University of Regina

Final Project Report

Title:

Grecipe

Group Name:

Grecipe

Student names
Manjotveer Singh
Jagroop Singh
Kabeer Sheikh
Mohammad Zafar
Mykhaylo Vasylyev

Department of Computer Science

CS 476-001

Winter 2024
Ali Bayeh

March 27, 2024

Table of contents

Problem Definition

Many people of all the different demographics struggle with the preparation of food and creating healthy meals on their own. This issue could get worse if the people are limited by their budget or what kind of food they can buy because of religious beliefs or allergies. Most of the online videos that have recipes do not provide an actual solution to the problem because these videos sometimes use the ingredients that are out of the user's budget or sometimes the ingredients are not available in their local area. Our application aims to solve this issue by allowing the family members, roommates, or a group of friends by aggregating the groceries that the group buys. This application aims at making it simpler to have the recipes, reduce the food waste and the users can budget their groceries easily.

On the user side the user when they purchase groceries, they can upload the receipt to the application and the application will scan and process the receipt and update the inventory for the user. Then the user can use the application to recommend the recipes that are possible form the availability in the user's inventory. This application can help the users to budget their groceries more effectively and cook food with the ingredients that are available to them.

Our motivation for the project was that being the students we all have struggled with budgeting for our groceries and cooking food from the ingredients that are available to us. A lot of food wastage from the things that reach their best before date was also one of the reasons to do so. So this application is to help our selves and other students who faced the same issues. We not only wanted to create a website just for the class but also some thing that could be potentially be used in practical life. We opted for this specific application direction and domain for our project due to the inherent challenge it presented, offering us valuable hands-on experience.

Application Benefits

When we compare our application system it is apparent that there are multiple advantages that are offered by our application system. The systems that we used for the comparison of our systems are a) a traditional paper cookbook and b) YouTube cooking videos.

Existing system 1: A traditional paper cook book

The first system that our application would exceed are the traditional paper cookbooks. The cook books that people have are the written books with various kinds of recipes described in them. The table below compares the advantages and the disadvantages of the recipe books to our system.

	Advantages	**Disadvantages**
Recipe books	• Pre organized by the book creators according to the different categories of the food. • Cheap to produce	• Does not take into consideration of the ingredient availability. • Outdated information is prevalent as no one is going to update a recipe on an already printed cook book. • Does take the budget into consideration.
Grecipe	• It only makes the recipes according to the ingredients presented to it. • Regularly updated recipes. • Recommends the recipes according to user's budget.	• Some people are not technologically proficient. • Requires internet connection.

Reference for the cook book is: https://www.gregdoucette.com/products/cookbook-2 .

Existing system 2: Video based cooking recipes

The second and one of the most popular kinds of cooking instructions is through YouTube videos. We can find numerous videos online of people teaching others how to cook meals. The

advantages and the disadvantages of these are discussed in a table below.

	Advantages	**Disadvantages**
Cooking videos	Anyone can make a video and people can decide based on likes and comments which video is good. The large number of videos are being uploaded everyday so user can find fresh videos.	Does not take the consideration of availability of ingredients. Does take the budget into consideration.
Grecipe	Gives the recipes according to the ingredients scanned from the receipt. Budget friendly.	User should know how to use basic technology. Requires internet connectivity.

There are various references for the videos for cooking recipes the one's that we used for this one is : https://www.youtube.com/watch?v=dMXEM9KXB9s .

Requirements Elicitation and Specification

Functional Requirements:

For Family Head:

- **Grocery Receipt Upload**: The family head can upload a photo or a scanned image of a grocery receipt to the app. The app will analyze the receipt and extract the item names, quantities, and prices from the receipt. The app will then update the inventory database with the new items and their details. The app will also display a confirmation message to the family head, showing the items that were added to the inventory.

- **Inventory Management**: The family head can view the current inventory of groceries on the app. The app will show the item names, quantities, expiration dates, and categories (such as fruits, vegetables, dairy, etc.) of the groceries. The family head can also add, delete, or modify any item in the inventory manually. The app will update the inventory database accordingly and display a confirmation message to the family head.
- **Family Member Management**: The family head can create and manage accounts for other family members on the app. The app will allow the family head to assign different permissions to each family member, such as viewing, editing, or adding items to the inventory, accessing recipes, or submitting meal preferences. The app will also allow the family head to delete or deactivate any family member account if needed.
- **Budget Tracking**: The family head can track and manage the grocery spending on the app. The app will show the total amount spent on groceries for the current month, as well as the previous months. The app will also show the breakdown of spending by category, item, or store. The family head can also set a monthly budget limit for groceries and receive alerts from the app when the limit is reached or exceeded.

For Family Member:

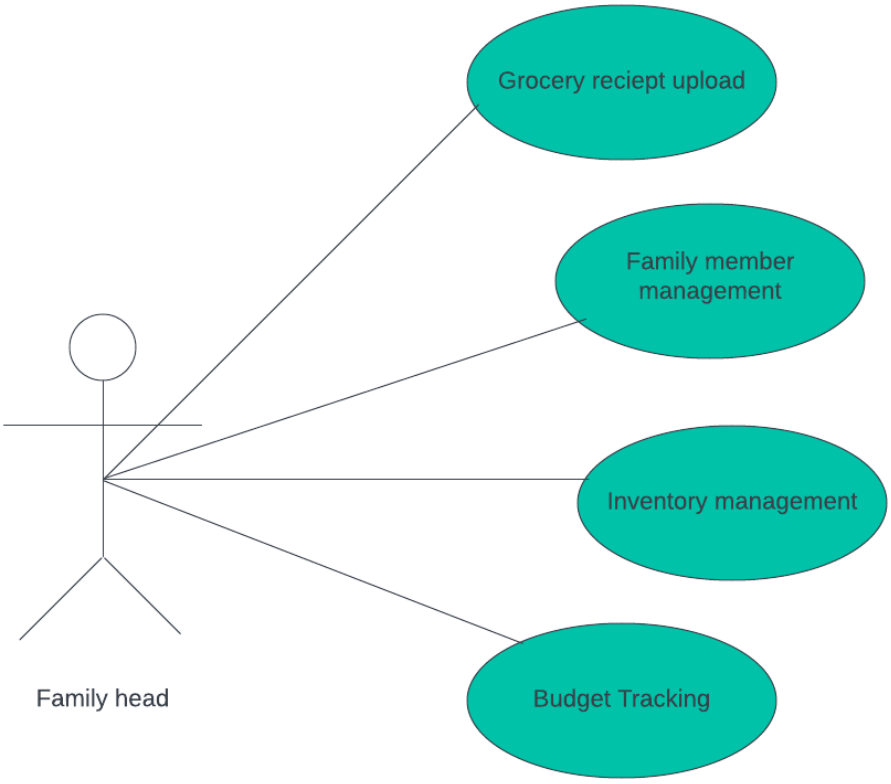
- **Manual Item Entry**: The family member can add items to the grocery list manually on the app. The app will ask the family member to enter the item name, quantity, and expiration date. The app will then update the grocery list database with the new item and its details. The app will also display a confirmation message to the family member, showing the item that was added to the list.
- **Recipe Access**: The family member can view recipes suggested by the app on the app. The app will use an algorithm to generate recipes based on the available groceries in the

inventory, as well as the meal preferences and dietary restrictions of the family members. The app will show the recipe name, ingredients, instructions, and nutritional information for each recipe. The family member can also rate, save, or share the recipes on the app.

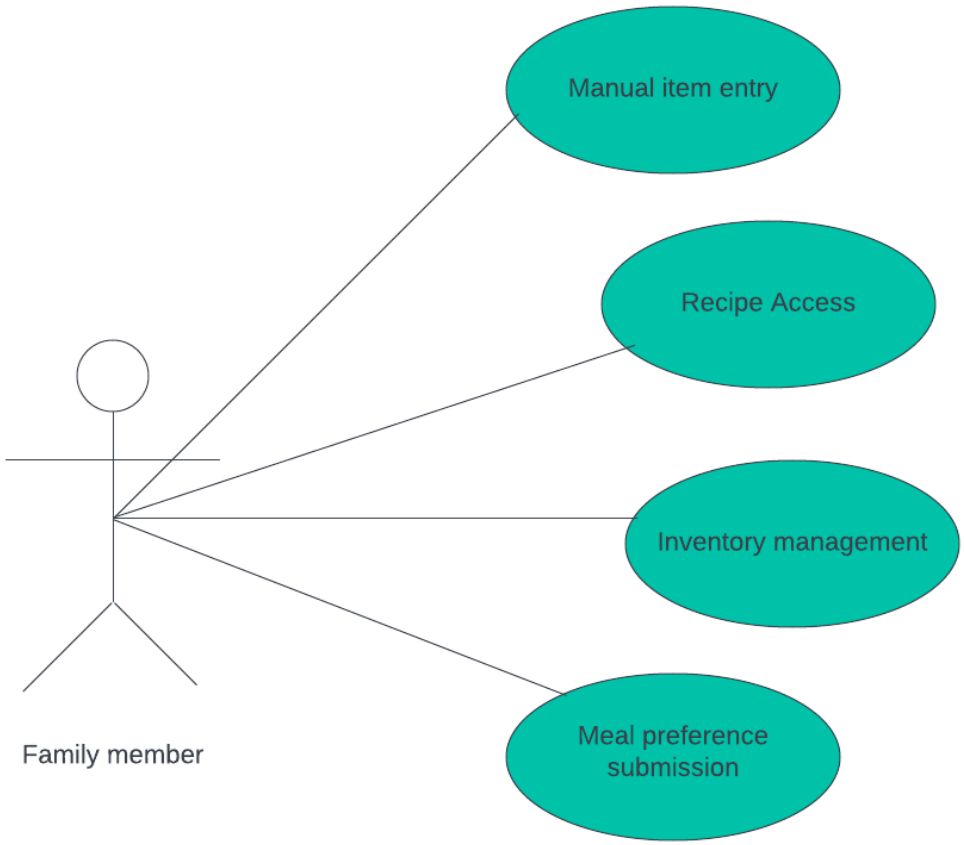
- **Inventory Interaction**: The family member can view the inventory of groceries on the app. The app will show the same information as the family head, except for the prices of the groceries. The family member can also suggest edits to the inventory, such as correcting the item name, quantity, or expiration date. The app will send the suggested edits to the family head for approval or rejection. The app will also display a confirmation message to the family member, showing the status of their suggested edits.
- **Meal Preferences Submission**: The family member can submit their meal preferences or dietary restrictions on the app. The app will ask the family member to select their preferred cuisines, dishes, or ingredients, as well as any allergies, intolerances, or special diets they have. The app will then store the meal preferences or dietary restrictions of the family member in the database. The app will also use this information to generate recipes that suit the family member's tastes and needs.

Use Case Diagrams:

Family Head Diagram

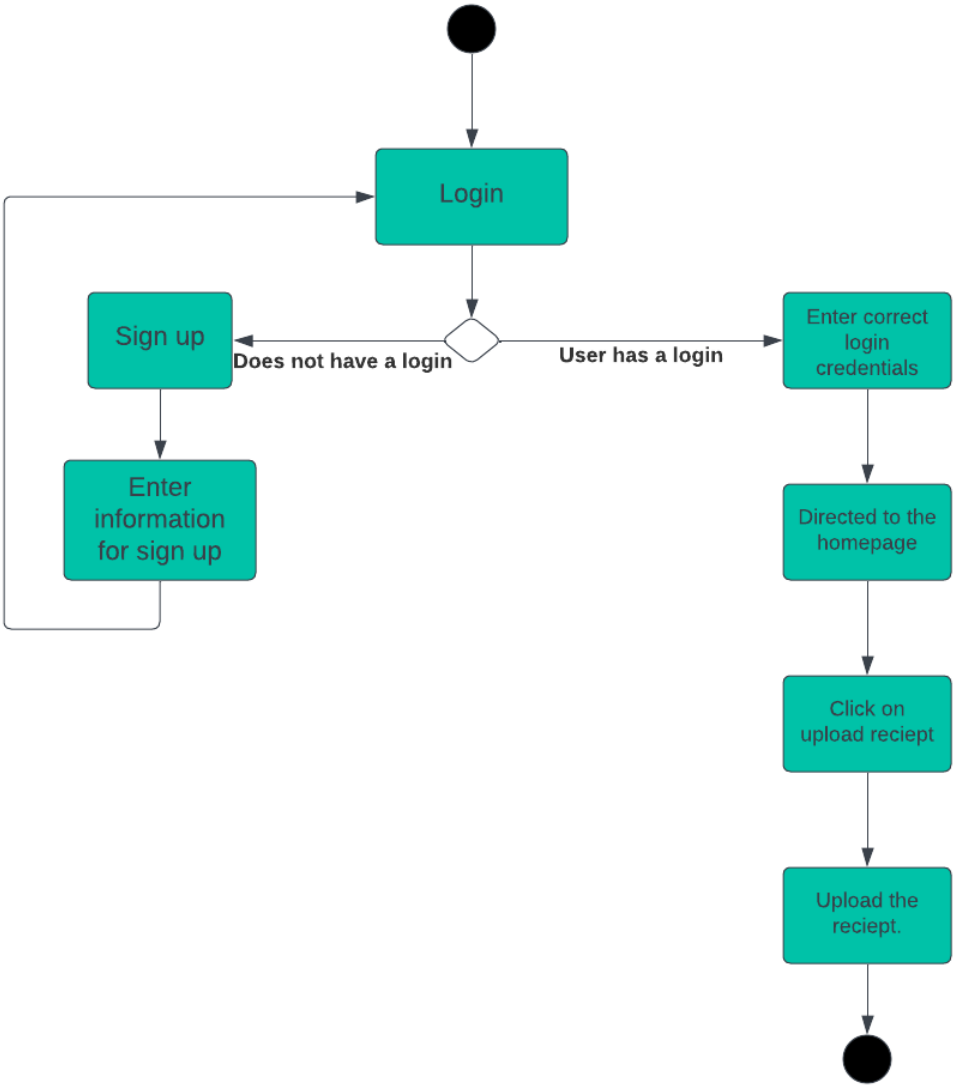


Family Member Diagram



Activity Diagrams:

Grocery Receipt Upload



Activity 1: Grocery receipt upload

Actors: Family head

Preconditions:

- User must have a login to the website.
- User must have receipts to upload.

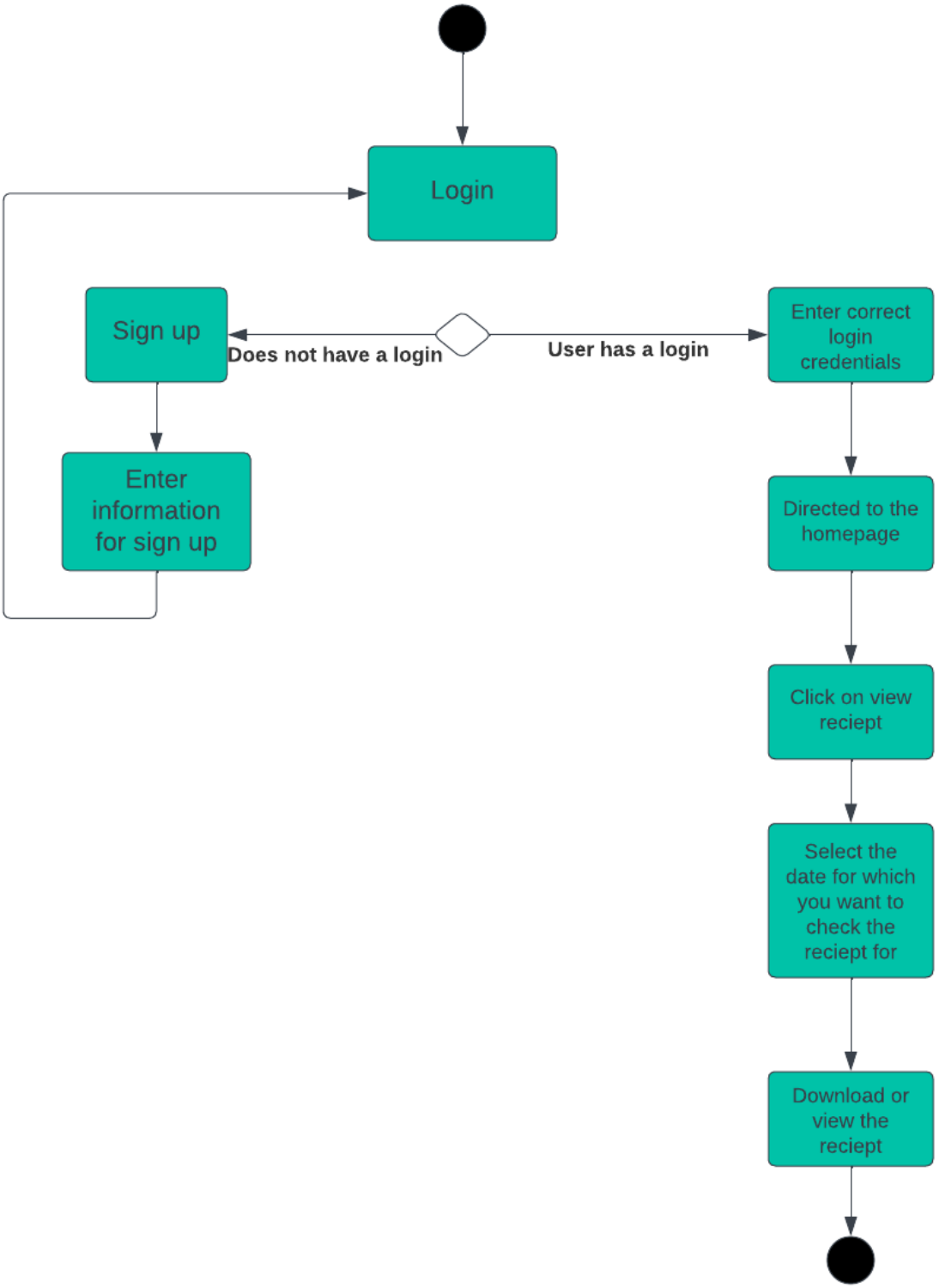
Main flow:

- The user logs in to the website.
- The system checks for the correct login credentials.
- After verification of the credentials the user is directed to the homepage of the website.
- The user selects the option to upload the receipt.
- The user uploads the receipt.
- The user saves the uploaded receipt.

Post conditions

- The receipt could be found in the receipt access section.
- Every member of the group can access receipts.

Recipe Access



Activity 2: Receipt Access

Preconditions

- The user should have a login to the website.
- There should be at least one receipt uploaded.

Main flow:

- The user logins to the website.
- The system verifies the login credentials.
- Once verified the user is directed to the homepage.
- The user goes to the receipt access page and selects the date for which the user wants the receipt.
- The system shows the receipt to the user.

Post conditions:

- The receipt is available to user for viewing.
- The user cannot make any edits to the receipts.

Software Qualities:

For Family Head:

- **Correctness**: The app ensures that only the family head can upload receipts, manage the inventory, manage the family members, and track the budget. The app also verifies the data extracted from the receipts and the inventory updates with the family head before saving them to the database. The app also validates the inputs from the family head and prevents any invalid or inconsistent data from entering the system.
- **Robustness**: The app handles any errors or exceptions that may occur during the receipt upload, inventory management, family member management, or budget tracking

processes. The app also provides error messages and suggestions for correcting the errors to the family head. The app also protects the data from unauthorized access, modification, or deletion by implementing security measures such as encryption, authentication, and authorization.

- **Performance**: The app processes the receipt uploads and the inventory updates in a timely manner. The app also retrieves the inventory data and the budget data quickly, using indexing and query optimization techniques.

For Family Member:

- **Correctness**: The app ensures that only the family members can access the recipes, interact with the inventory, and submit their meal preferences. The app also generates the recipes that match the available groceries, as well as the meal preferences and dietary restrictions of the family members. The app also validates the inputs from the family members and prevents any invalid or inconsistent data from entering the system.
- **Robustness**: The app handles any errors or exceptions that may occur during the manual item entry, recipe access, inventory interaction, or meal preferences submission processes. The app also provides error messages and suggestions for correcting the errors to the family members. The app also protects the data from unauthorized access, modification, or deletion by implementing security measures such as encryption, authentication, and authorization.
- **Performance**: The app processes the manual item entries and the inventory interactions in a timely manner. The app also retrieves the recipes and the inventory data quickly, using indexing and query optimization techniques.

Top-Level and Low-Level Software Design

MVC Architecture

The MVC architecture stands for Model View Controller architecture. This kind of architecture encourages differentiation between the behavior, display and data. The separation because of the MVC architecture provides a better division of work and better maintenance of the system.

Model:

The model is the data handling unit of the architecture. It is responsible for managing the data and any requests to change or modify the data.

View:

The view is the component that helps in display the data to the user. View displays the information passed by the model and forward any input from the user to the controller.

Controller:

The controller is intermediate component between the model and the view. The view sends the requests to the controller. The controller processes the request with communicating to the model and then update the view accordingly.

In our system, the view:

- Manages the presentation layer which includes getting requests from the user to change the data and display the result of the changed data.
- The view is implemented by using HTML and CSS programming languages.

In our system, the model:

- Creates a blueprint for all of the important functions of the system using the data and the business logic.

In our system, the controller:

- Serves as an intermediate between the View and the Model.
- Changes the Model from the user request like adding the receipt.
- Controller respond to the events and execute API requests.

Model files

- Family.js: This model file defines the structure and behavior of the Family entity in the application, allowing developers to perform operations on family data and establish relationships between families and users.
- Item.js: This file creates a Mongoose model named 'Item' based on the itemsSchema. This model will be used to interact with the MongoDB collection corresponding to the Item entity.
- User.js: This file defines the structure and behavior of the User entity in the application, allowing developers to perform operations on user data and establish relationships with families using Mongoose.

View files

- Index.js: This file serves as a central point for organizing and integrating routes for different entities in the web application, allowing for modular and maintainable code structure.
- FamilyRoutes.js: This view file defines routes for handling operations related to families in the web application, ensuring proper request validation and authentication using middleware functions.
- ItemRoutes.js: Index.js: This file serves as a central point for organizing and integrating routes for different entities in the web application, allowing for modular and maintainable code structure.

- RecipeRoutes.js: This file defines routes for fetching recipes and generating meal plans based on user input, allowing users to access recipe-related functionalities in the web application.
- UserRoutes.js: This file defines routes for performing operations on user data, with support for request validation and authentication, providing endpoints for managing users in the web application.

Controller files

- FamilyController.js: Have various functions like createFamily, getFamily, updateFamilyMembers etc. to manipulate family related data in our web application. These functions interact with services that encapsulate the business logic for managing families.
- ItemsController.js: Functions like getItemsbyFamily, createItems, deleteItems etc. interact with services that encapsulate the business logic for managing grocery items and user data.
- RecieptController.js: This controller file contains a single function, parseReciept, responsible for parsing a receipt image uploaded by a user in our web development project.
- RecipeController.js: This controller file contains two functions responsible for handling requests related to recipes and meal plans in a web development project.
- UserController.js: This file contains functions like createUser, deleteUser, updateUser etc. responsible for handling requests related to user management in the web-based system.

Benefits of Using MVC

Overall MVC architecture helped us in improving the code organization, maintainability, scalability, flexibility and testing. Here are the few benefits that we experienced while using the MVC architecture.

1. Separation of content:
    - Using MVC we were able to separate the contents of our code into distinct components, each component focusing on the different aspect of the application.
    - Using the separation, we were able to make changes to one part of the components without affecting the others.
    - For example, if we were to change the how the user is able to upload the receipt, we can only change that and nothing else would be affected.
2. Enhanced testability:
    - With MVC we were able to test our website very effectively by dividing the testing in to components based on Model, View and the Controller.
    - Not only we can test the different components distinctively but also using MVC we were able to test the interactions of these components.
3. Improved user experience:
    - Using MVC has allowed us to create more interactive and responsive user interface.
    - This thing has let us to create an interface which is beneficial for both of the users, the family head and the family member.

Design Patterns

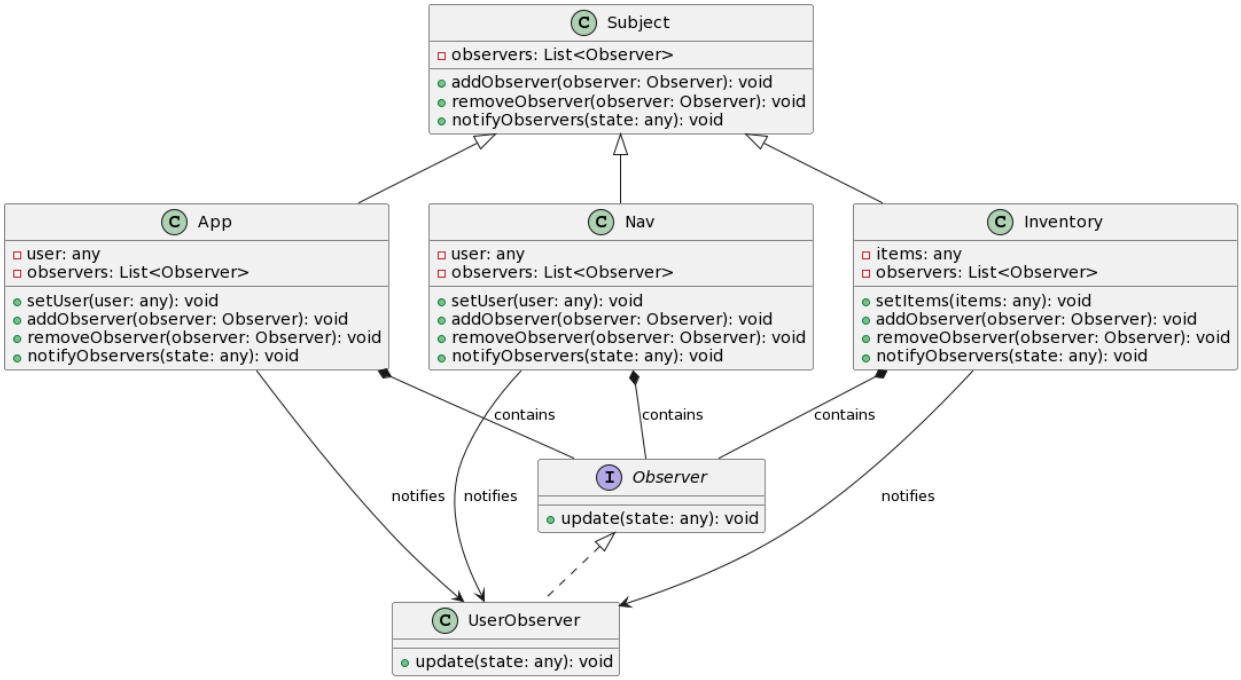
Observer design pattern

The observer pattern is the design pattern in which one component maintains a list of dependents called observers and notifies them of any state changes, usually calling of their functions. This system is beneficial for developing distribute event handling systems.

Usability for our system:

- **User Interface Updates:** When any element of our website system is updated the observer design pattern is used. For example, whenever a family head uploads a receipt a confirmation message is sent to user's browser window saying that the receipt has been uploaded. In the family member account also once the change is made, they also get notified about the change.
- **User Authentication and Navigation:**

  When a user navigates to the inventory page, the App will make a request to the Nav to route the user to the correct page, which in this case would be the Inventory. In this case, the "client" is the UserContext that routes to user to the appropriate page, while the App, Nav, and Inventory are "observers" that change their function based on a request from the client Additionally, the user is checked to see if they're logged in, and are restricted from certain features based on that statusObserver design pattern class diagram



Methods and prototypes

- setUser(user: any): void
- addObserver(observer: Observer): void
- removeObserver(observer: Observer): void
- notifyObservers(state: any): void
- addObserver(observer: Observer): void
- removeObserver(observer: Observer): void

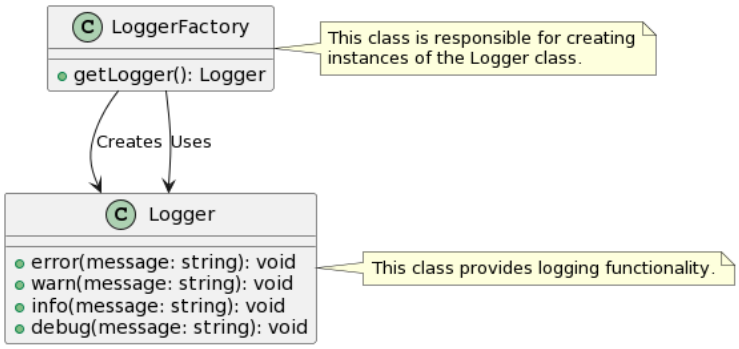
Factory Design Pattern

The factory design pattern is a widely employed pattern in the software development. Using this pattern, the developer can create objects in the project without specifying their actual classes. This design pattern offers encapsulation using which we separated the client code from the specifics of object implementation.

Usability for our system:

- **Upload File Factory:** We used the factory design that handles the files that are being uploaded to our website. In this project the files are the receipts that the family head is uploading and all the family members can access.
- **Content Regulator Factory:** In our project the user is going to through a lot of content of the generated recipes and hence to manage all that content we have used the content regulator factory design pattern because the user can effectively view the recipes that are selected for the user.

Factory design pattern class diagram

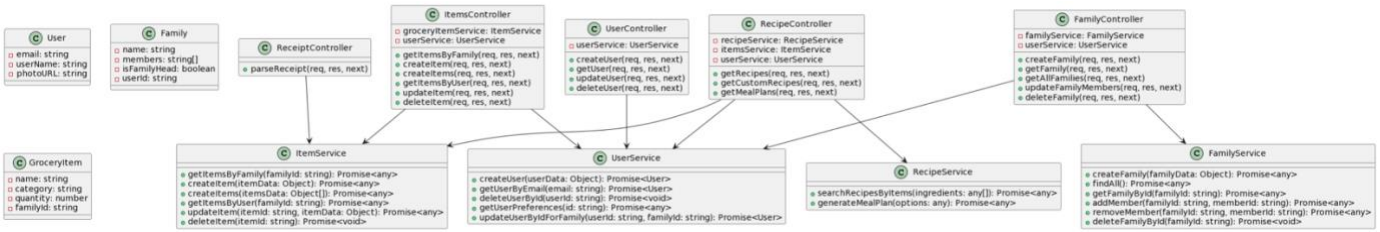


Methods and Prototypes

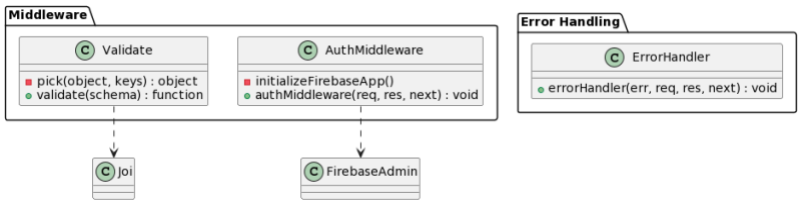
- get Logger(): Logger
- error(message: string): void
- warn(message: string): void
- info(message: string): void
- debug(message: string): void

Class diagram for whole system

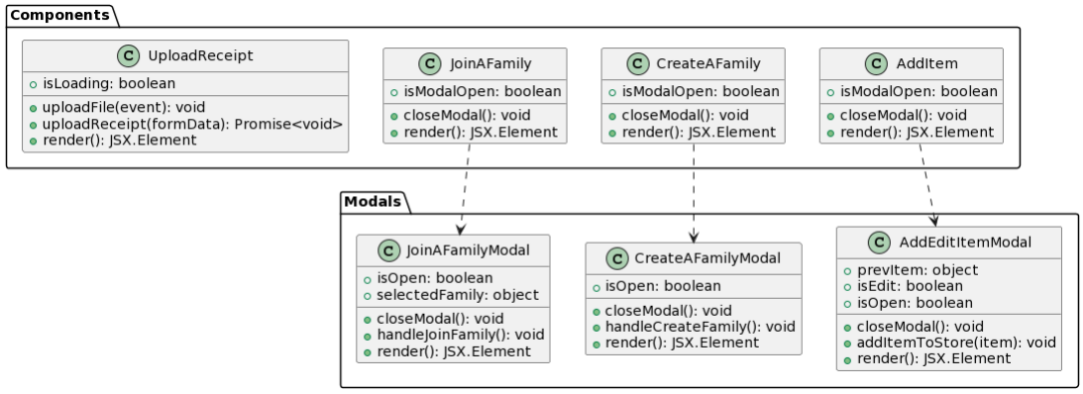
Server Class Diagram:



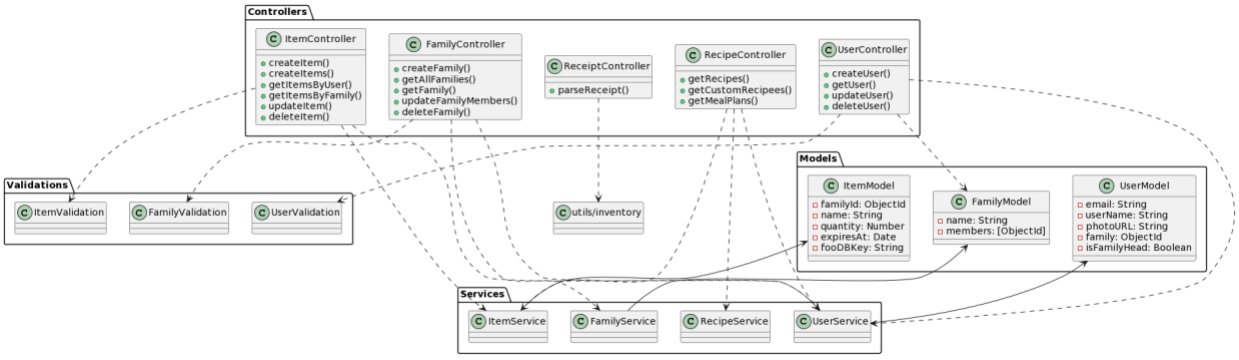
Middleware Class Diagram:



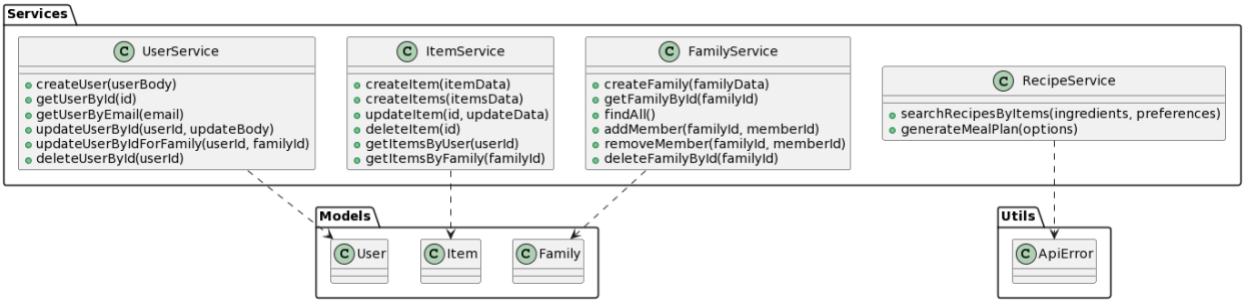
Client Modal Class Diagram



MVC Class Diagram



Service layer Class Diagram



Software Construction

Structure code within web framework

“

```
├── LICENSE
├── README.md
├── client
│   ├── README.md
│   ├── package-lock.json
│   ├── package.json
│   ├── public
│   │   ├── favicon.ico
│   │   ├── index.html
│   │   ├── logo192.png
│   │   ├── logo512.png
│   │   ├── manifest.json
│   │   └── robots.txt
│   ├── src
│   │   ├── App.css
│   │   ├── App.js
│   │   ├── App.test.js
│   │   ├── Assets
│   │   │   ├── Grecipe-Logo.svg
│   │   │   ├── Grecipe.svg
│   │   │   ├── about-background-image.png
│   │   │   ├── about-background.png
│   │   │   ├── fonts
│   │   │   │   └── coolvetica rg.otf
│   │   │   ├── home-banner-background.png
│   │   │   ├── home-banner-image.png
│   │   │   ├── john-doe-image.png
│   │   │   ├── manage-family.jpg
│   │   │   ├── manage-inventory.jpg
│   │   │   └── upload-receipt.jpg
```

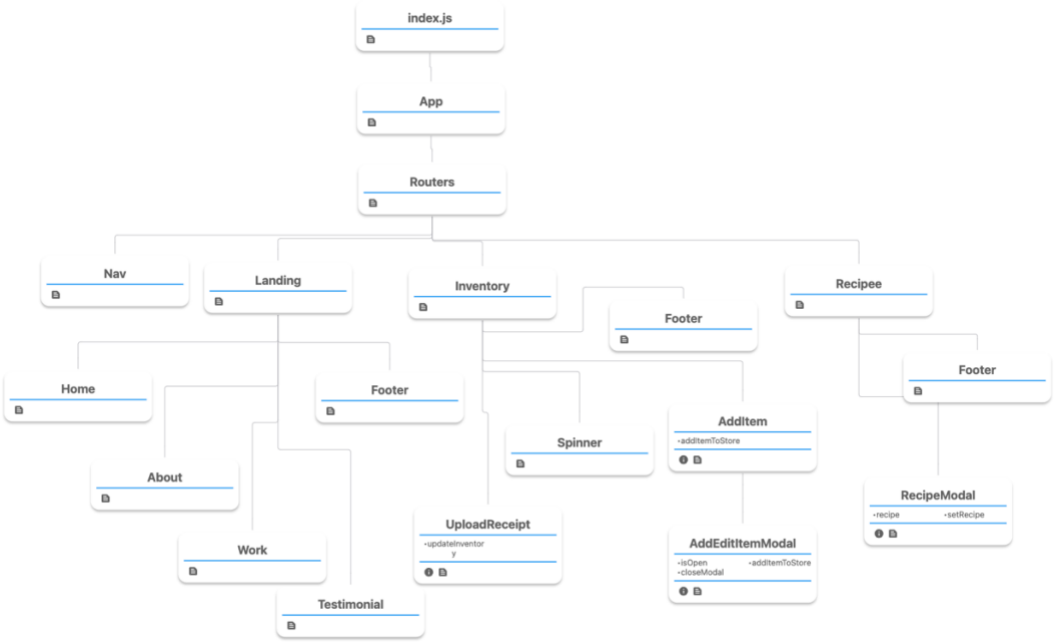
```
│   │   ├── Routers.js
│   │   ├── components
│   │   │   ├── About.js
│   │   │   ├── Footer.jsx
│   │   │   ├── Home.jsx
│   │   │   ├── Nav.js
│   │   │   ├── RecipeModal.jsx
│   │   │   ├── Spinner.js
│   │   │   ├── Testimonial.js
│   │   │   ├── Work.js
│   │   │   └── inventory
│   │   │       ├── AddEditItemModal.js
│   │   │       ├── AddItem.js
│   │   │       └── UploadReceipt.js
│   │   ├── css
│   │   │   ├── Inventory.css
│   │   │   ├── RecipeModal.css
│   │   │   └── additional-styles
│   │   │       ├── range-slider.css
│   │   │       ├── toggle-switch.css
│   │   │       └── utility-patterns.css
│   │   ├── data
│   │   │   └── foodByName.json
│   │   ├── firebase
│   │   │   └── firebase.js
│   │   ├── index.css
│   │   ├── index.js
│   │   ├── pages
│   │   │   ├── Inventory.jsx
│   │   │   ├── Landing.jsx
│   │   │   └── Recipee.jsx
│   │   ├── reportWebVitals.js
│   │   ├── setupTests.js
│   │   └── utils
│   │       ├── fetchWrapper.js
│   │       └── inventory.js
│   └── tailwind.config.js
├── docs
│   ├── diagrams
│   │   ├── Screenshot 2024-03-27 at 6.36.30 PM.png
│   │   └── deploy.png
│   ├── receipts
│   │   ├── receipt-2.png
│   │   └── receipt.png
│   ├── report
│   │   └── Final_Project_Report.pdf
```

```
│   └── slides
│       └── CS_476_Presentation.pdf
├── package-lock.json
└── server
    ├── README.md
    ├── eng.traineddata
    ├── package-lock.json
    ├── package.json
    ├── src
    │   ├── app.js
    │   ├── config
    │   │   ├── connectDB.js
    │   │   └── logger.js
    │   ├── controllers
    │   │   ├── FamilyController.js
    │   │   ├── ItemsController.js
    │   │   ├── ReceiptController.js
    │   │   ├── RecipeController.js
    │   │   └── UserController.js
    │   ├── data
    │   │   └── foodByName.json
    │   ├── middlewares
    │   │   ├── auth.js
    │   │   ├── error.js
    │   │   └── validate.js
    │   ├── models
    │   │   ├── Family.js
    │   │   ├── Items.js
    │   │   └── User.js
    │   ├── routes
    │   │   ├── FamilyRoutes.js
    │   │   ├── ItemRoutes.js
    │   │   ├── RecipeRoutes.js
    │   │   ├── UserRoutes.js
    │   │   └── index.js
    │   ├── server.js
    │   ├── server.log
    │   ├── services
    │   │   ├── FamilyService.js
    │   │   ├── ItemService.js
    │   │   ├── RecipeService.js
    │   │   └── UserService.js
    │   ├── utils
    │   │   ├── ApiError.js
    │   │   └── inventory.js
    │   └── validation
```

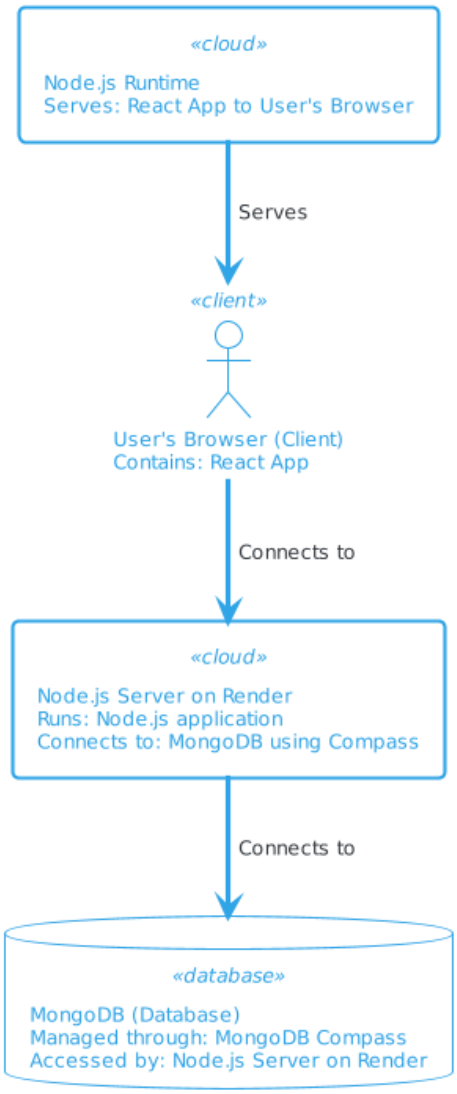
```
│       ├── ValFamily.js
│       ├── ValItems.js
│       └── ValUser.js
└── uploads

31 directories, 96 files"
```

Structure Diagram of the React App



Deployment Diagram



Supported Browsers:

- Firefox
- Google Chrome
- Safari
- Microsoft Edge

Web server

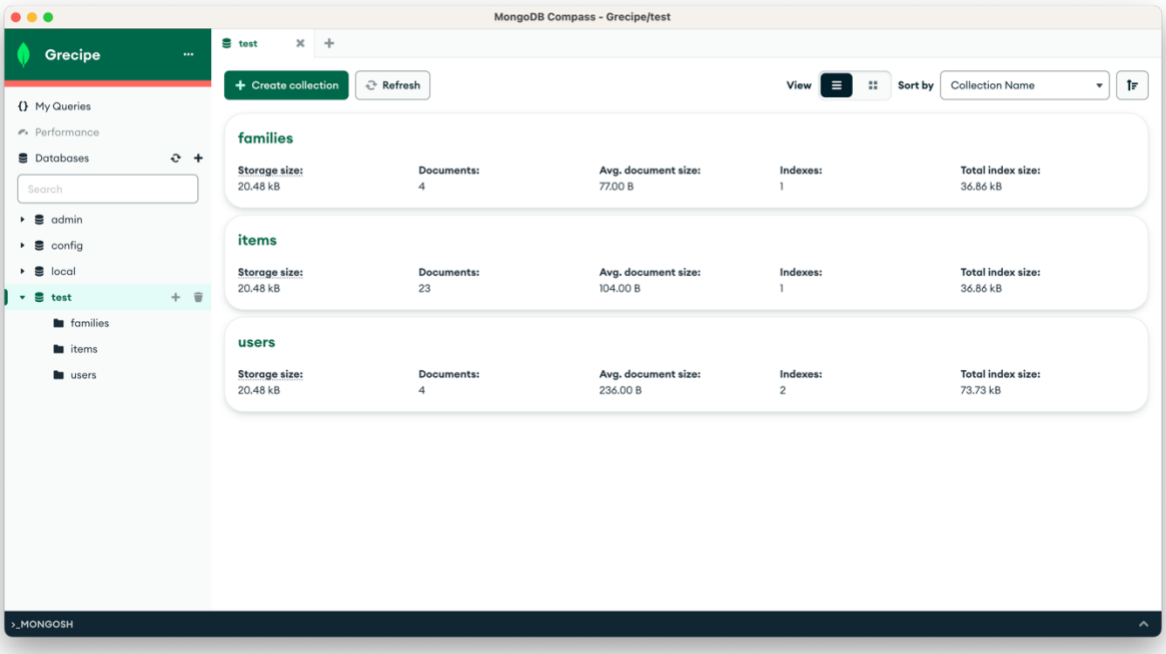
Nodejs server on render(cloud).

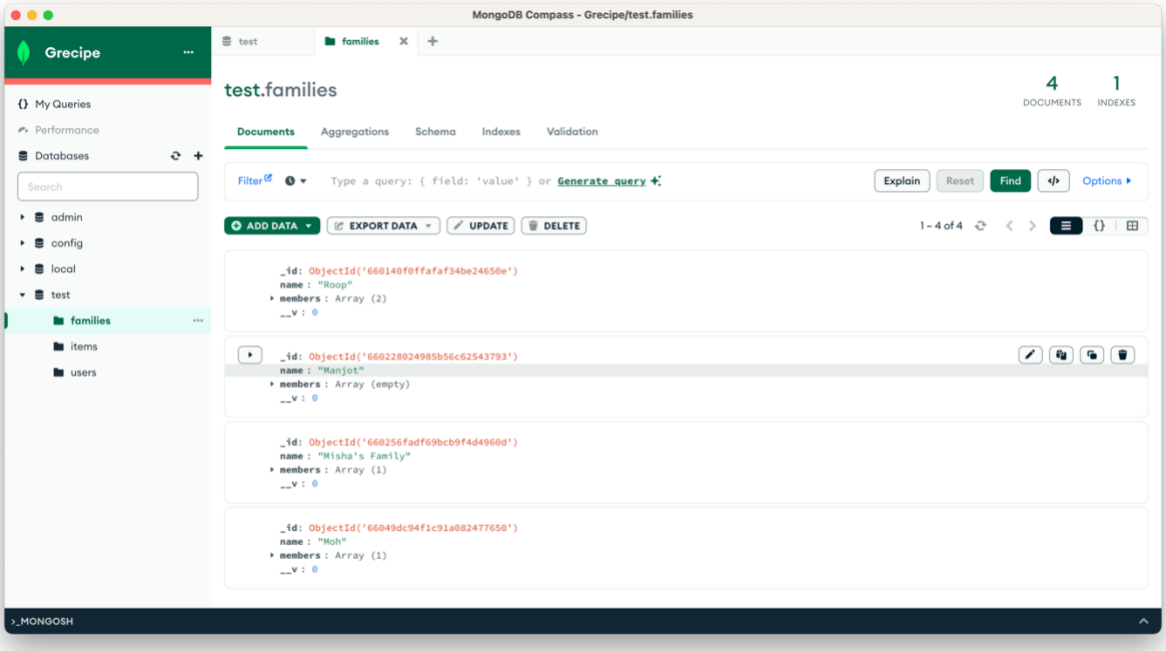
Database solution

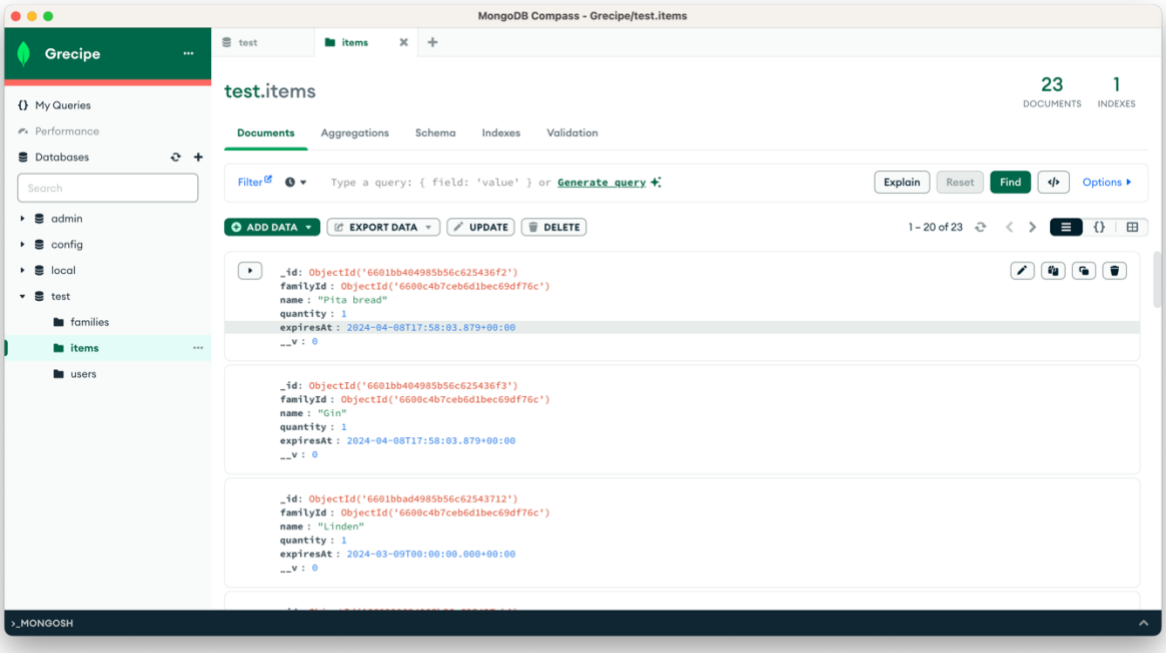
MongoDb

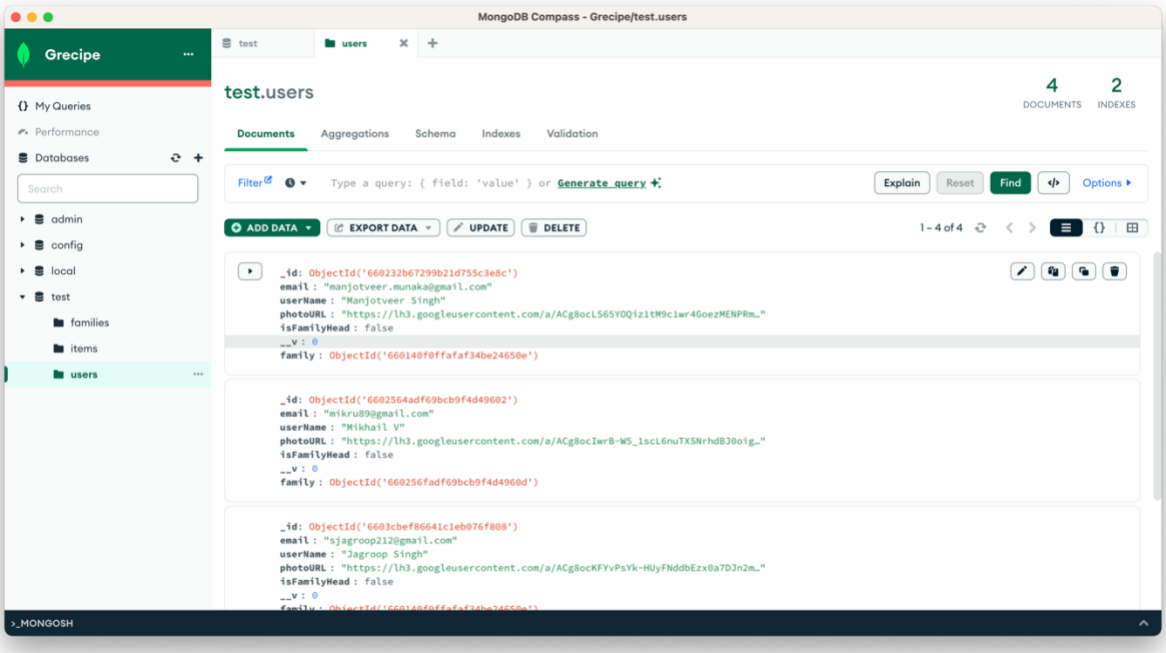
Table of contents

Each screenshot shows sample data from our MongoDb database.




MongoDB Compass - Grecipe/test.families
Grecipe
My Queries
Performance
Databases
admin
config
local
test
families
items
users
test.families
4 DOCUMENTS
1 INDEXES
Documents
Aggregations
Schema
Indexes
Validation
ADD DATA
EXPORT DATA
UPDATE
DELETE
Explain
Reset
Find
Options
_id: ObjectId('660140f0ffafaf34be24650e')
name : "Roop"
members : Array (2)
__v : 0
_id: ObjectId('660228024985b56c62543793')
name : "Manjot"
members : Array (empty)
__v : 0
_id: ObjectId('660256fadf69bcb9f4d4960d')
name : "Misha's Family"
members : Array (1)
__v : 0
_id: ObjectId('66049dc94f1c91a082477650')
name : "Moh"
members : Array (1)
__v : 0
>_MONGOSH


MongoDB Compass - Grecipe/test.items
Grecipe
My Queries
Performance
Databases
admin
config
local
test
families
items
users
test.items
23 DOCUMENTS
1 INDEXES
Documents
Aggregations
Schema
Indexes
Validation
ADD DATA
EXPORT DATA
UPDATE
DELETE
Explain
Reset
Find
Options
_id: ObjectId('6601bb404985b56c625436f2')
familyId : ObjectId('6600c4b7ceb6d1bec69df76c')
name : "Pita bread"
quantity : 1
expiresAt : 2024-04-08T17:58:03.879+00:00
__v : 0
_id: ObjectId('6601bb404985b56c625436f3')
familyId : ObjectId('6600c4b7ceb6d1bec69df76c')
name : "Gin"
quantity : 1
expiresAt : 2024-04-08T17:58:03.879+00:00
__v : 0
_id: ObjectId('6601bbad4985b56c62543712')
familyId : ObjectId('6600c4b7ceb6d1bec69df76c')
name : "Linden"
quantity : 1
expiresAt : 2024-03-09T00:00:00.000+00:00
__v : 0
>_MONGOSH



GitHub Repository Link

https://github.com/r97draco/grecipe

Link to the web-Based application

https://grecipe-1.onrender.com

Technical Documentation

Programming Languages/Libraries used

1. JavaScript
   - JavaScript is a high-level language and is primarily used to create interactive web content. It is also used for validation.
2. HTML
   - HTML is a markup language used to create structure of webpage, content and functionality.
3. JSON

- JSON is a subset of JavaScript but it is independent of any language and could be used with any language that has libraries or functions for parsing JSON data. It is used for transferring data between web server and web client.

4. CSS
    - This is a styling programming language which is used for styling HTML pages created using HTML.
5. React
    - React is widely used for creating interactive and dynamic web applications with reusable UI components.
6. Node.js
    - Node.js is an open source, cross platform JavaScript runtime environment that allows developers to run code outside a web browser

Reused Algorithms/small programs

- Axios: HTTP client library used for making HTTP requests from Node.js
- CORS: Middleware that is used to enable cross-origin HTTP requests from web browser.
- Dotenv: Zero dependency module used for loading environment variables from a .env file int Node.js applications.
- Express: Minimalist web framework for Node.js that simplifies the process of building web applications and APIs.
- Firebase-admin: Provides tools for managing firebase projects from a server-side environment.
- Mongoose: Object modelling library for for MongoDB and Node.js. It provides a

schema-based solution for modeling application data.

- Multer: Part of middleware for handling multipart/form-data which is commonly used for file uploading.
- Tesseract.js: An OCR (Optical Character Recognition) library for Node.js that allows applications to extract text from images.
- Winston: A logging library for node.js that provides a customizable and flexible logging solution.
- Nodemon: Utility that monitors changes to files in a Node.js application and automatically restarts the server when changes are detected.
- Prettier: Code formatter that automatically formats code according to predefined rules.
- Reused code for some components and assets [url]
- Algorithm to detect find the keywords matching with receipt and keywords.
- Spoonacular API for generating the receipts [url]
- Google Auth using Firebase

Software tools and environment

1. Visual Studio Code
    a. Visual studio code is an open-source code development environment developed by Microsoft.
    b. VS code helped us in editing all the code that we wrote for this web application.
    c. We also used the git integration feature provided by the software for version control, branching, merging and more.
2. Postman

   a. Postman is a user-friendly interface that simplifies the process of designing, testing and debugging APIs.
   b. Using postman, we created requests to APIs using various methods like GET and POST.
   c. We used postman to monitor the performances of the APIs.

3. MongoDb compass
   a. MongoDb compass is a visual tool developed by MongoDb Inc. to make it easier for the developers to interact with MongoDb databases.
   b. MongoDb was used to build and execute MongoDb queries for our web application.
4. Terminal
   a. Terminal is a software which is used to interact with computer system using certain commands.
   b. Terminal was used to install various programming languages and other tools required for the website building.
5. Chrome DevTools
   a. Chrome DevTools are the tools available directly on google chrome web browser that help the developer in building the required softwares.
   b. We used inspect element and console to help us building a good working website.
6. GitHub
   a. GitHub is a platform which is used to have the version control for any development project.

b. It allowed us to work independently and then merge all of our work.

7. Lucid charts

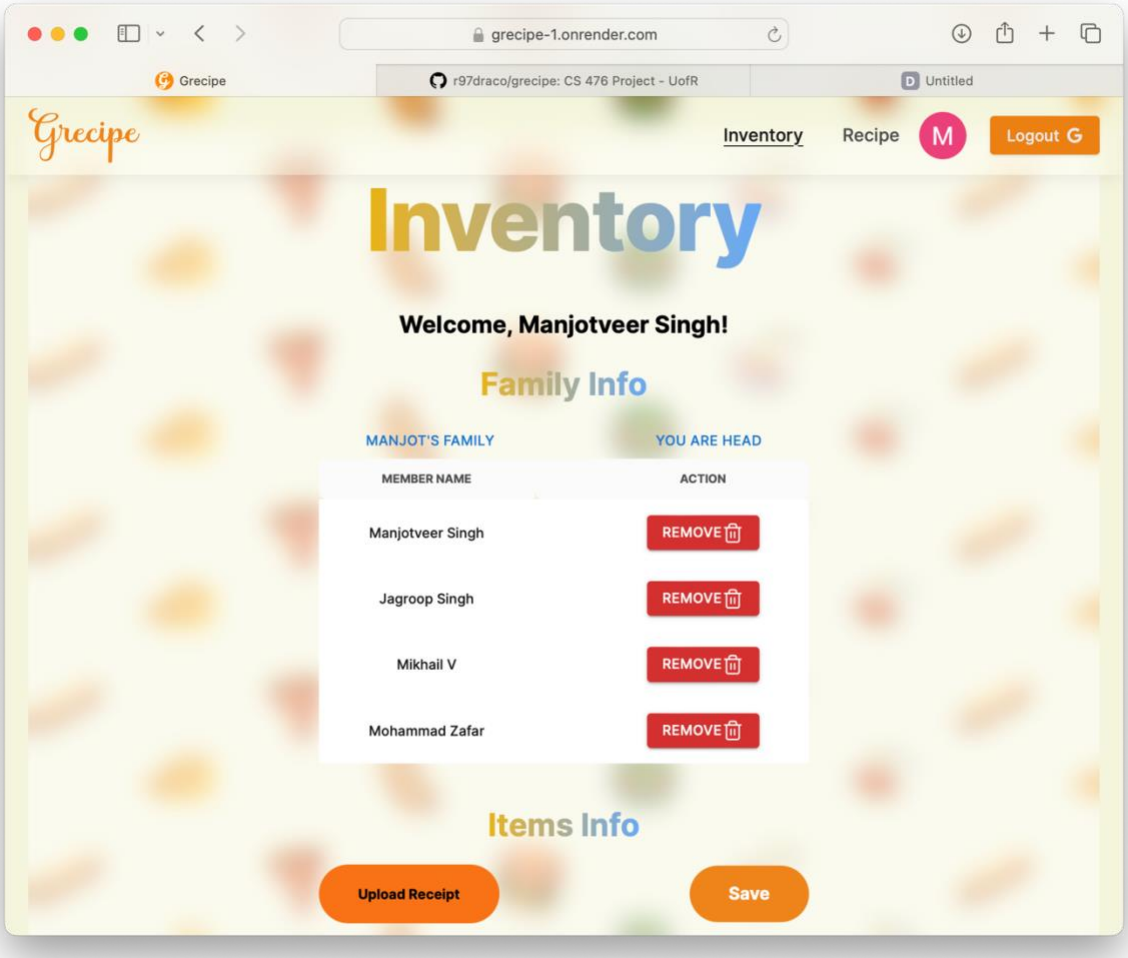
   a. A free software which allows the users to develop diagrams.

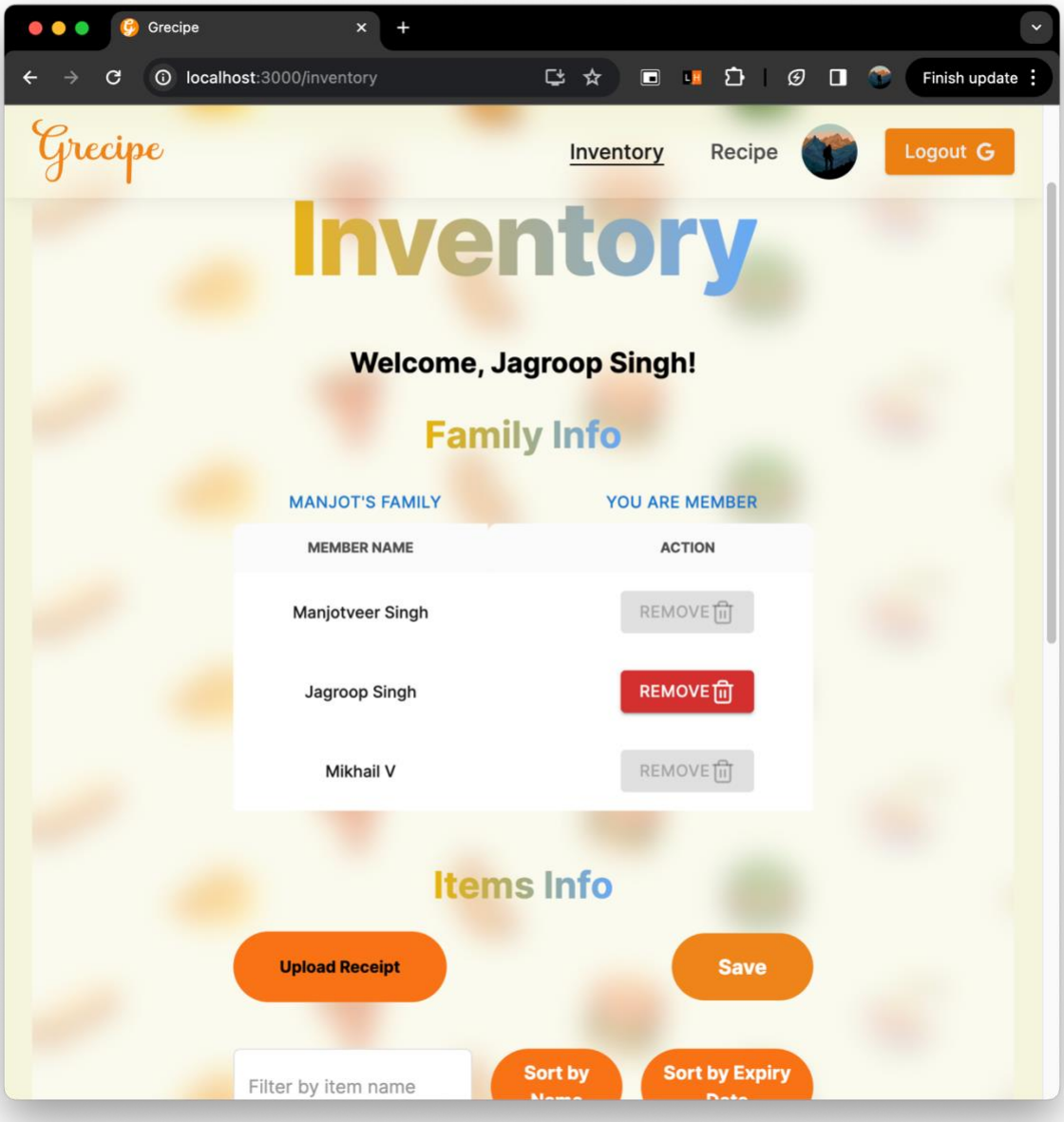
   b. We used lucid charts to develop the use case diagrams, activity diagrams and many more.

Acceptance Testing

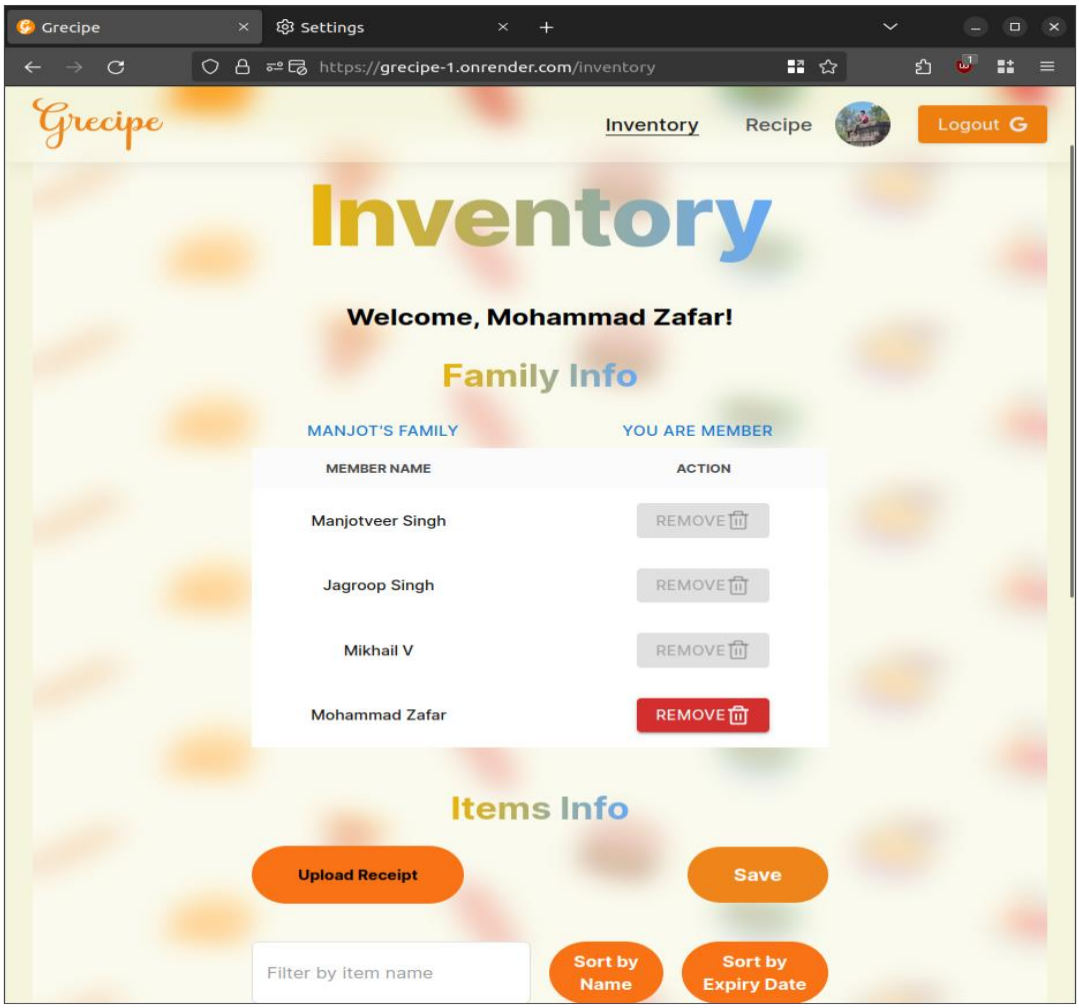
Correctness Testing

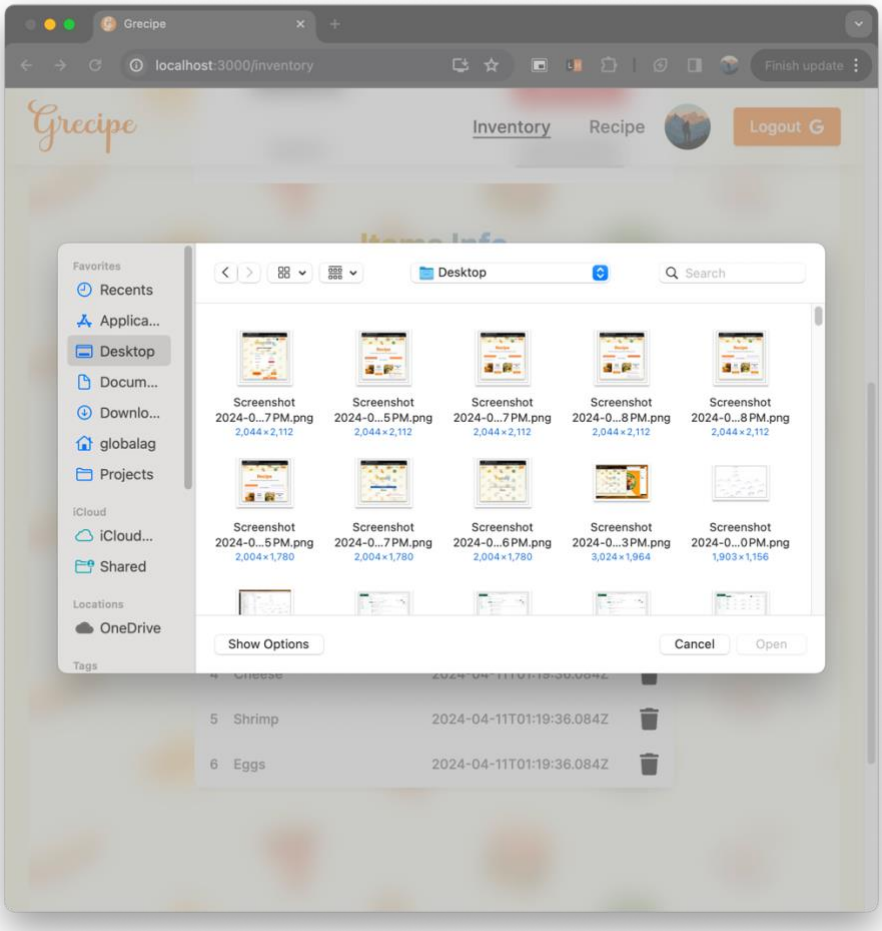
Test Case 1: Only family head can remove the family members:



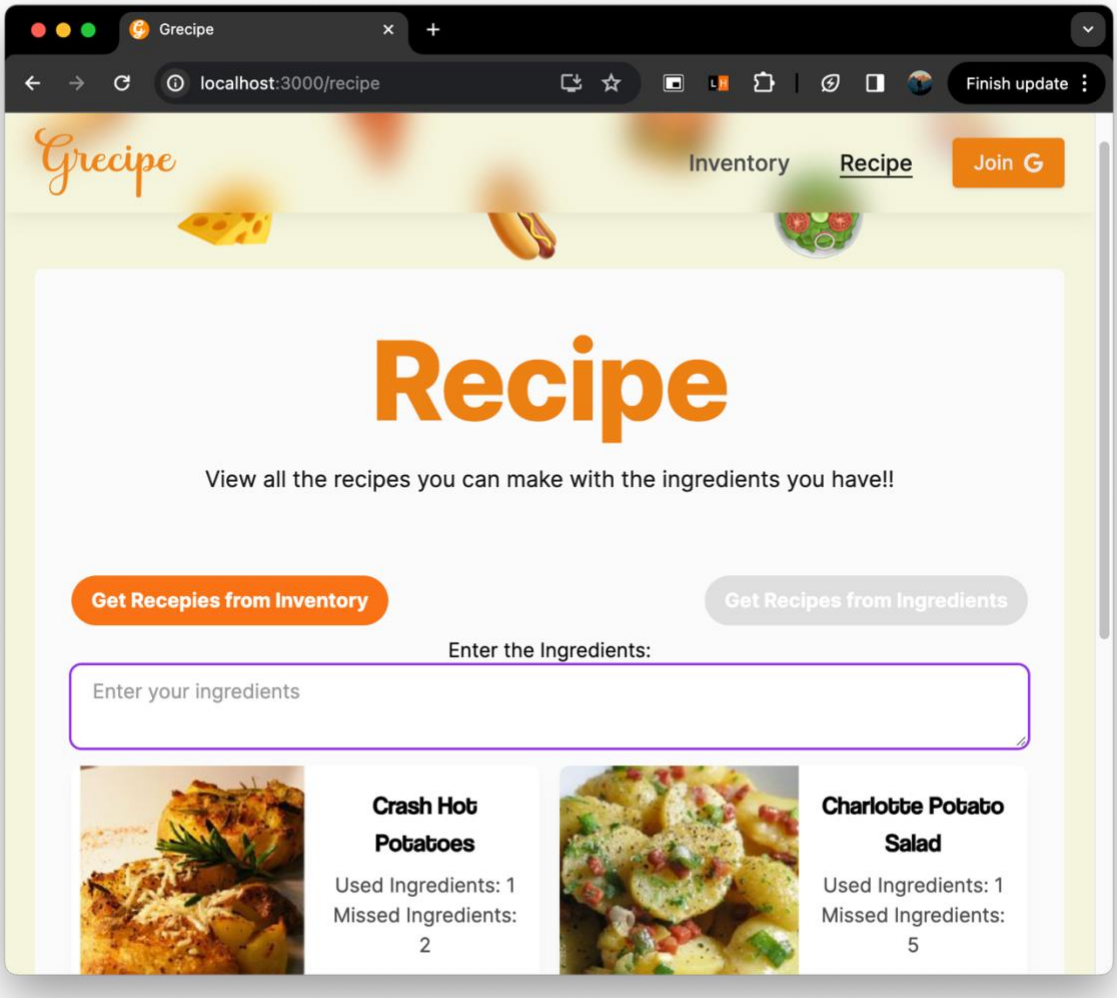


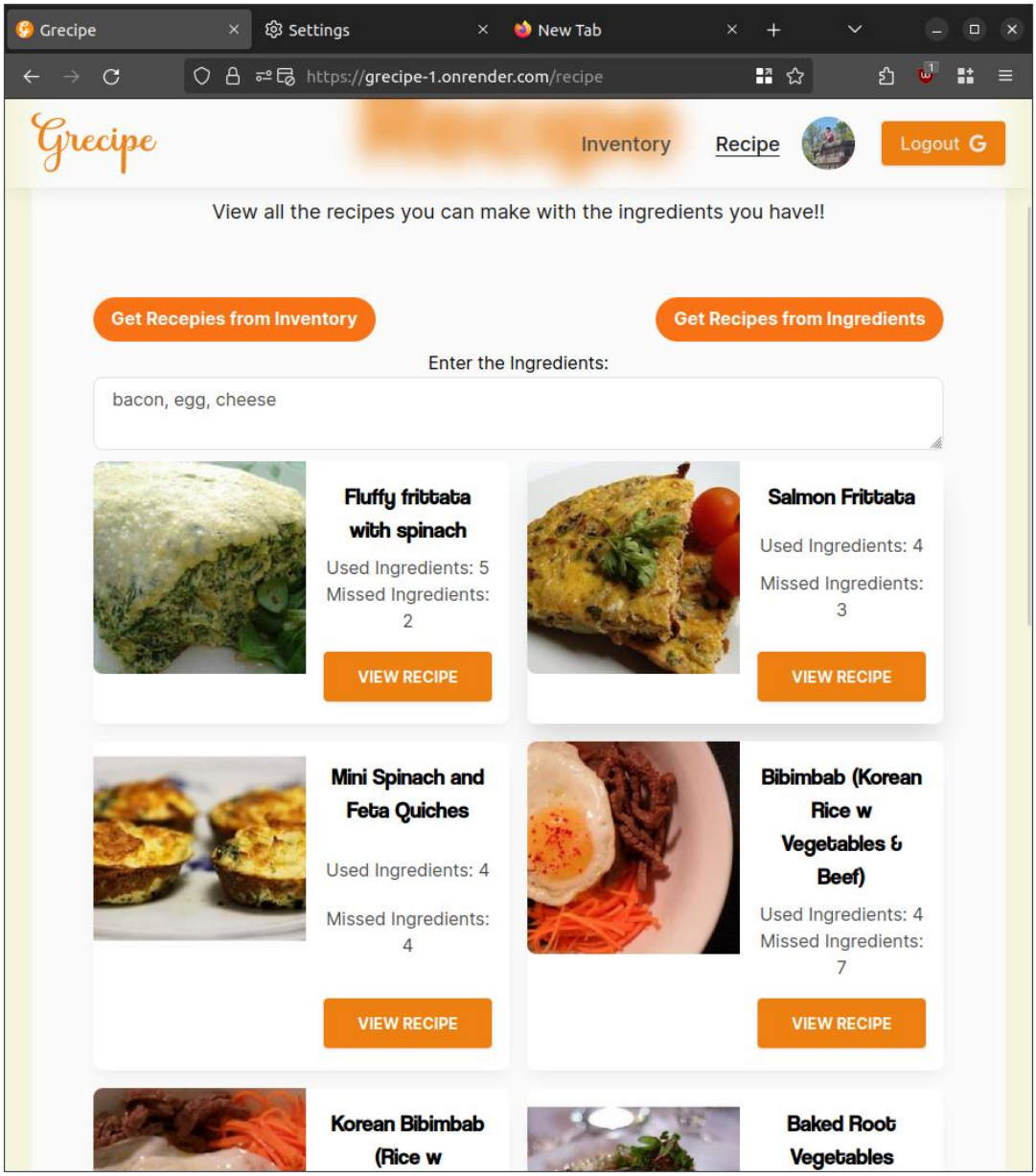
Test Case 2: User can only select image format files


Grecipe
Settings
https://grecipe-1.onrender.com/inventory
Grecipe
Inventory
Recipe
Logout
Inventory
Welcome, Mohammad Zafar!
Family Info
MANJOT'S FAMILY
YOU ARE MEMBER
MEMBER NAME
ACTION
Manjotveer Singh
REMOVE
Jagroop Singh
REMOVE
Mikhail V
REMOVE
Mohammad Zafar
REMOVE
Items Info
Upload Receipt
Save
Filter by item name
Sort by Name
Sort by Expiry Date

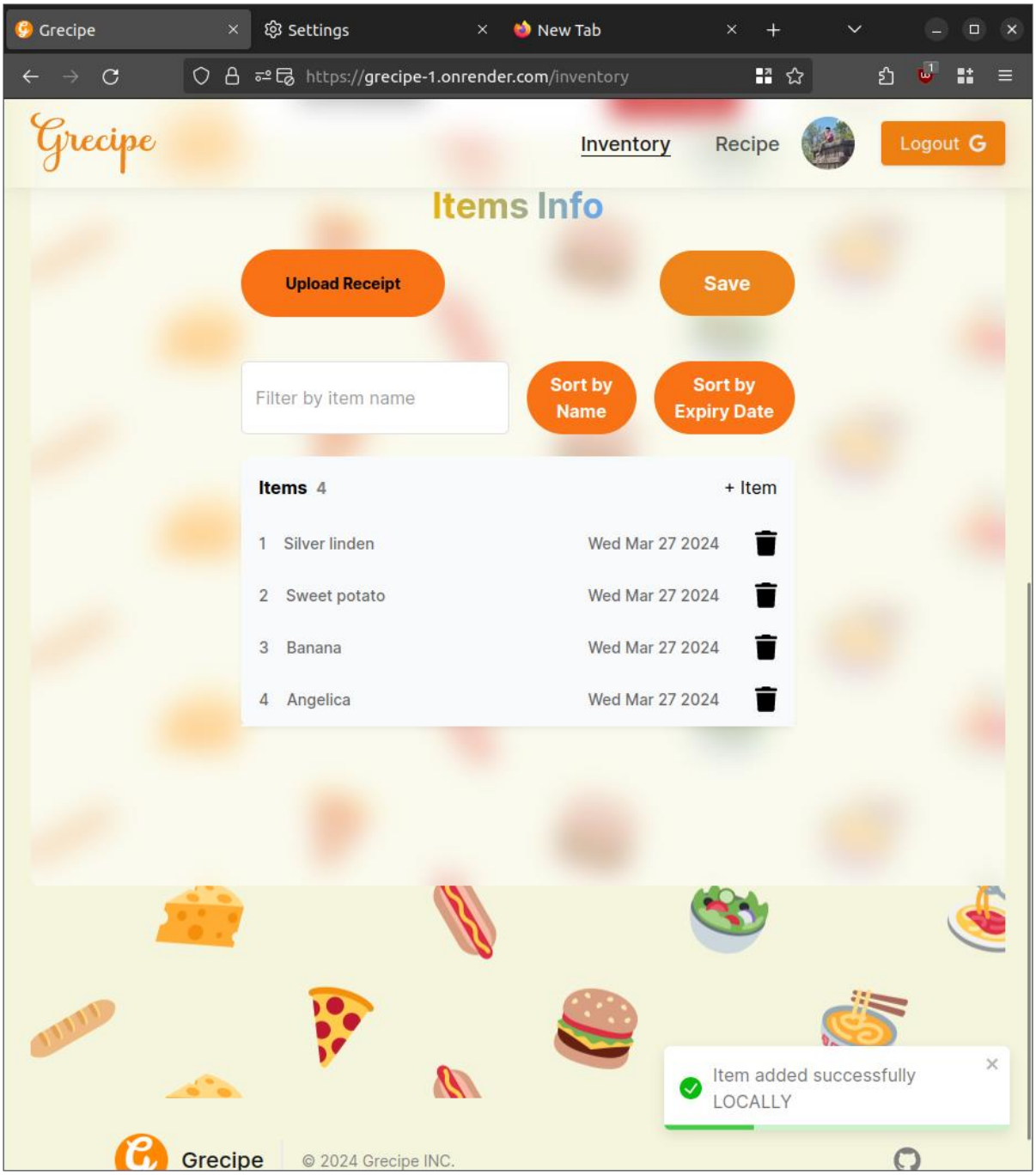


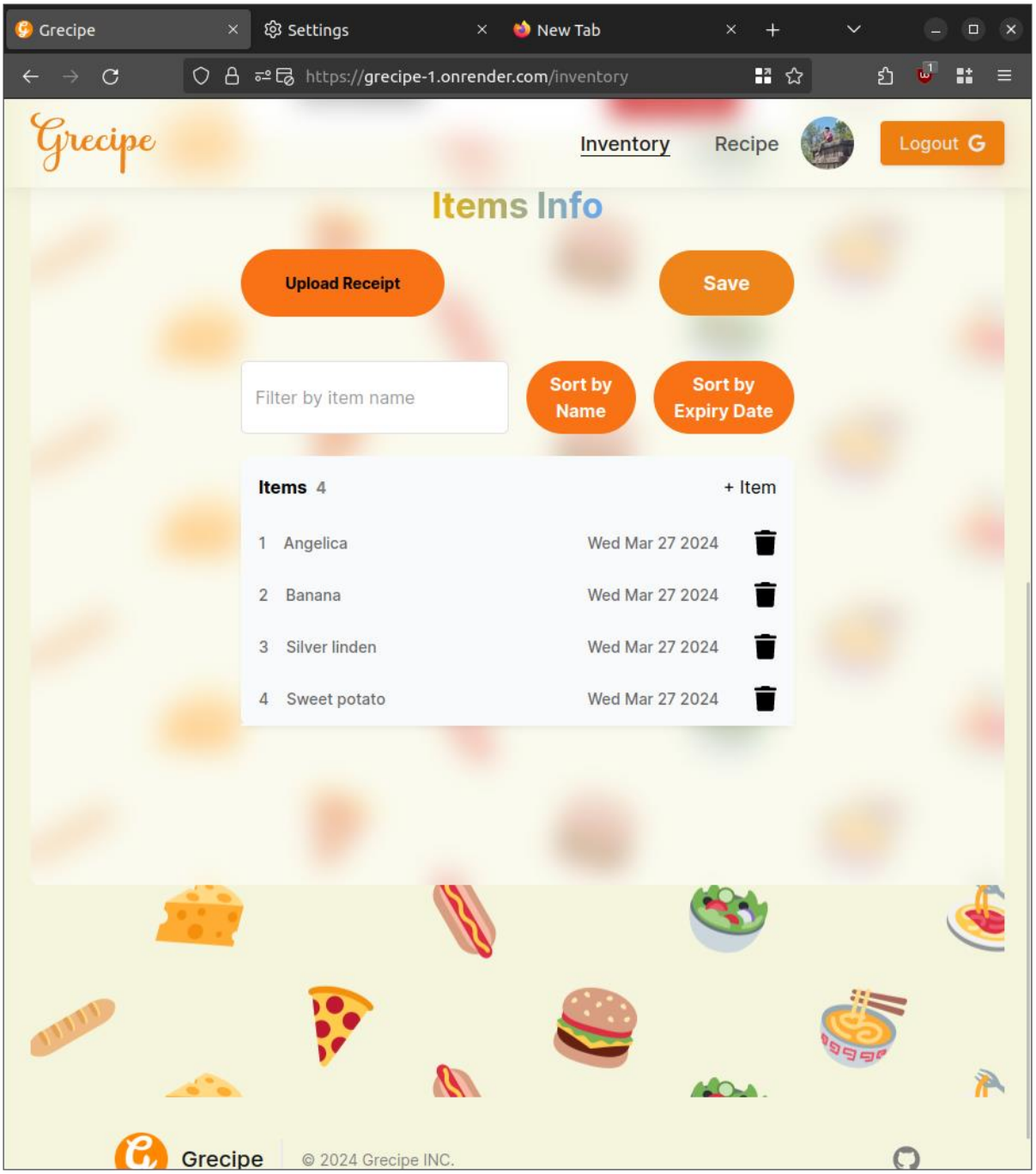
Test Case 3: User gets the recipes after writing the ingredients.


Grecipe
localhost:3000/recipe
Finish update
Grecipe
Inventory
Recipe
Join
Recipe
View all the recipes you can make with the ingredients you have!!
Get Recepies from Inventory
Get Recipes from Ingredients
Enter the Ingredients:
Enter your ingredients
Crash Hot Potatoes
Used Ingredients: 1
Missed Ingredients: 2
Charlotte Potato Salad
Used Ingredients: 1
Missed Ingredients: 5



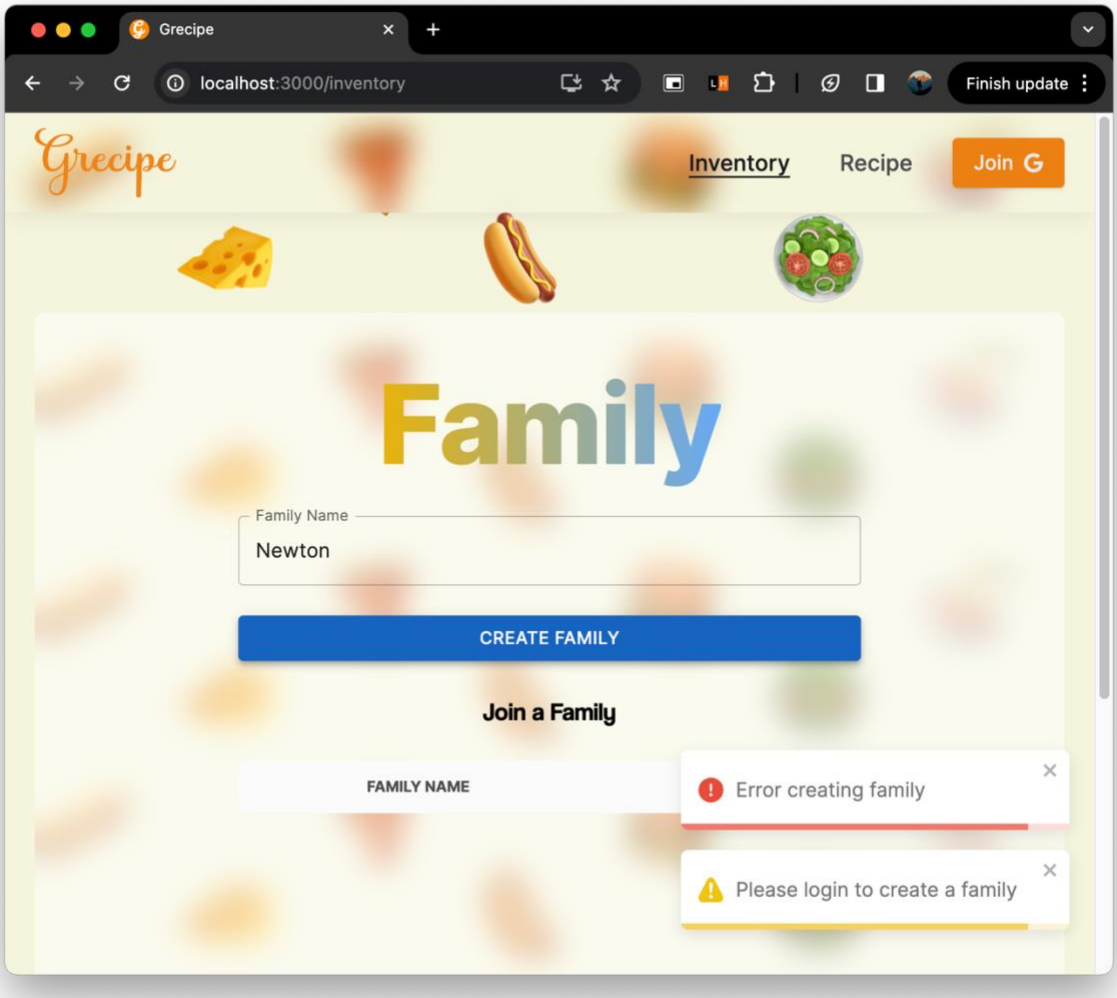
Test Case 4: User tries to sort the inventory alphabetically.


Grecipe
Settings
New Tab
https://grecipe-1.onrender.com/inventory
Grecipe
Inventory
Recipe
Logout
Items Info
Upload Receipt
Save
Filter by item name
Sort by Name
Sort by Expiry Date
Items 4
+ Item
1 Silver linden Wed Mar 27 2024
2 Sweet potato Wed Mar 27 2024
3 Banana Wed Mar 27 2024
4 Angelica Wed Mar 27 2024
Item added successfully LOCALLY
Grecipe
© 2024 Grecipe INC.



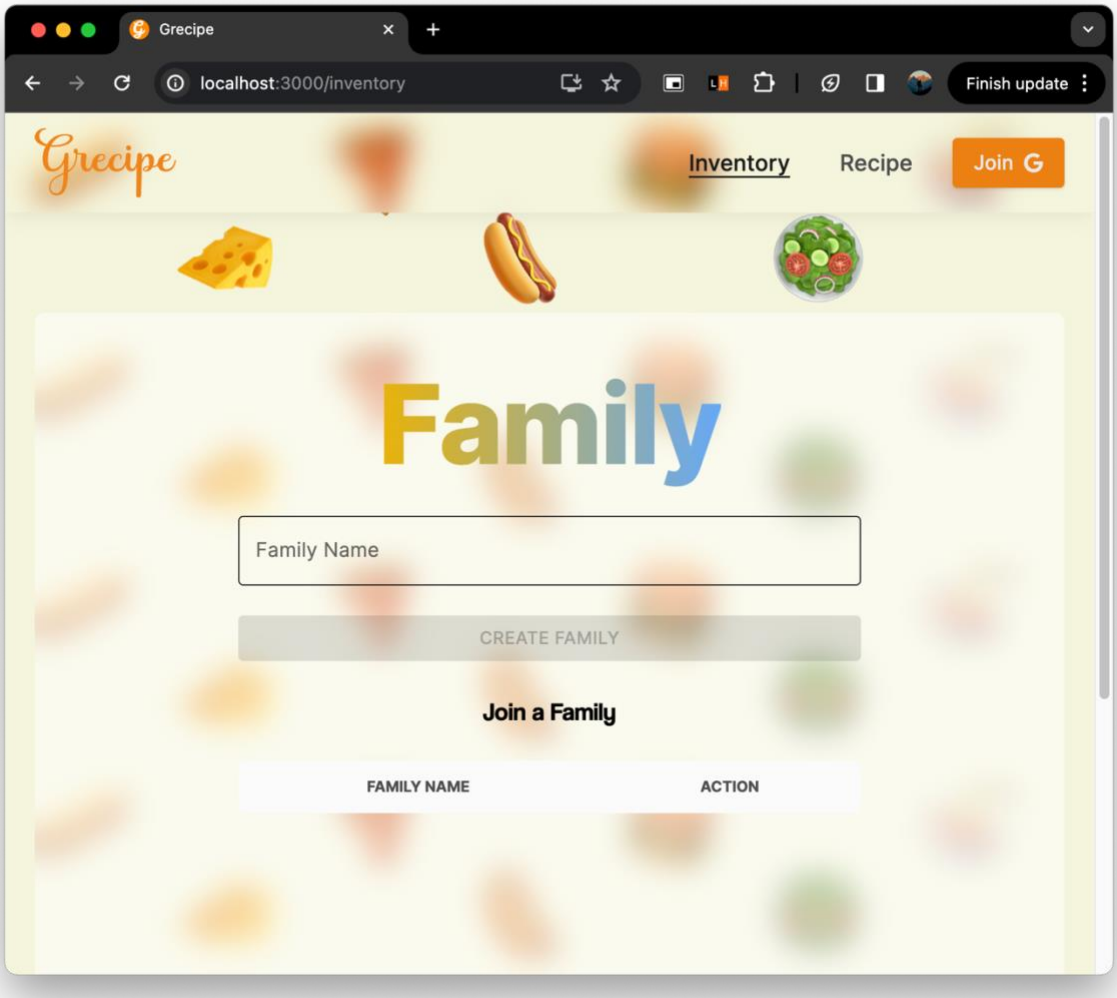
Robustness Testing

Test Case 1: User tries to create a family without logging in:



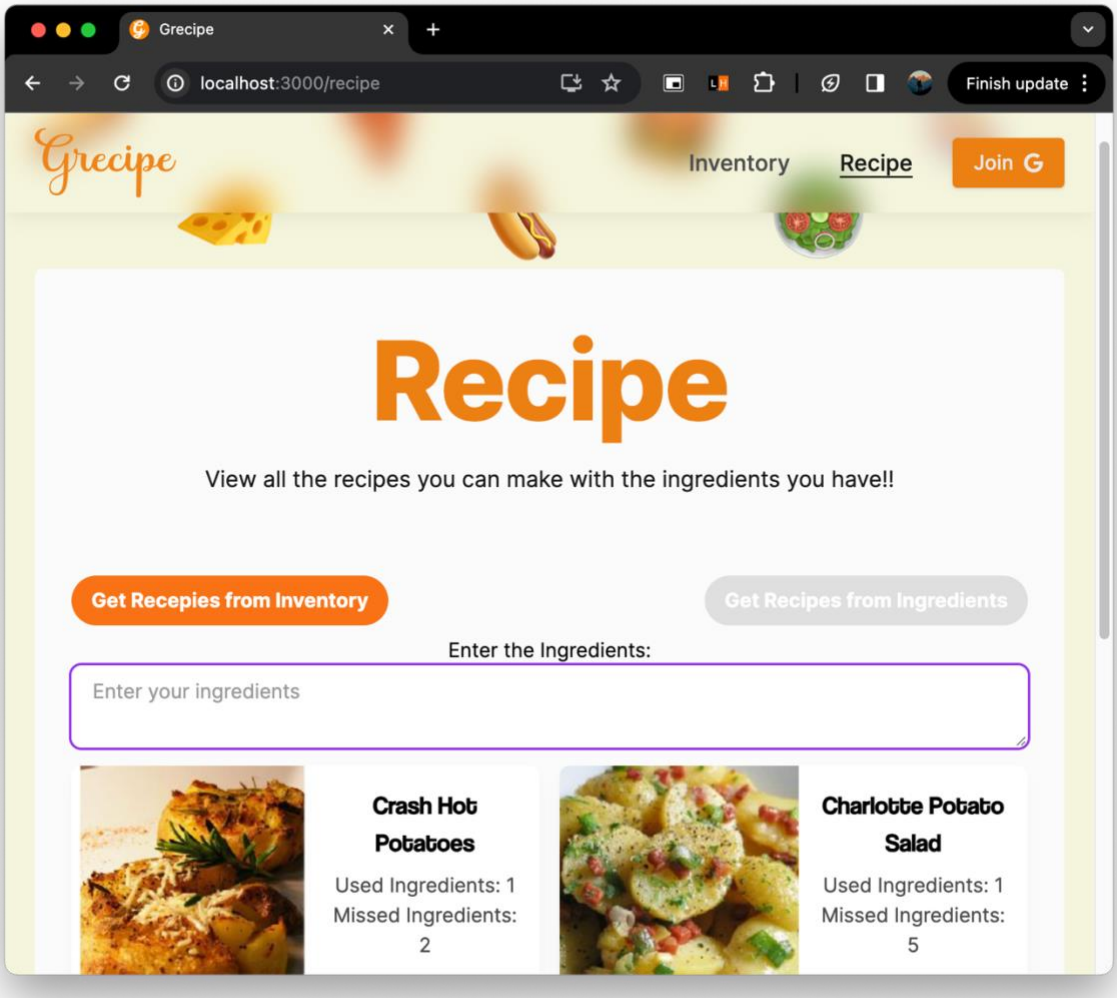
User gets an error saying they need to login to create a family.

Test Case 2: User tries to create a family without a family name.



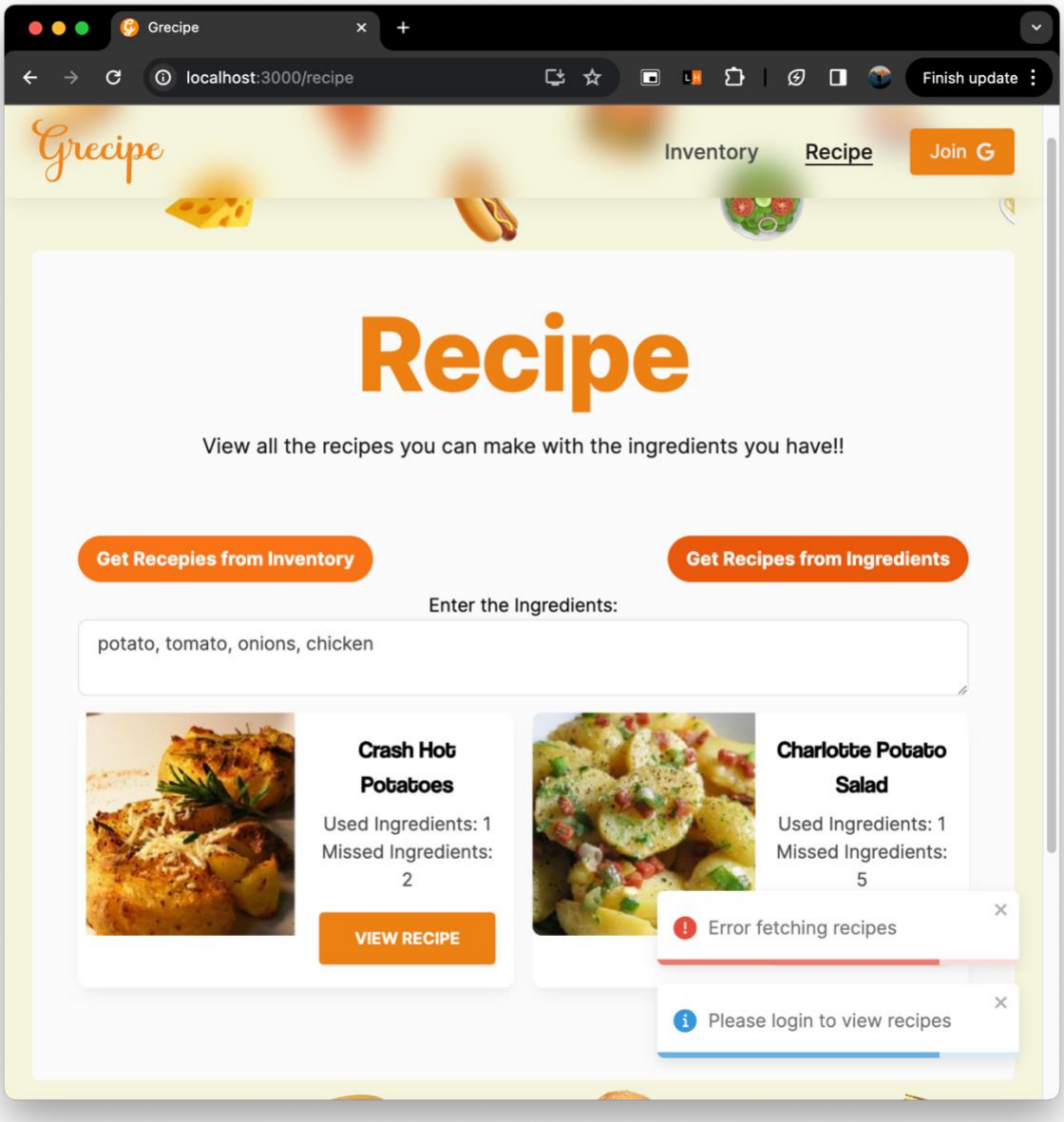
The create family button is not active.

Test Case 3: User tries to get recipes without entering the ingredients.



Get recipes button is inactive.

Test Case 4: User tries to get recipes without logging in.

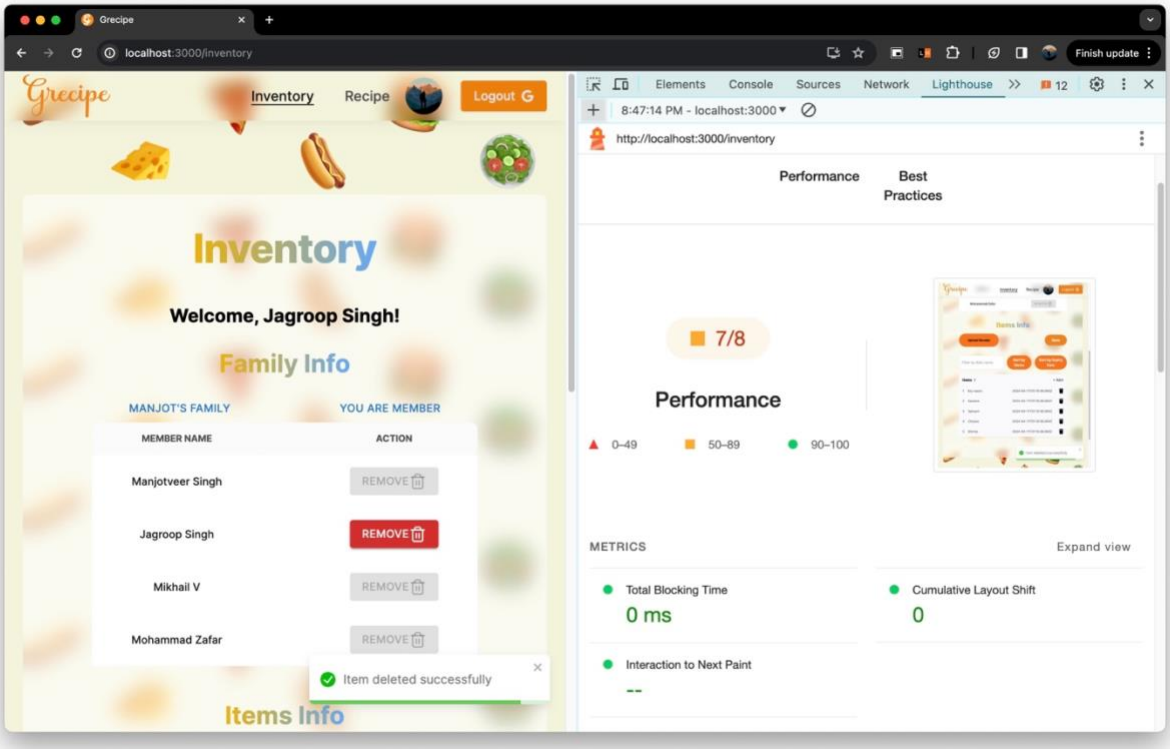


The website gives the user an error that user needs to log in.

Time Efficiency Testing

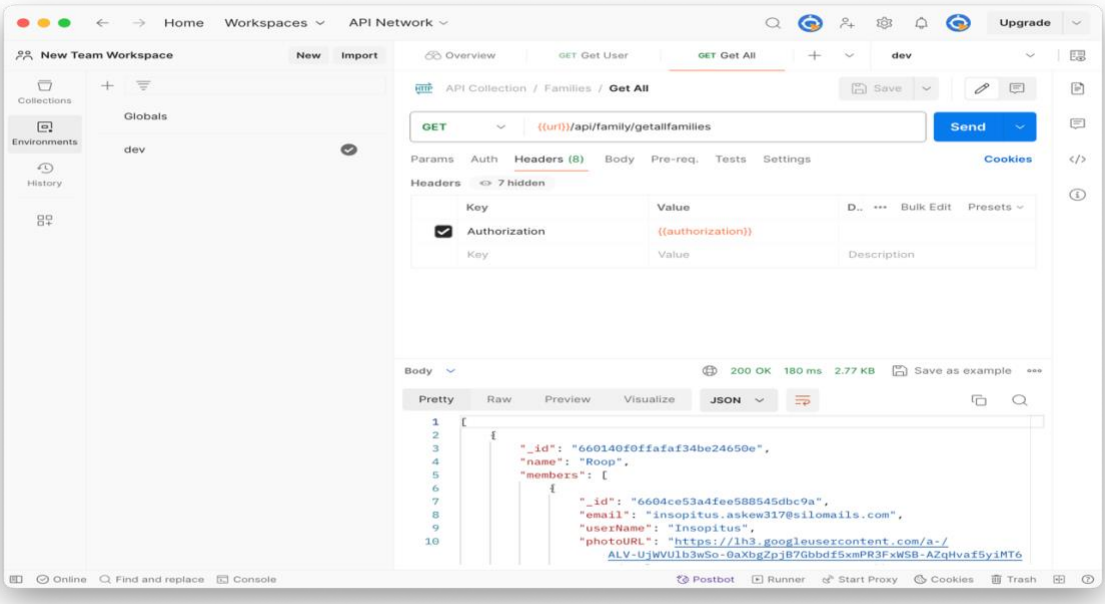
Test 1:

Method used: Lighthouse Chrome DevTools test



Test 2:

Method used: Postman response time test



Thank you

From every member of our group, we thank you Mr. Ali Bayeh our instructor to give us the opportunity to work on this project which will be really helpful in our careers. This project proved to be a wonderful opportunity for every one of us to connect and know how to work as a team.

This project not only allowed us to build coding skills but various other skills like communication, distribution of work and many more skills. Thank you once again for this opportunity.