit the criteria.
Frequency of Use:	● Only needed once for new users
Assumption:	● There is active internet connection.

USE CASE ID:	UC-02		
USE CASE NAME:	User Login		
Created by:	Delbert Li	**Updated by:**	N/A
Date Created:	2/23/2024	**Revision Date:**	N/A
Actors:	User		
Description:	This use case authenticates a user’s credentials to provide access to their personalized application environment.		
Trigger:	A registered user wants to log into their account.		
Preconditions:	● User has a verified account ● System servers must be operational		
Post Conditions:	● The user gains access to their account dashboard and application features		
Normal Flow:	1. User access the web app 2. They navigate to the login page via the login button 3. They enter their credentials 4. System validates their credentials 5. After successful validation the system grants access to the users dashboard		
Alternative Flows:	None		
Exception:	● If credentials are incorrect the system displays an error and offers the user to try again or to reset password		

Frequency of Use:	● Regularly by users who have an account
Assumption:	● There is active internet connection.

USE CASE ID:	UC-03		
USE CASE NAME:	Email Verification		
Created by:	Delbert Li	**Updated by:**	N/A
Date Created:	2/23/2024	**Revision Date:**	N/A
Actors:	User, System		
Description:	This use case verifies the email address of a newly registered user to activate their account.		
Trigger:	User clicks on the email verification link sent to their email after registration.		
Preconditions:	● User has registered with a valid email and received a verification email ● System servers must be operational		
Post Conditions:	● The user's account is fully verified and activated.		
Normal Flow:	1. User receives a verification email. 2. User clicks on the verification link in the email. 3. System verifies the link and activates the account. 4. User is notified of the successful account activation.		
Alternative Flows:	● If the link is expired, the system will notify the user that their link has expired.		
Exception:	None		
Frequency of Use:	● Once per new user registration		
Assumption:	● There is active internet connection.		

USE CASE ID:	UC-04

USE CASE NAME:	Forgot Password		
Created by:	Delbert Li	**Updated by:**	N/A
Date Created:	2/23/2024	**Revision Date:**	N/A

Actors:	User
Description:	This use case provides a mechanism for users to reset their password in case they forgot it.
Trigger:	Users cannot remember their password and cannot log in, and then press the forget password button.
Preconditions:	● User has a created account with a registered email ● System servers must be operational
Post Conditions:	● User resets their password and regain their access to the account
Normal Flow:	1. Users access the login page and attempt to sign in. 2. Their password is incorrect. 3. They select “Forgot Password”. 4. They enter their email. 5. System verifies the email and senses a password reset link. 6. User clicks the link in their email and enters a new password. 7. System updates the password and the user can now log in.
Alternative Flows:	● If the email is not associated with an account the system will inform the user
Exception:	None
Frequency of Use:	● Used whenever the user forgets their password.
Assumption:	● There is active internet connection.

USE CASE ID:	UC-05		
USE CASE NAME:	Profile Management		
Created by:	Delbert Li	**Updated by:**	N/A
Date	2/23/2024	**Revision Date:**	N/A

Created:			
Actors:	User		
Description:	This use case allows users to update their profile information, including first and last name and password, ensuring their information is current and accurate.		
Trigger:	Trigger is when the user decides to update their profile information.		
Preconditions:	● System servers must be operational. ● User has a verified account and is logged in.		
Post Conditions:	● The user's information is updated in the database and the user will receive confirmation on their update.		
Normal Flow:	1. User logs into the Waste To Taste web app. 2. User then navigates to their profile settings page. 3. Users then select the information they want to update. 4. User then enters new information and submits new updates. 5. System then validates the new information against system criteria a. Password length and special criteria 6. System updates the information in database 7. User receives a confirmation notification		
Alternative Flows:	None		
Exception:	● If a user fills in information that doesn't fit the criteria such as first name, last name or password(name not correct or password doesn't have enough characters) It will notify the user the changes will not be submitted until information is correct.		
Frequency of Use:	● Anytime a user needs to change their password		
Assumption:	● There is active internet connection. ● User understands the requirements for password and email criteria.		

USE CASE ID:	UC-06		
USE CASE NAME:	Personal Dashboard		
Created by:	Delbert Li	**Updated by:**	N/A

Date Created:	2/23/2024	**Revision Date:**	N/A
Actors:	User, System		
Description:	This use case provides users with a personalized dashboard upon login, that will display saved recipes, and recommended content.		
Trigger:	User successfully logs into the account.		
Preconditions:	● System servers must be operational. ● Users must have a verified account and be logged in.		
Post Conditions:	● Users are presented with their dashboard viewing their account information, shared recipes and some informational content.		
Normal Flow:	1. User logs into their account 2. The system retrieves user's shared recipes and account information 3. Users are presented with their personalized dashboard containing their retrieved information. 4. User interacts with their dashboard a. Such as selecting a recipe to view details. b. Editing their profile information.		
Alternative Flows:	● There are no shared recipes on the website, so under the "shared recipes" card, the user will see that there aren't any recipes at the moment.		
Exception:	None		
Frequency of Use:	● Utilized every time the user logs in		
Assumption:	● There is active internet connection. ● User has recipes saved ● User has interacted with the platform in some way		

USE CASE ID:	UC-07		
USE CASE NAME:	Admin Panel		
Created by:	Delbert Li	**Updated by:**	N/A
Date Created:	2/23/2024	**Revision Date:**	N/A

Actors:	Admin
Description:	This use case allows for the administrators to manage the application, including user accounts, content moderation, and recipe moderation.
Trigger:	The trigger for this use case is if the Admin needs to perform management tasks
Preconditions:	● System servers must be operational ● Admin needs to be authenticated with valid credentials(admin email and password)
Post Conditions:	● Admin successfully performs the necessary management actions
Normal Flow:	1. Admin logs into the system using pre-set admin credentials and goes to the admin panel with their credentials 2. Admin will perform their necessary tasks. a. such as(delete users, delete recipes, managing recipes, users) 3. System will have the tools and interfaces for the admins to perform their task. 4. Admin completes the tasks and logs out.
Alternative Flows:	None
Exception:	● If there is an attempted unauthorized access, the system will log the attempt and deny access.
Frequency of Use:	● Is as needed for administrative tasks
Assumption:	● There is active internet connection. ● Admins have an understanding of the functions for the platform

USE CASE ID:	UC-08		
USE CASE NAME:	Creating a New Recipe		
Created by:	Delbert Li	**Updated by:**	N/A
Date Created:	2/23/2024	**Revision Date:**	N/A

Actors:	User
Description:	This use case will allow for users to create a new recipe. This will allow for users to contribute to the platform by submitting their recipes complete with photos, videos, and detailed cooking instructions with a checklist format.
Trigger:	Trigger is user decides to create a new recipe
Preconditions:	● System servers must be operational ● User is logged in and understands the recipe creation process
Post Conditions:	● The recipe is successfully created and saved to their ViewAllRecipe paged and will be added to the platform if they choose to click share
Normal Flow:	1. User logs in and navigates to the “Create New Recipe” page for the web app. 2. Users begin filling out the recipe creation form including the recipe title, ingredients, instructions, tags, allergens, and youtube video if they have one. a. Ingredients and instructions will be in a checklist format that’ll allow for users to check off if they finished an instruction or have utilized an ingredient. b. They can also add tags that can be searched for easy finding. (healthy, unhealthy, gluten-free, etc) 3. Ser reviews their submission. 4. User submits the recipe for addition to the platform. 5. System validates the submission for completeness and adherence to the guidelines. 6. After the recipe is saved the recipe will appear in users “View All Recipes” and user will receive a confirmation on the saving.
Alternative Flows:	● The system detects missing/improper information during submission and will prompt the user to correct the issue before saving the recipe can be done. ● The user decides to save the recipe after the creation, and wants to make changes so they can edit the recipe later.
Exception:	None
Frequency of Use:	● Depending on the user.
Assumption:	● There is active internet connection. ● Users have a necessary understanding of cooking that can create a coherent recipe ○ Or has a recipe saved somewhere else that they can copy and save.

USE CASE ID:	UC-9		
USE CASE NAME:	Recipe Saving		
Created by:	Delbert Li	**Updated by:**	N/A
Date Created:	2/23/2024	**Revision Date:**	N/A
Actors:	User		
Description:	This use case allows for users to save their favorite recipes for easy access later, within their personal dashboard and ViewAllRecipe page.		
Trigger:	Users find a recipe they want to save from ExploreNewCreations and CreateNewRecipe.		
Preconditions:	● System servers must be operational ● User is logged in and has found a recipe in ExploreNewCreations and CreateNewRecipe		
Post Conditions:	● The recipe is saved to the user's profile		
Normal Flow:	1. User navigates to ExploreNewCreations and finds a recipe they like 2. They view the recipe and click the "Save Recipe" button 3. System verifies if the recipe isn't already saved to the user's profile 4. System then adds the recipe to the user's ViewAllRecipes page. 5. They receive confirmation that the recipe was saved. 6. They navigate to the ViewAllRecipes page to verify that it is saved there and can be accessed for use.		
Alternative Flows:	1. Users can also CreateANewRecipe and save the recipe instead of going to ExploreNewCreations. 2. If someone sends a recipe from Waste To Taste to the user and the user clicks on it to open it in the Waste To Taste App, the user will then directly access the recipe and can save it from there		
Exception:	● Recipe is already saved, and the system notifies the user that it's already saved.		
Frequency of Use:	● User-dependent on whether users find recipes they wish to save.		
Assumption:	● There is active internet connection.		

USE CASE ID:	UC-10		
USE CASE NAME:	Recipe Management		
Created by:	Delbert Li	**Updated by:**	N/A
Date Created:	2/23/2024	**Revision Date:**	N/A

Actors:	User
Description:	This use case allows for users to create, view, edit, or delete their own(or saved) recipes allowing for customization
Trigger:	User decides to manage their recipes
Preconditions:	● System servers must be operational ● User is logged in and has recipes to manage or wants to create a new one.
Post Conditions:	● User recipes actions are successfully reflected in their profile.

Normal Flow:	1. User navigates to their Recipes page
	2. User selects the action they want to perform
	  a. VieweAllRecipes page: View, Edit, Delete
	  b. CreateNewRecipe page: Create
	3. System provides the necessary interface for the selected action.
	  a. User clicks on view recipe, the clicks the yellow edit button to edit
	  i. There is a red delete button in the editing page to allow the user to delete a recipe.
	4. User completes the action and the system updates accordingly.
	5. User receives a confirmation of the action performed.

Alternative Flows:	None
Exception:	None
Frequency of Use:	● User-Dependent on when they want to manage their recipes
Assumption:	There is active internet connection.

USE CASE ID:	UC-11		
USE CASE NAME:	Managing Foodlist		
Created by:	Delbert Li	**Updated by:**	N/A
Date Created:	2/23/2024	**Revision Date:**	N/A

Actors:	User
Description:	This use case allows for the user to curate and manage their personalized libraries of recipes similar to music playlists. Users can create multiple Foodlists to categorize recipes based on personal preference.
Trigger:	Users want to categorize and organize their own recipes into personalized lists.
Preconditions:	● System servers must be operational ● User must be logged into their account ● User must have recipes saved or created(and saved) into their ViewAllRecipes
Post Conditions:	● User then successfully creates, edits, or deletes Foodlists ● Recipes are categorized according to the users preferences
Normal Flow:	1. User navigates to Foodlist section within the web app 2. To create a new Foodlist the user selects “Create New Foodlist” option. 3. User enters a name for the Foodlist (e.g. Guilty meals, healthy meals) and optionally adds a description. 4. Users add recipes to the Foodlist to form their saved/created recipes. 5. User reviews and confirms the creation of the foodlist. 6. System saves the Foodlist and updates the user’s profile to include the new list. 7. User receives a confirmation that the Foodlist has been created.
Alternative Flows:	● Users select an existing Foodlist to add or remove recipes or to change the list’s name or description. ● User decides to delete an entire Foodlist. System will ask a “are you sure” before deletion.
Exception:	None
Frequency of Use:	● User-dependent based on how often they want to create a Foodlist.

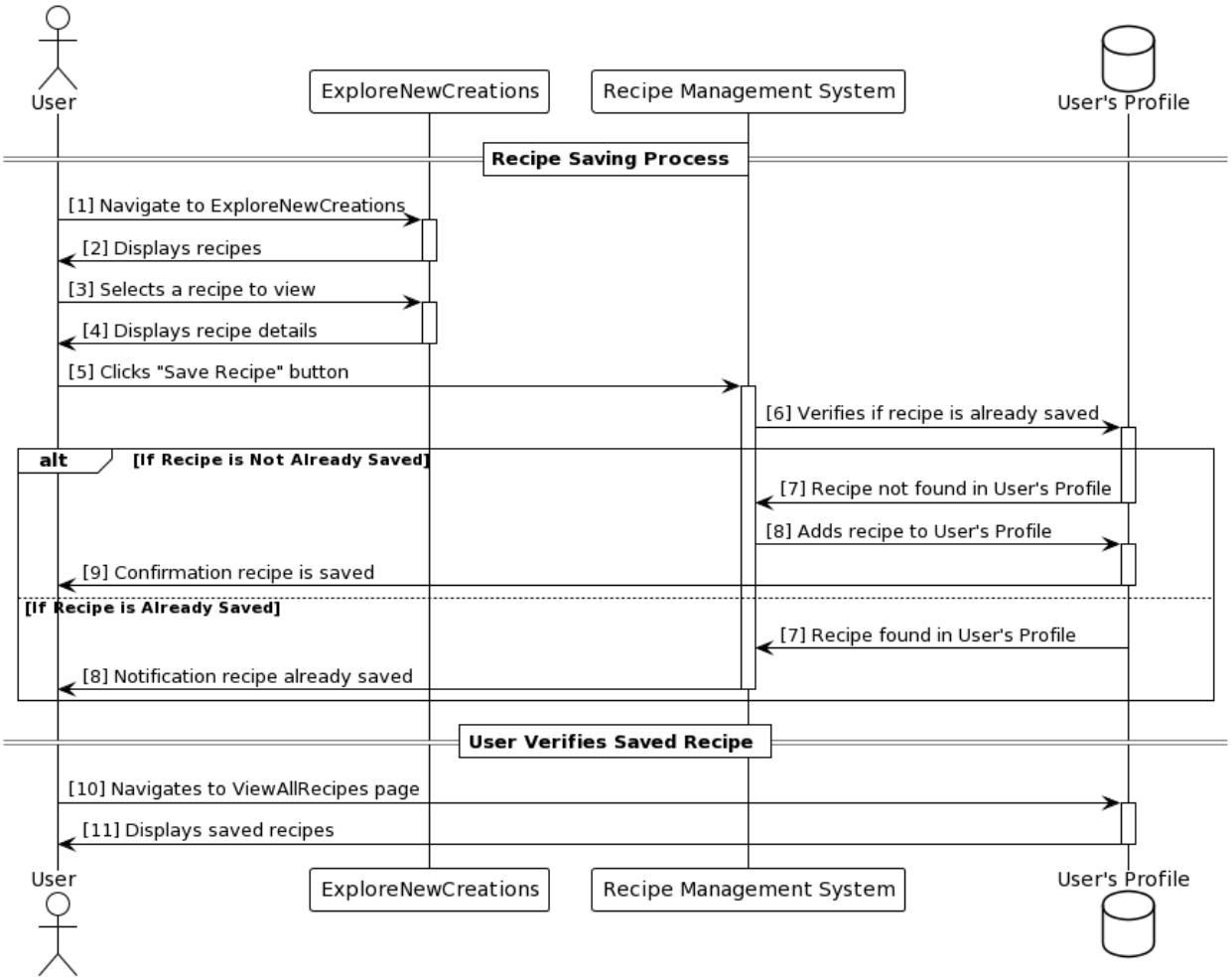
Assumption:	There is active internet connection.

USE CASE ID:	UC-12		
USE CASE NAME:	Search for Recipe		
Created by:	Delbert Li	**Updated by:**	N/A
Date Created:	2/23/2024	**Revision Date:**	N/A
Actors:	User		
Description:	This use case allows for users to search the platform for recipes based on specific criteria(ingredients, cuisine, dietary restrictions.). Users can use the search function in ViewAllRecipes(that contain their saved recipes) or in ExploreNewCreations(contains every recipe that is created and shared to the database).		
Trigger:	The trigger for this use case is when a user wants to find a recipe that matches their specific criteria.		
Preconditions:	● System servers must be operational. ● User must have an account and be logged in ● The recipe they want to find is actually created and or saved(depending on where they search).		
Post Conditions:	● User is presented with the recipe matching search criteria.		
Normal Flow:	1. User access CreateNewRecipe page. 2. User enters search criteria in the search bar a. E.g. “Healthy”, “Gluten-Free”, “Chicken” 3. System then processes the query and retrieves matching the recipe. 4. Results will appear to the user. 5. Users can then select the recipe to view more details.		
Alternative Flows:	● User is logged in and access’s their ViewAllRecipe page ○ They go to the search bar and enter their search criteria ■ E.g. “Healthy”, “Gluten-Free”, “Chicken” ○ System will process the query and retrieve the recipe. ○ Results will appear to the user and the user can view the recipe in more detail. ● No matches found, System will suggest broadening the search criteria or displays related recipes.		

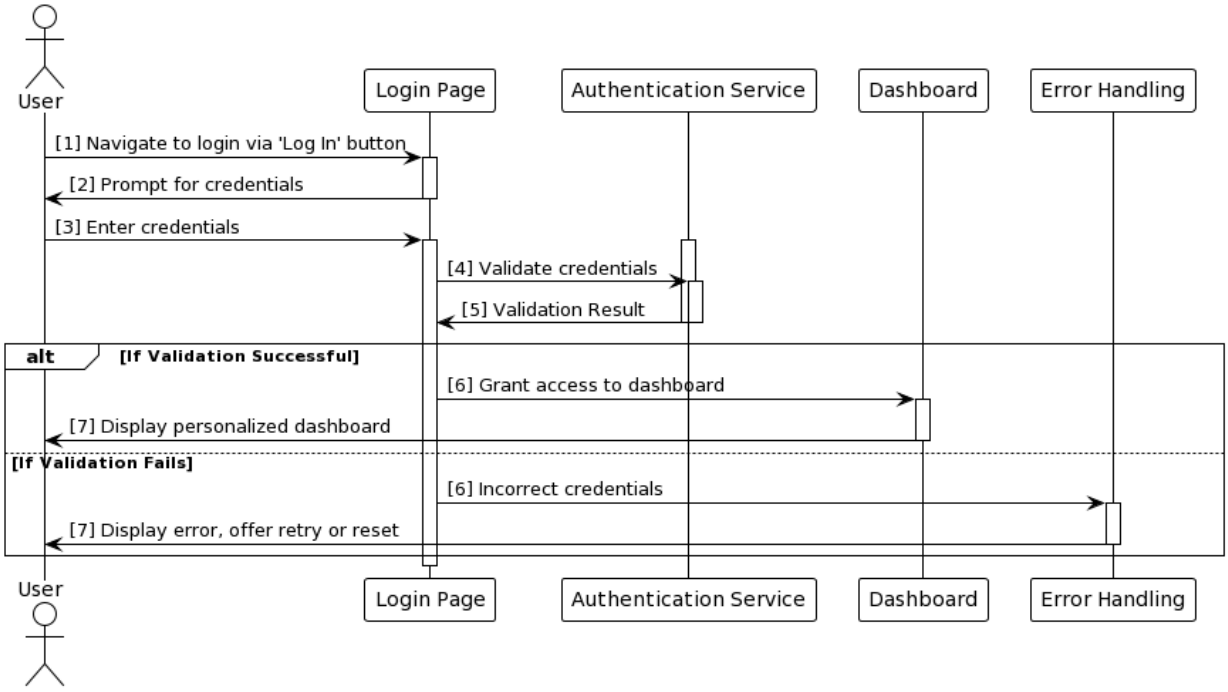
Exception:	None
Frequency of Use:	• User-dependent on what users want to search for in the recipes.
Assumption:	There is active internet connection.

4.3 Sequence Diagram

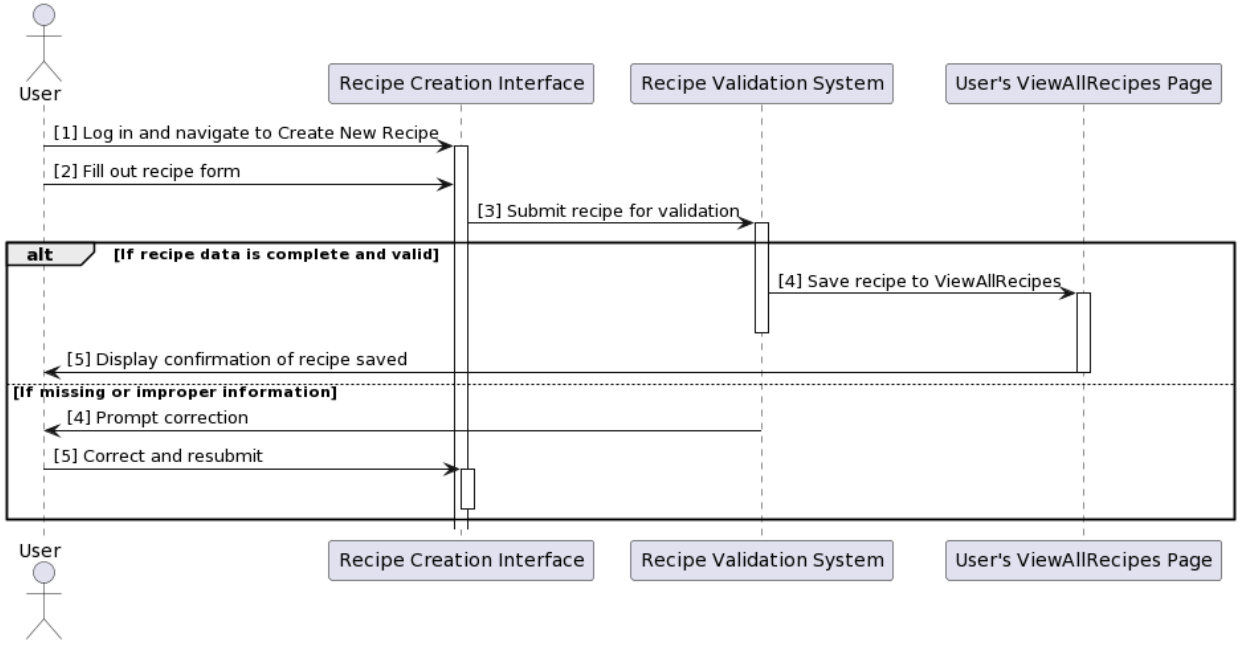
4.3.1 User Registration



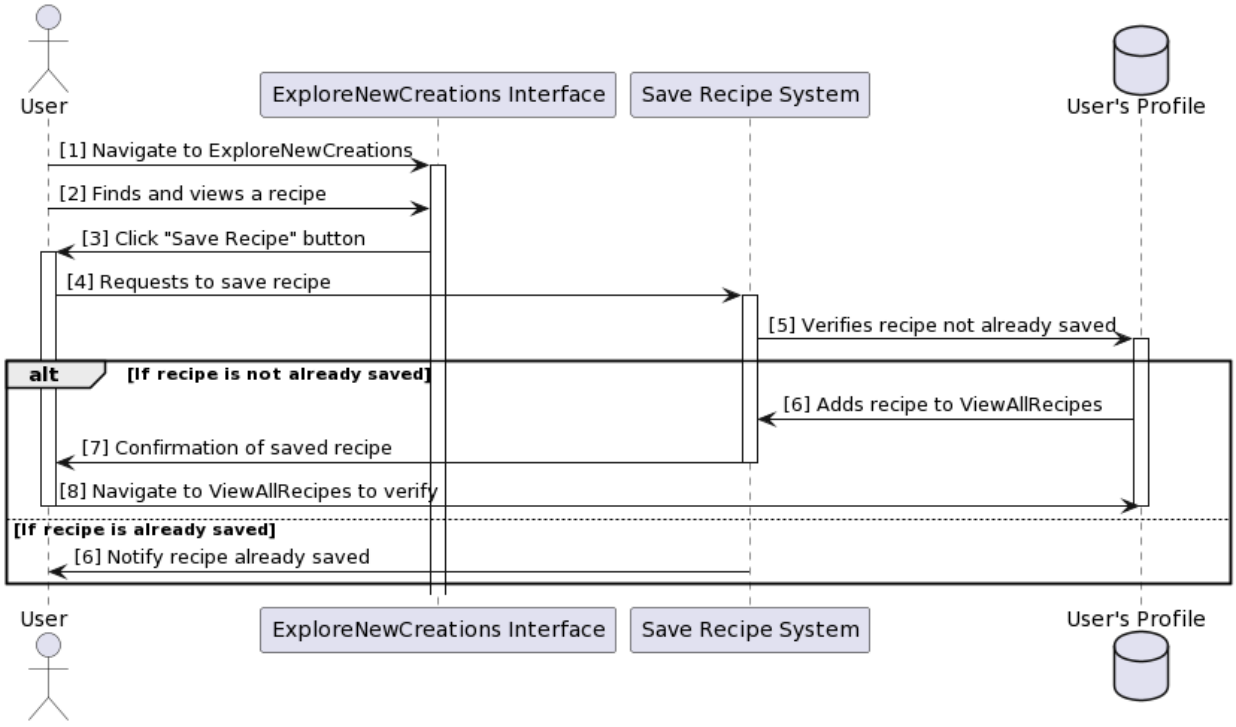
4.3.2 User Login



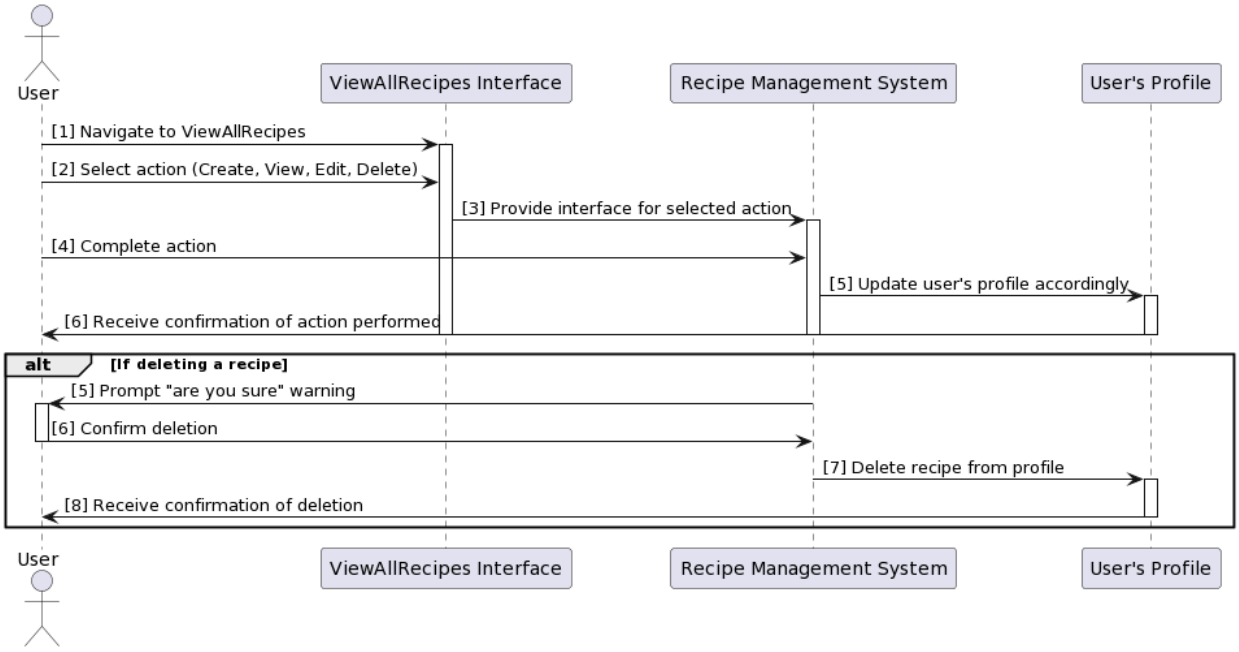
4.3.3 Create New Recipe



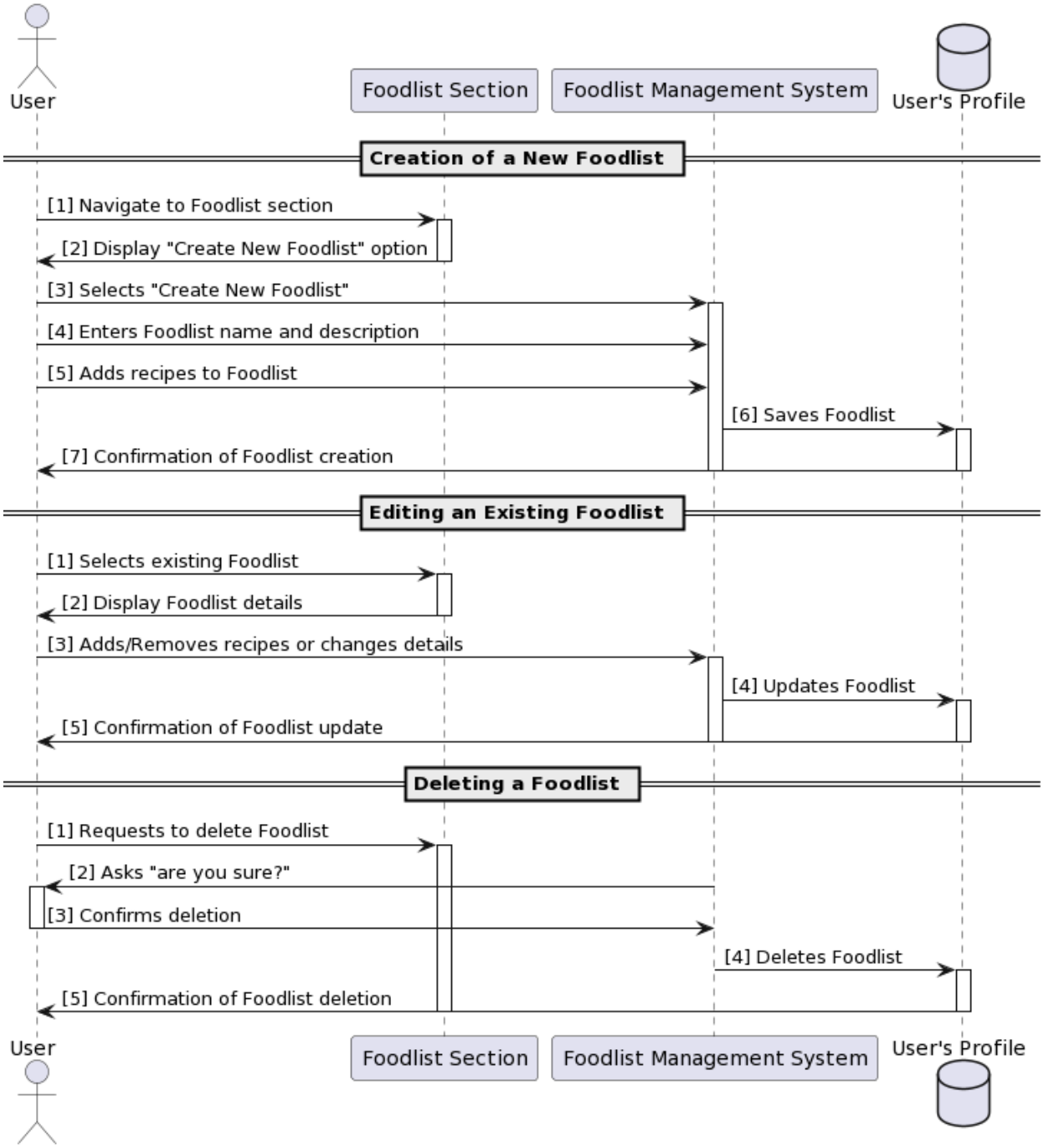
4.3.4 Recipe Saving



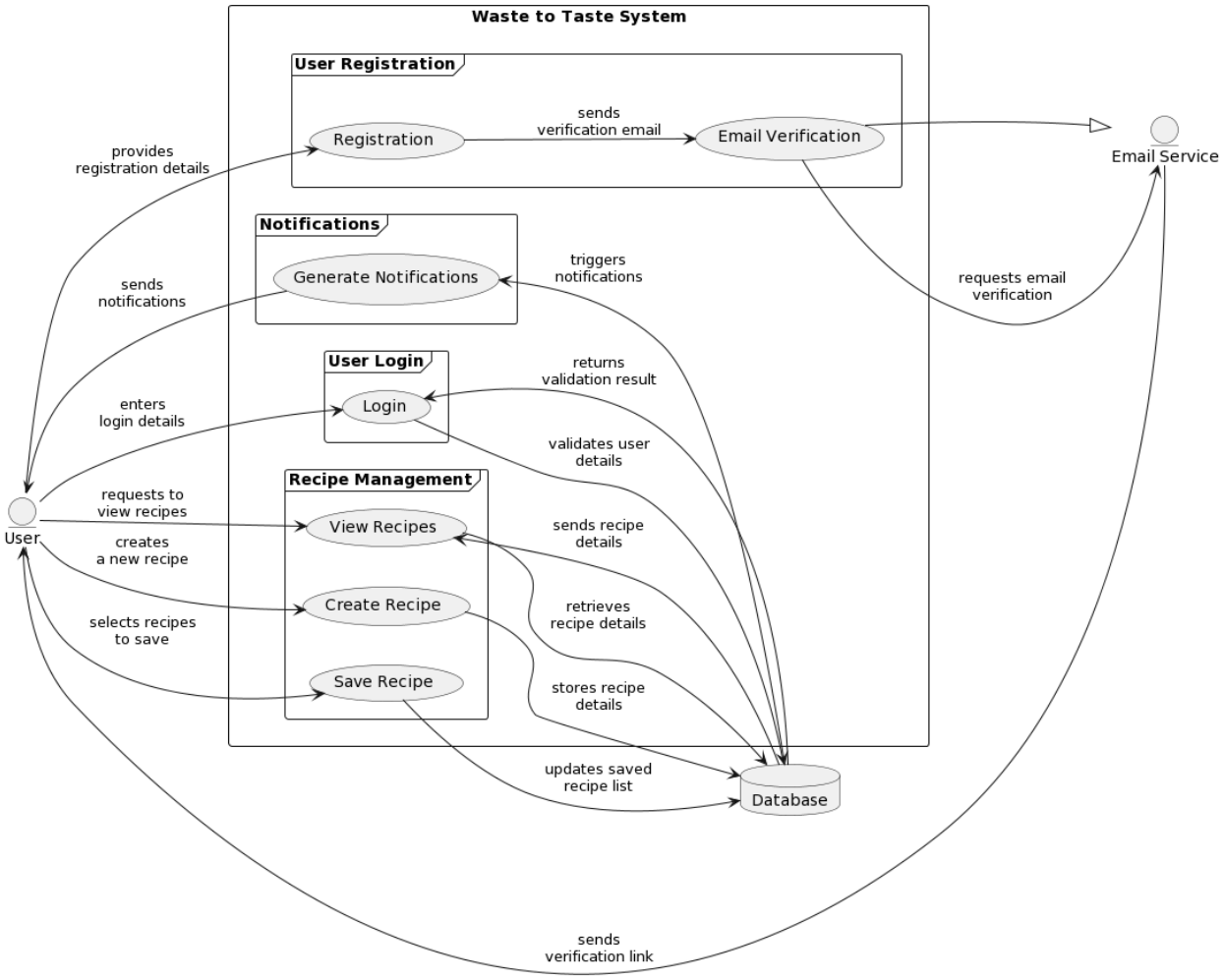
4.3.5 Recipe Management



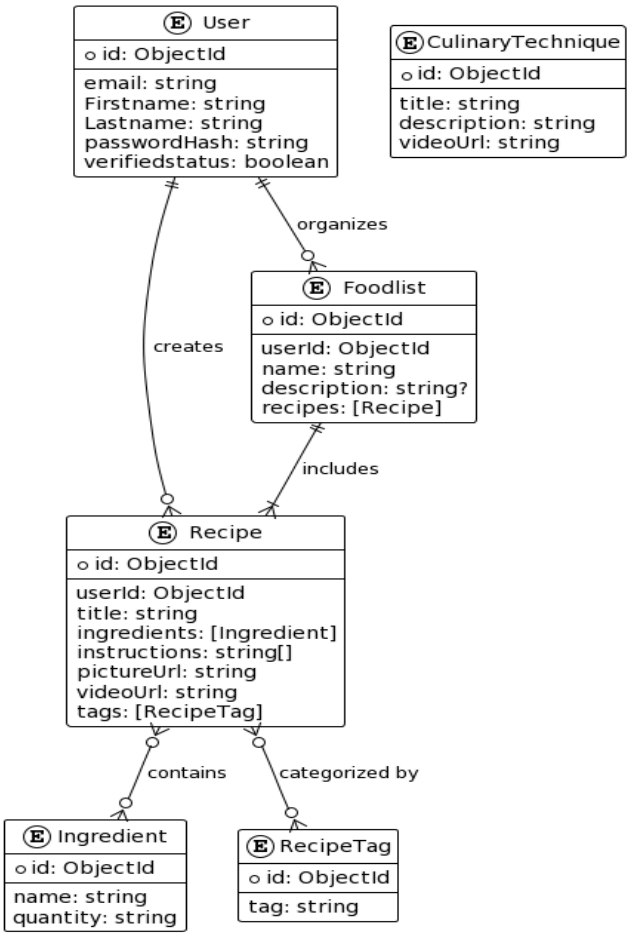
4.3.6 Managing Foodlist



4.4 Data flow diagram



4.5 Database Design



4.7 Application Program Interfaces

API	Reason of use
MongoDB Atlas	The reason why we are utilizing MongoDB for our application is because it is vital to maintain all of the data within the Waste To Taste web application. ● User Account: User features are extremely important for this application because anytime recipes are created, saved, or shared is going to be linked to the user account. ○ Dashboard: When the user logs in, they will be met with their personal dashboard that will include their saved recipes. ● Recipe: MongoDB ○ ViewAllRecipe: Any recipe that a account user saves will be kept here and will be associated with their account so

	when they log back in, they will be able to view all of their saved recipes. This requires the database to save this information and have it linked to the user account. ○ ExploreNewCreation: This is one of the main features of Waste To Taste, when a user accesses the Create New Recipe page and create and save a recipe, the recipe will save into the ViewAllRecipe page, and will be associated with the user that created it. ○ Foodlist: this feature will also require the use of a database to fulfil it's purpose.

API	**Reason of use**
YouTube	The reason why we are utilizing YouTube for our application is because we will embed videos for various purposes. ● Creating Recipes ○ When creating a recipe along with the addition of ingredients and instructions, users are allowed to insert a YouTube URL to embed a video into the recipe. ○ It allows for users to put in a video tutorial for anyone to follow a video along with the text instructions. ● Culinary Techniques/Strategic savings and storage ○ These pages of the web app will primarily contain videos and text on how to cook, save and store. ● Purpose: It is extremely easy and convenient to create and upload videos to the YouTube platform, and video media is a fantastic way of getting users to focus and interact without having them read mountains of text to learn a skill. The visual aid along with physical explanation is extremely important in teaching delicate skills such as knife handling for the users. It's also much more practical to have videos embedded into the web app via URL versus physically adding the long data dense videos into the web app itself. This will save on loading times and prevent slowdowns with the web application. The only downside is that it requires YouTube to constantly be functioning, and if the YouTube web app were to go down, the users wouldn't be able to view the embedded videos.

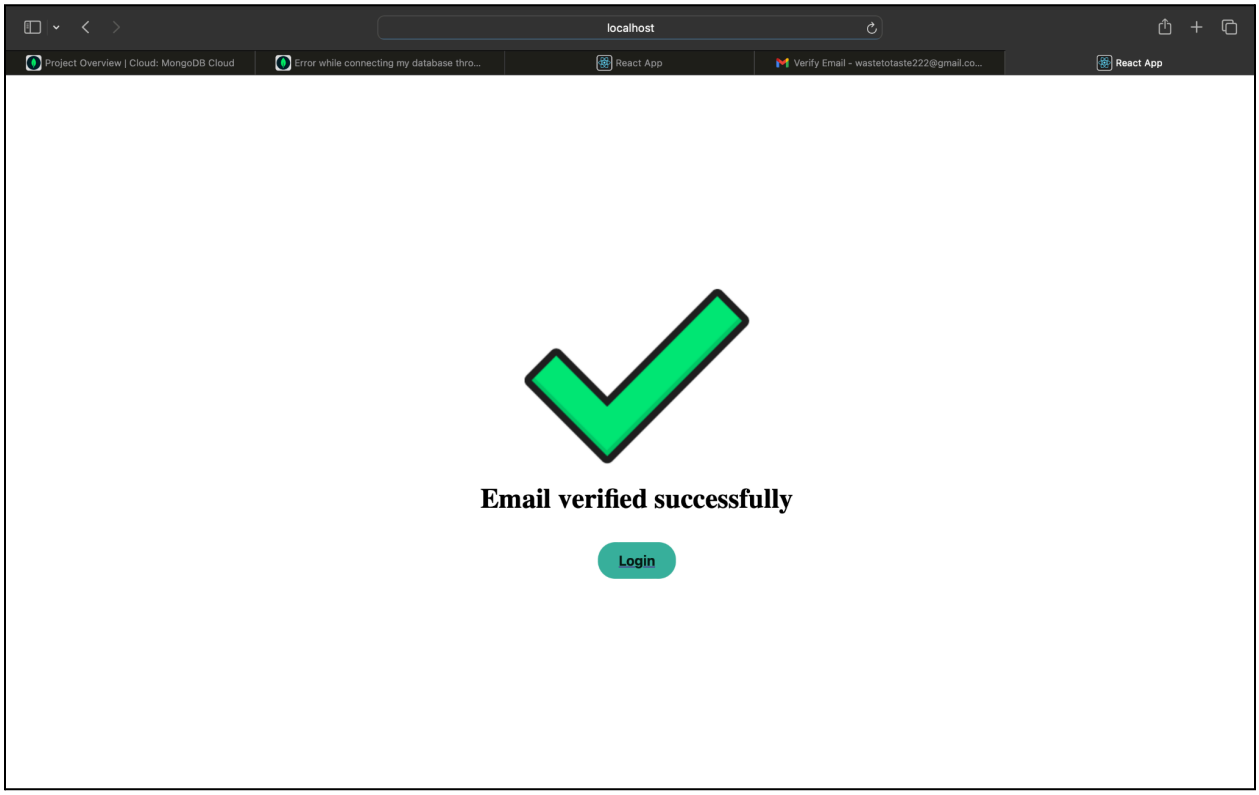
4.8 User Interface Design

4.8.1 Landing page

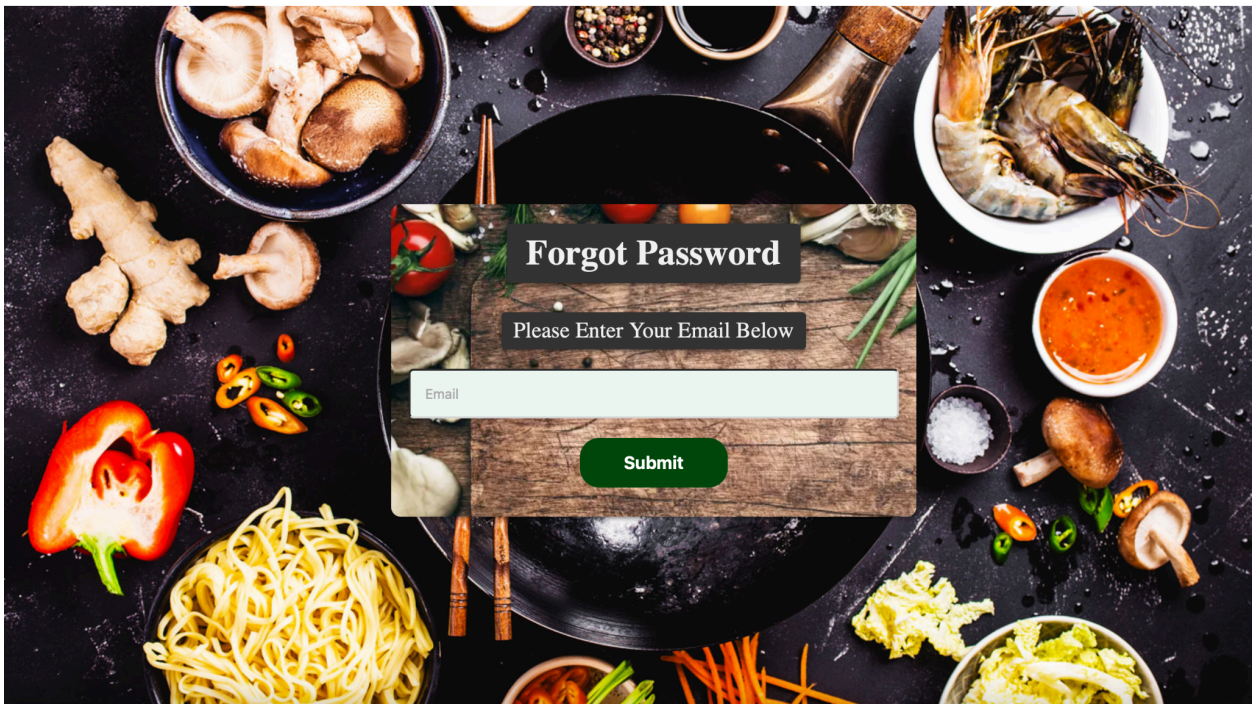
4.8.2 Registration

4.8.3 Login page

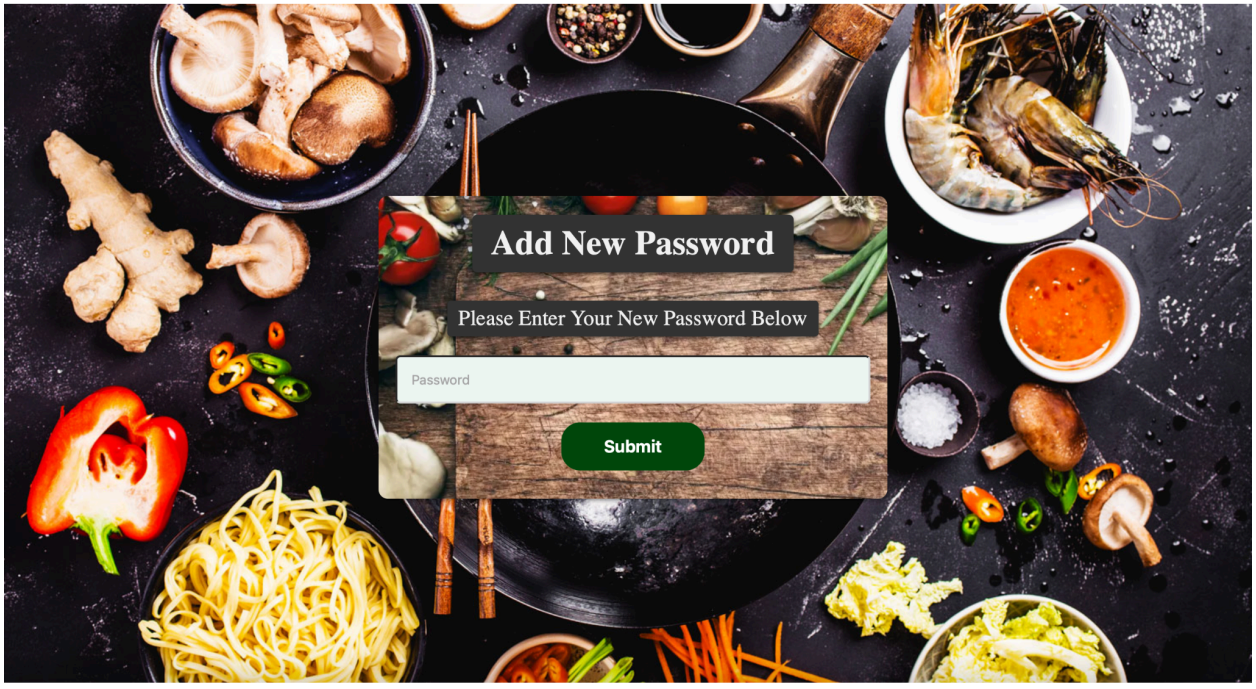
4.8.4 Email Verification



4.8.5 Forgot Password



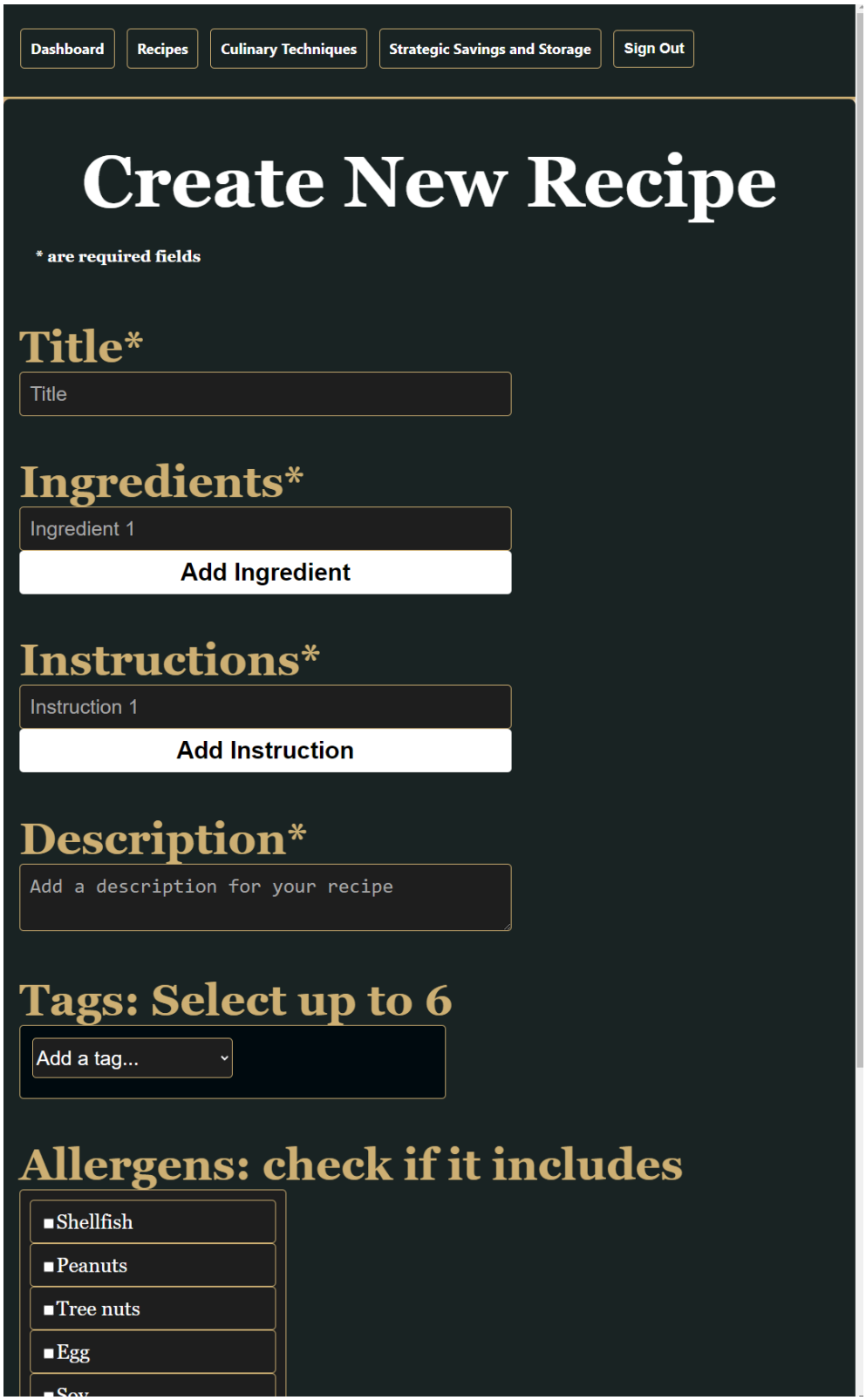
4.8.6 Reset Password



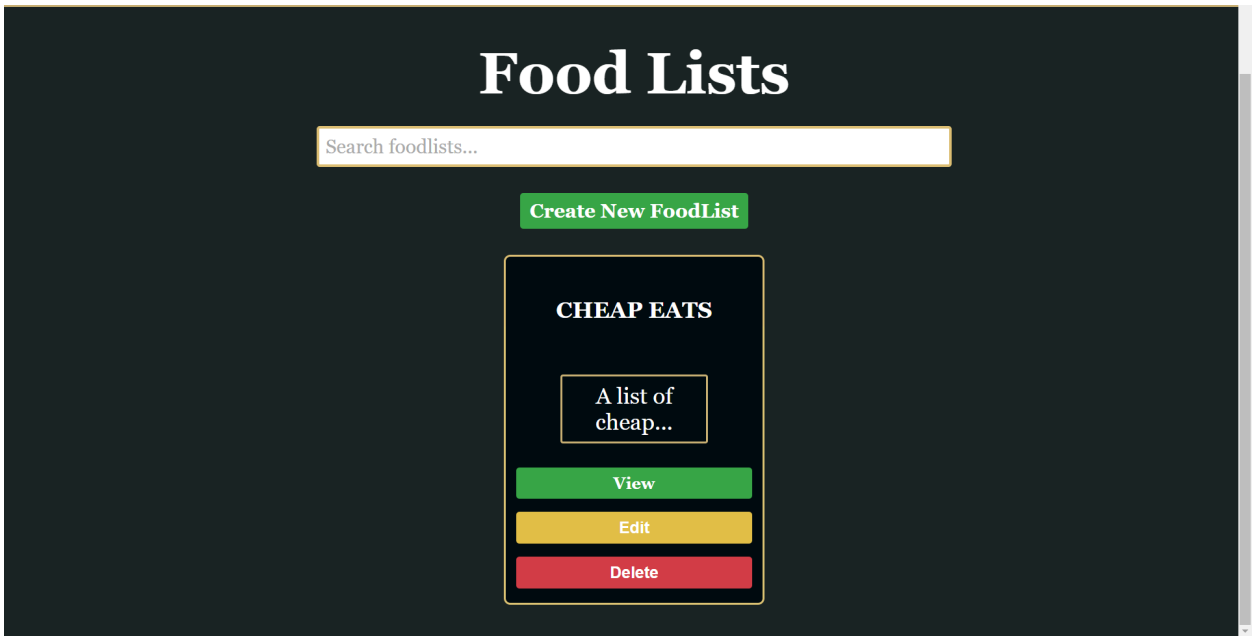
4.8.7 View All Recipes


View All Recipes
Search recipes...
Sort Options
Filter by tags:
Chicken
Chinese
Easy
Time: Short
Affordable
Gluten-Free
High-Protein
Vegetarian
Difficult
Vegan
HAINANESE CHICKEN: CRISPY CHICKEN THIGH STYLE
View Recipe
A spin on tradi...
Chicken
Chinese
Easy
Time: Short
Affordable
Share
Allergen-FREE
BOILED EGGS
View Recipe
A basic recipe ...
Gluten-Free
Easy
Time: Short
Affordable
High-Protein
Vegetarian
Share
Egg
FRIED RICE
View Recipe
A classic modif...
Chicken
Easy
Time: Short
Share
Egg
Soy
more...
WHITE RICE
View Recipe
The lifecycle o...
High-Protein
Difficult
Vegan
Vegetarian
Easy
Affordable
Share
Allergen-FREE

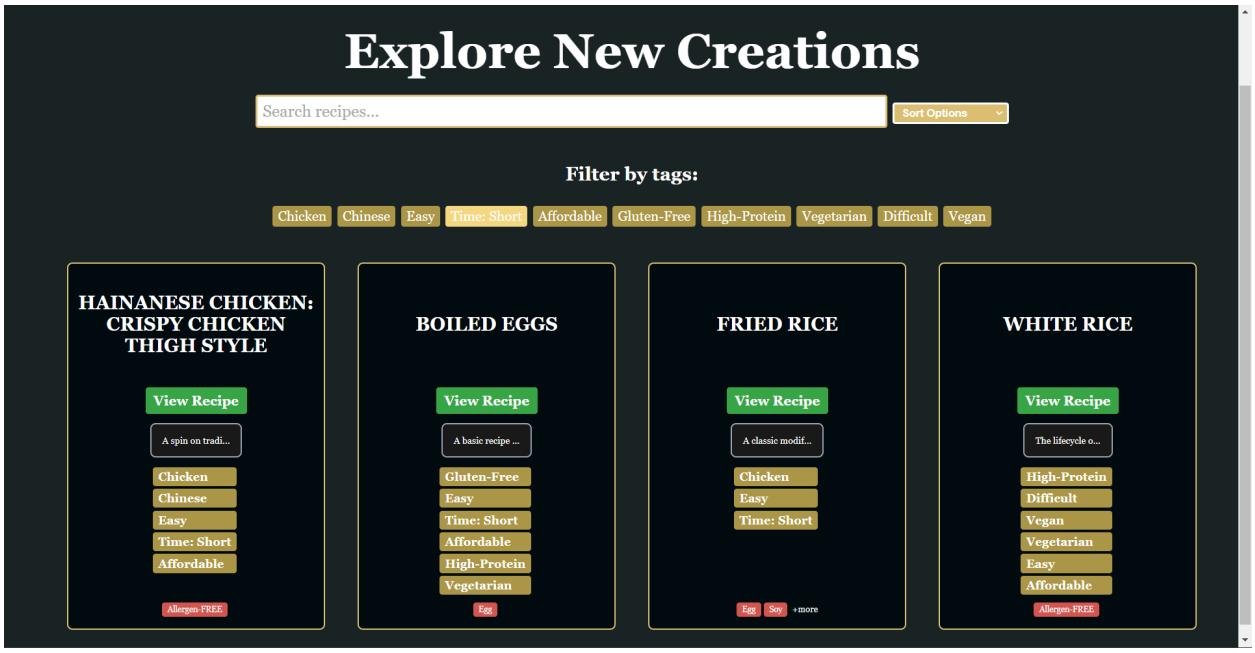
4.8.8 Create New Recipe



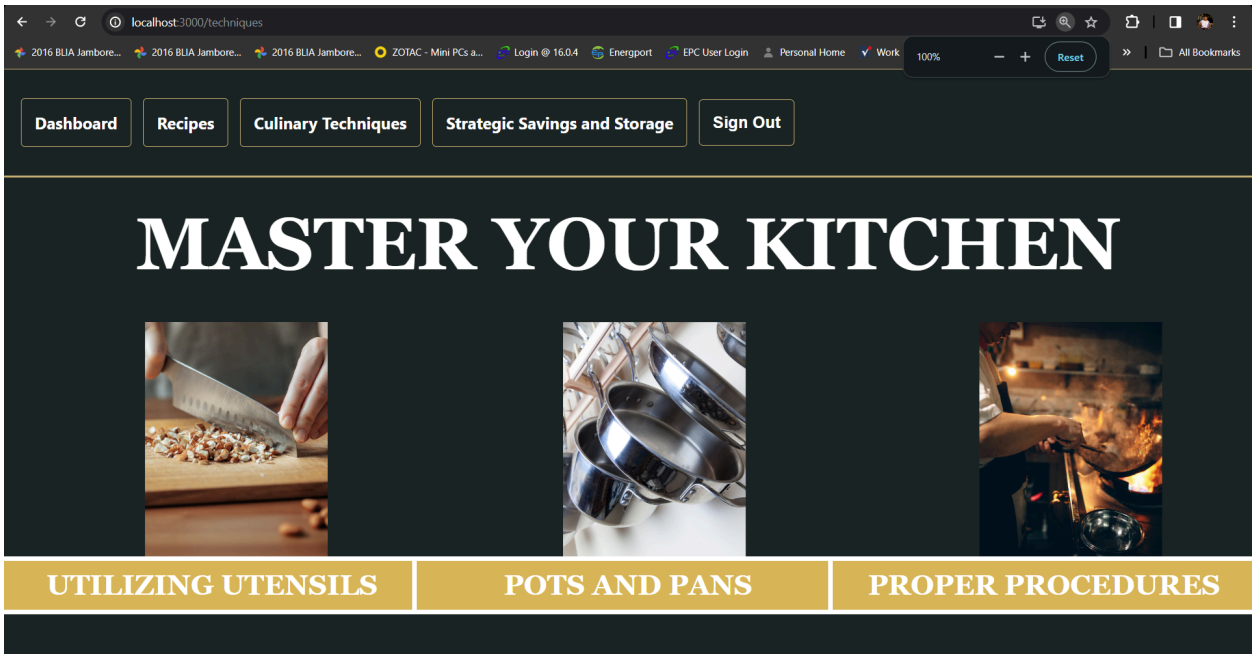
4.8.9 Food List

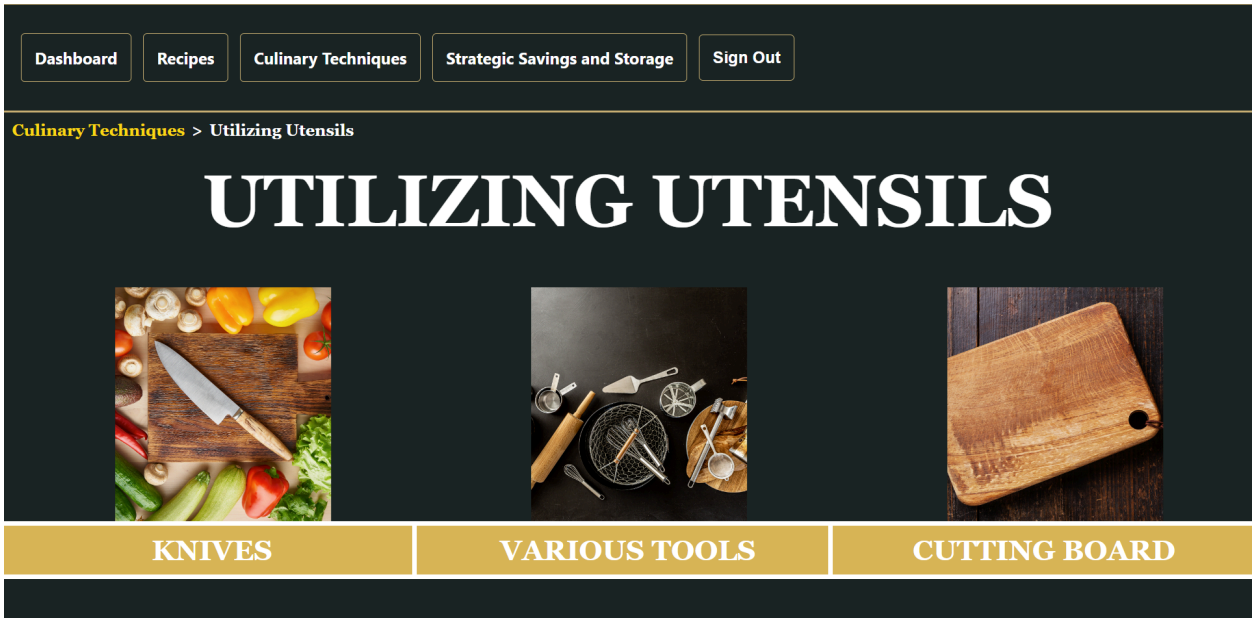


4.8.10 Explore New Creation

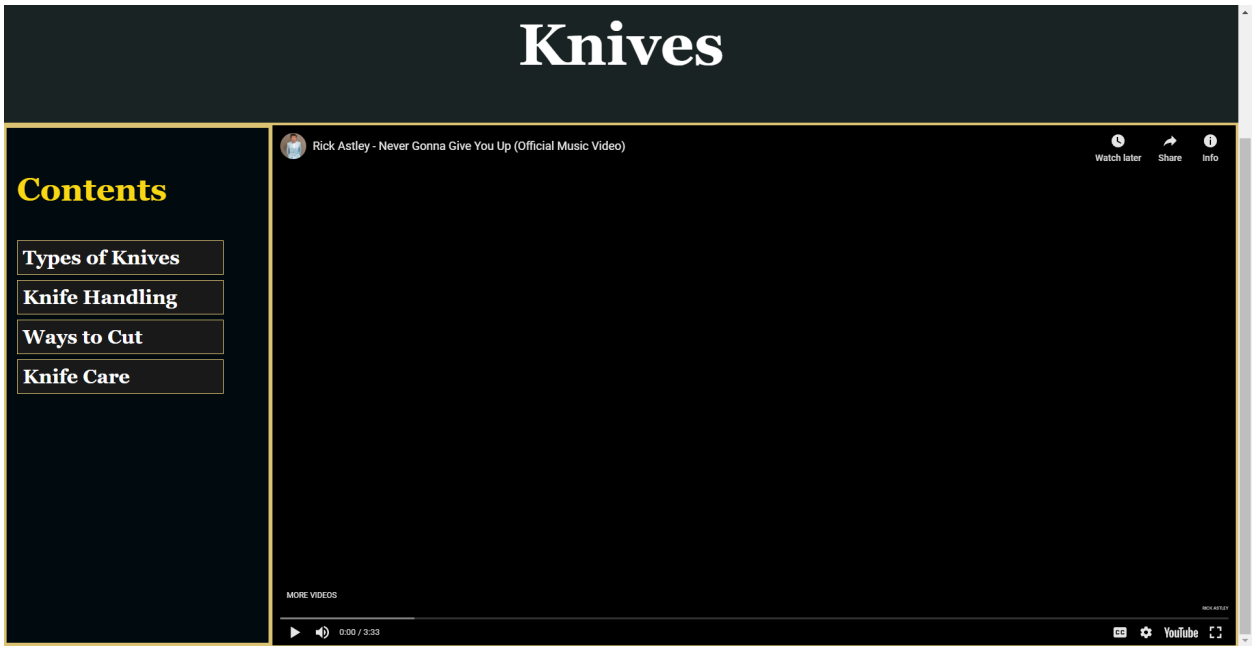


4.8.11 Culinary Techniques





Below is the format for a video.



5. Product Design Specification Approval

The undersigned acknowledge they have reviewed the Waste to Taste **Product Design Specification** document and agree with the approach it presents. Any changes to this Requirements Definition will be coordinated with and approved by the undersigned or their designated representatives.

Signature:	LP	Date:	4/10/2024
Print Name:	Lucas Prifti		
Title:	Developer		
Role:	Database/Backend Lead, Presentation Lead, Note-Taker, QA Lead		

Signature:	SC	Date:	4/10/2024
Print Name:	Saadman Choudhury		
Title:	Developer		
Role:	Documentation Lead UI/Frontend Lead Assistant Database/Backend Lead		

Signature:	DL	Date:	4/10/2024
Print Name:	Delbert Li		
Title:	Developer		

Role:	Team Lead Presentation Lead Assistant Documentation Lead UI/Frontend Lead

6. Appendix A: References

[Insert the name, version number, description, and physical location of any documents referenced in this document. Add rows to the table as necessary.]

The following table summarizes the documents referenced in this document.

Document Name and Version	Description	Location
Development Plan V.1	This is the first iteration of the Waste To Taste Development Plan document.	https://github.com/Aluckyasian/WasteToTaste
Software Requirements Specification V.1	This is the first iteration of the Waste To Taste Software Requirements Specification document.	https://github.com/Aluckyasian/WasteToTaste
Web Content Accessibility Guidelines (WCAG) 2.1	Guidelines for making web content more accessible to people with disabilities.	https://www.w3.org/TR/WCAG21/

7. Appendix B: Key Terms

The following table provides definitions for terms relevant to this document.

Term	Definition
MongoDB Atlas	MongoDB's fully automated cloud service. Used for database services as mentioned in Dependencies.
YouTube	A video sharing service where users can watch, like, share, comment, and upload their own videos. Referenced in section 3.2.9 Recipe Saving for video content.

React.js	The JavaScript library I'm utilizing for building our application's user interfaces, chosen for its efficiency in creating dynamic, single-page applications.
Node.js	The JavaScript runtime environment enables me to execute JavaScript code server-side, forming the backbone of our application's server logic.
Express.js	The web application framework for Node.js that I'm using to build our application's server and API, due to its simplicity and flexibility.
JWT	The method I've implemented for managing user sessions and access permissions, using compact, URL-safe tokens to represent claims between two parties.
API	The set of protocols and tools that allows different parts of our software to communicate, essential for the operation of our web application.
HTTPS	The protocol I'm enforcing for secure communication over the network, ensuring all data transfer is encrypted and protected.