Grecipe

Streamlining Meal Planning and Grocery Budgeting with Grecipe

Manjotveer Singh
Jagroop Singh
Kabeer Sheikh
Mohammad Zafar
Mykhaylo Vasylyev
April 4, 2024

Introduction and Problem Definition

Addressing the Challenge of Healthy Meal Preparation:

- Difficulty in meal preparation across demographics
- Budget constraints, religious beliefs, and allergies as limiting factors
- Inadequacy of online recipe videos for personalized needs
- **Our web application:** A solution for aggregating groceries and simplifying recipe access
- Features: Receipt upload, inventory management, recipe recommendation
- Motivation: Student life challenges with budgeting and food wastage

A Brief Overview

- Manage inventory/grocery list items collectively for family members in the same group.
- Family head uploads a grocery receipt, and the app extracts item details, updating the inventory.
- Get recipe recommendations based on ingredients in one' inventory
- Goal: Helping users plan meals effectively and save money on groceries.
- Users can either pull recipes from their respective inventory or add custom ingredients and request recipes based on them.

Application Benefits

Advantages Over Traditional and Modern Systems:

	Advantages	Disadvantages
Cookbooks	• Pre-organized categories • Cost-effective production	• Ignores ingredient availability • Contains outdated information • Does not consider budget
Grecipe	• Tailors recipes to available ingredients • Offers regularly updated recipes • Budget-conscious recommendations	• Requires technological proficiency • Needs internet access

	Advantages	Disadvantages
YouTube Cooking Videos	• User-generated content with community feedback • Fresh content uploaded daily	• Does not account for ingredient availability • Does not consider budget
Grecipe	• Personalized recipes from receipt scans • Budget-friendly options	• Basic tech knowledge required • Internet dependency

Requirements Elicitation and Specification

Functional Requirements for Family Head and Members

For Family Head

- Grocery Receipt Upload
- Inventory Management
- Family Member Management
- Budget Tracking

For Family Member:

- Manual Item Entry
- Recipe Access
- Inventory Interaction
- Meal Preferences Submission

Technology Used

Front-end:

- HTML
- CSS
- JavaScript
- React.js

Back-end:

- Node.js
- Express.js
- MongoDB

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.
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.

Cont. Technology Used

4. Chrome/Firefox DevTools
    a. Chrome/Firefox DevTools are the tools available directly on google chrome and mozilla firefox 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.
5. 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.
6. 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

Software Qualities

Ensuring Excellence in Application Performance:

- **For Family Head**:
  - Correctness: Secure and verified data handling
  - Robustness: Comprehensive error management and data protection
  - Performance: Efficient processing and data retrieval
- **For Family Member**:
  - Correctness: Accurate recipe generation and data validation
  - Robustness: Error handling and security measures
  - Performance: Prompt item entry and recipe access

Inventory

Join/leave a family.

Upload receipt

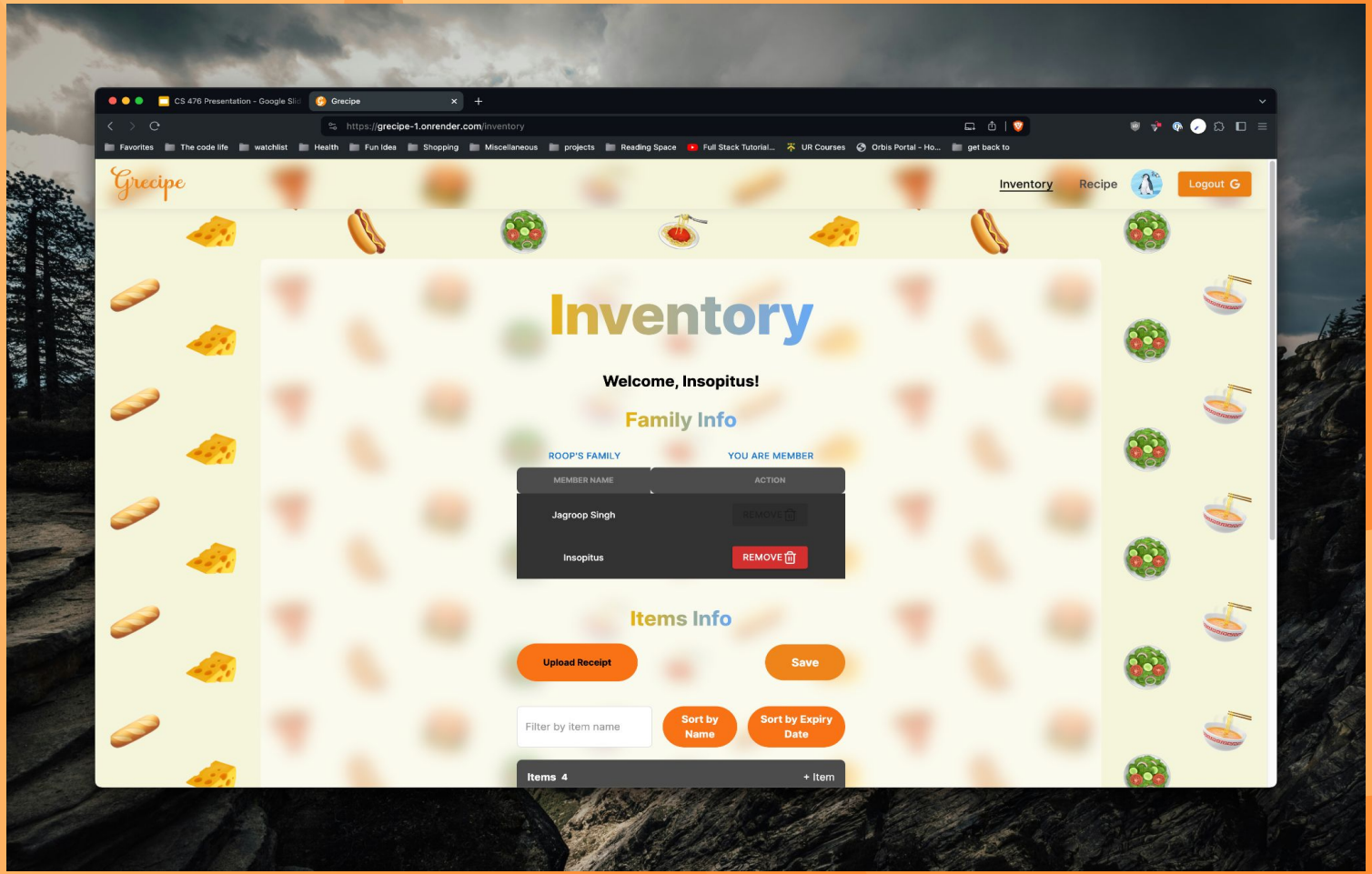
Sort ingredients in current family

Secure login and access control mechanisms

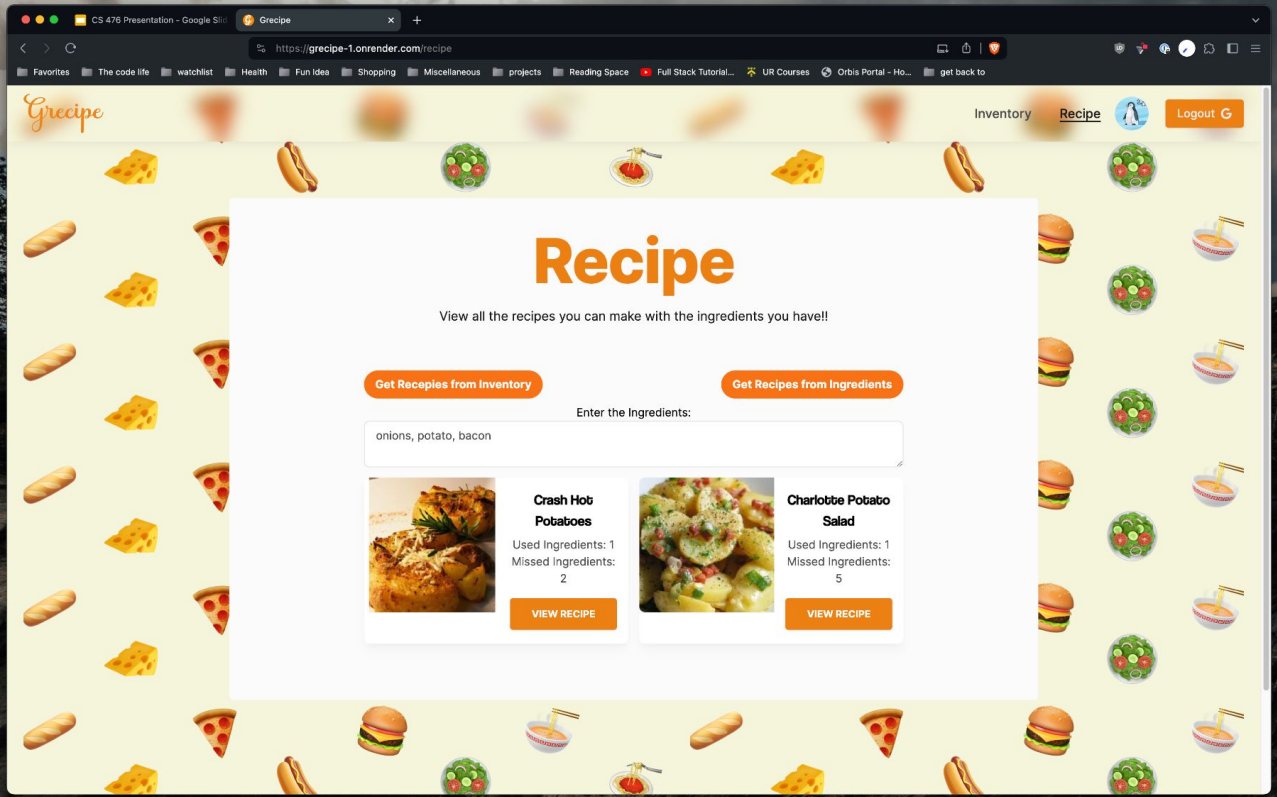
Add, remove, update items

Expiration date tracking

Category organization



Recipe



View recipes from custom list of ingredients

View recipes from ingredients from inventory

Personalized recipe suggestions for users

Grecipe’s Error Handling mechanisms:

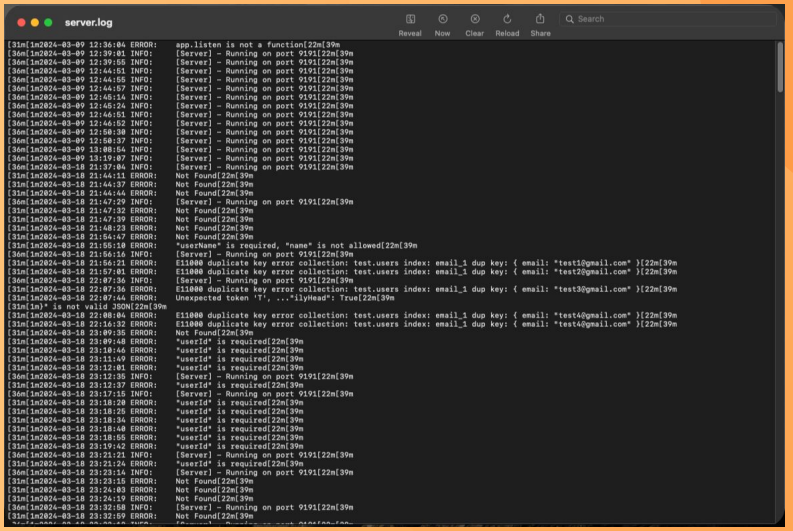
Grecipe implements robust error handling mechanisms to ensure the reliability and stability of the application, particularly during database operations.

```
try {
    // Database operation that may throw synchronous errors
    const result = await Recipe.findOneAndUpdate({ _id: recipeId }, updateData);
    res.status(200).json({ success: true, data: result });
} catch (error) {
    // Handle synchronous errors gracefully
    res.status(500).json({ success: false, error: "Internal Server Error" });
}

```

Here, we try to the catch-block method to wrap our database operation which allows us to detect and handle any synchronous errors that may occur.

Promises and Async/Await Syntax for Asynchronous Errors:




```
[31m[1m2024-03-09 12:36:04 ERROR:    app.listen is not a function[22m[39m
[36m[1m2024-03-09 12:39:01 INFO:     [Server] - Running on port 9191[22m[39m
[36m[1m2024-03-09 12:39:55 INFO:     [Server] - Running on port 9191[22m[39m
[36m[1m2024-03-09 12:44:51 INFO:     [Server] - Running on port 9191[22m[39m
[36m[1m2024-03-09 12:44:55 INFO:     [Server] - Running on port 9191[22m[39m
[36m[1m2024-03-09 12:44:57 INFO:     [Server] - Running on port 9191[22m[39m
[36m[1m2024-03-09 12:45:14 INFO:     [Server] - Running on port 9191[22m[39m
[36m[1m2024-03-09 12:45:24 INFO:     [Server] - Running on port 9191[22m[39m
[36m[1m2024-03-09 12:46:51 INFO:     [Server] - Running on port 9191[22m[39m
[36m[1m2024-03-09 12:46:52 INFO:     [Server] - Running on port 9191[22m[39m
[36m[1m2024-03-09 12:50:30 INFO:     [Server] - Running on port 9191[22m[39m
[36m[1m2024-03-09 12:50:37 INFO:     [Server] - Running on port 9191[22m[39m
[36m[1m2024-03-09 13:08:54 INFO:     [Server] - Running on port 9191[22m[39m
[36m[1m2024-03-09 13:19:07 INFO:     [Server] - Running on port 9191[22m[39m
[36m[1m2024-03-10 21:37:04 INFO:     [Server] - Running on port 9191[22m[39m
[31m[1m2024-03-10 21:44:11 ERROR:    Not Found[22m[39m
[31m[1m2024-03-10 21:44:37 ERROR:    Not Found[22m[39m
[31m[1m2024-03-10 21:44:44 ERROR:    Not Found[22m[39m
[36m[1m2024-03-10 21:47:29 INFO:     [Server] - Running on port 9191[22m[39m
[31m[1m2024-03-10 21:47:32 ERROR:    Not Found[22m[39m
[31m[1m2024-03-10 21:47:39 ERROR:    Not Found[22m[39m
[31m[1m2024-03-10 21:48:23 ERROR:    Not Found[22m[39m
[31m[1m2024-03-10 21:54:47 ERROR:    Not Found[22m[39m
[31m[1m2024-03-10 21:55:10 ERROR:    "userName" is required, "name" is not allowed[22m[39m
[36m[1m2024-03-10 21:56:16 INFO:     [Server] - Running on port 9191[22m[39m
[31m[1m2024-03-10 21:56:21 ERROR:    E11000 duplicate key error collection: test.users index: email_1 dup key: { email: "test1@gmail.com" }[22m[39m
[31m[1m2024-03-10 21:57:01 ERROR:    E11000 duplicate key error collection: test.users index: email_1 dup key: { email: "test2@gmail.com" }[22m[39m
[36m[1m2024-03-10 22:07:56 INFO:     [Server] - Running on port 9191[22m[39m
[31m[1m2024-03-10 22:07:36 ERROR:    E11000 duplicate key error collection: test.users index: email_1 dup key: { email: "test3@gmail.com" }[22m[39m
[31m[1m2024-03-10 22:07:44 ERROR:    Unexpected token 'T', ...*lyHead": True[22m[39m
[31m1m)" is not valid JSON[22m[39m
[31m[1m2024-03-10 22:08:04 ERROR:    E11000 duplicate key error collection: test.users index: email_1 dup key: { email: "test4@gmail.com" }[22m[39m
[31m[1m2024-03-10 22:16:32 ERROR:    E11000 duplicate key error collection: test.users index: email_1 dup key: { email: "test4@gmail.com" }[22m[39m
[31m[1m2024-03-10 23:09:35 ERROR:    Not Found[22m[39m
[31m[1m2024-03-10 23:09:48 ERROR:    "userId" is required[22m[39m
[31m[1m2024-03-10 23:10:46 ERROR:    "userId" is required[22m[39m
[31m[1m2024-03-10 23:11:49 ERROR:    "userId" is required[22m[39m
[31m[1m2024-03-10 23:12:01 ERROR:    "userId" is required[22m[39m
[36m[1m2024-03-10 23:12:35 INFO:     [Server] - Running on port 9191[22m[39m
[31m[1m2024-03-10 23:12:37 ERROR:    "userId" is required[22m[39m
[36m[1m2024-03-10 23:17:15 INFO:     [Server] - Running on port 9191[22m[39m
[31m[1m2024-03-10 23:18:20 ERROR:    "userId" is required[22m[39m
[31m[1m2024-03-10 23:18:25 ERROR:    "userId" is required[22m[39m
[31m[1m2024-03-10 23:18:34 ERROR:    "userId" is required[22m[39m
[31m[1m2024-03-10 23:18:40 ERROR:    "userId" is required[22m[39m
[31m[1m2024-03-10 23:18:55 ERROR:    "userId" is required[22m[39m
[31m[1m2024-03-10 23:19:42 ERROR:    "userId" is required[22m[39m
[36m[1m2024-03-10 23:21:21 INFO:     [Server] - Running on port 9191[22m[39m
[31m[1m2024-03-10 23:21:24 ERROR:    "userId" is required[22m[39m
[36m[1m2024-03-10 23:23:14 INFO:     [Server] - Running on port 9191[22m[39m
[31m[1m2024-03-10 23:23:15 ERROR:    Not Found[22m[39m
[31m[1m2024-03-10 23:24:03 ERROR:    Not Found[22m[39m
[31m[1m2024-03-10 23:24:19 ERROR:    Not Found[22m[39m
[36m[1m2024-03-10 23:32:58 INFO:     [Server] - Running on port 9191[22m[39m
[31m[1m2024-03-10 23:32:59 ERROR:    Not Found[22m[39m
```

```
app.get('/recipes', async (req, res) => {
    try {
        // Asynchronous database operation
        const recipes = await Recipe.find();
        res.status(200).json({ success: true, data: recipes });
    } catch (error) {
        // Handle asynchronous errors gracefully
        res.status(500).json({ success: false, error: "Internal Server Error" });
    }
});

```

Here, the async/await syntax is used with promises to handle asynchronous database operations. So, in case any errors occur during these operations they are caught in the catch block and appropriately handled.

Ensuring Data Integrity and Consistency in Grecipe:

Grecipe prioritizes data integrity and consistency to provide users with reliable and accurate information.

```
const mongoose = require('mongoose');

// Define recipe schema with required fields
const recipeSchema = new mongoose.Schema({
    title: { type: String, required: true },
    description: String,
    ingredients: { type: [String], required: true },
    instructions: { type: [String], required: true },
    // Additional schema fields...
});

// Compile recipe model
const Recipe = mongoose.model('Recipe', recipeSchema);
```

By defining schema with required fields and data types, we wanted to make sure that all recipes stored in the database respect and follow a consistent structure, minimizing data inconsistencies.

This helped us make sure that invalid data is not inserted into the database and helps us maintain data integrity and thus helped us reduce risk of data corruption.

Acceptance Testing

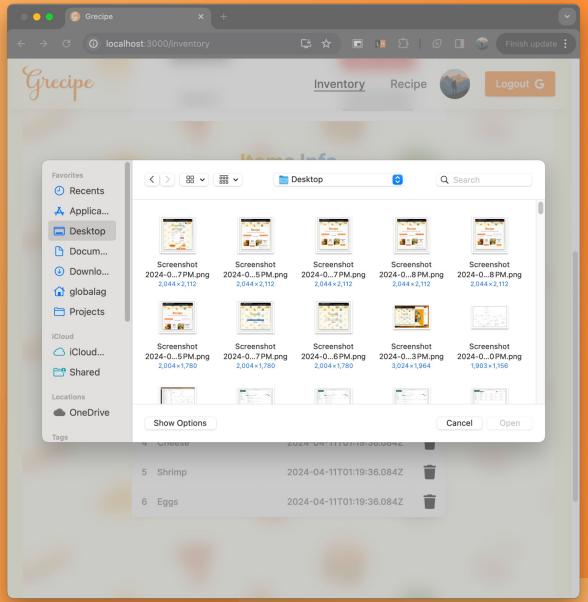
Ensuring our code meets user expectations

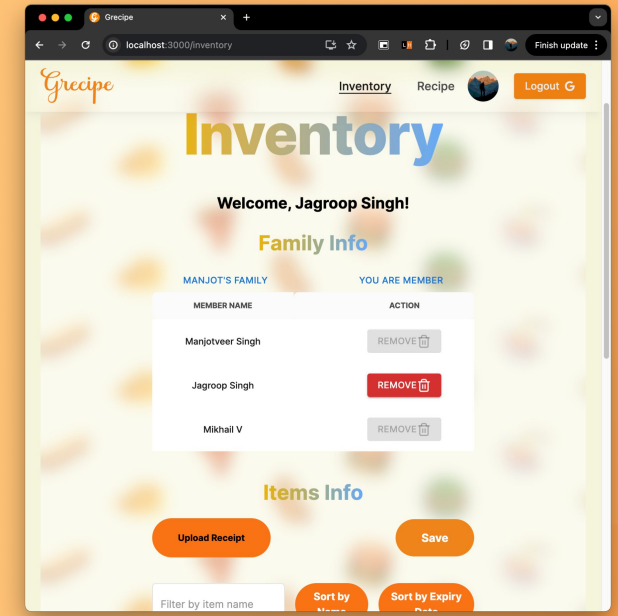
Correctness Testing

1. **User can only select images when uploading receipt**

When the user is selecting a receipt to upload, they are only able to add Images (.png, .jpg)

This ensures that we are not receiving incorrectly formatted documents that the program can't process



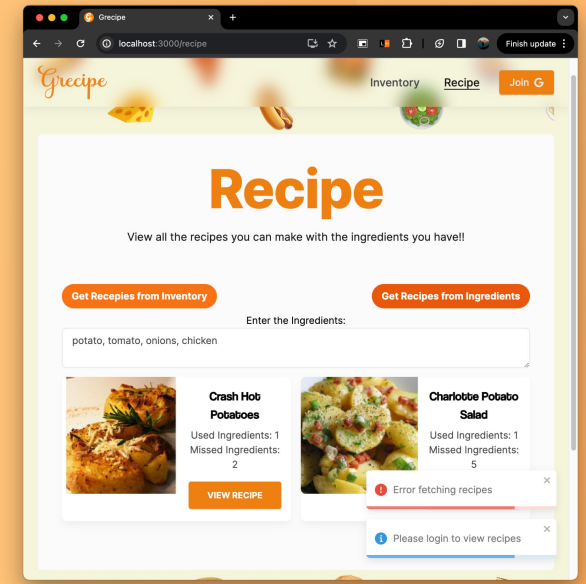


2. **Members cannot remove other members**

When joining a family, a member can only remove themselves and not others

Only the family head, which is the one who created the family, has the ability to remove members

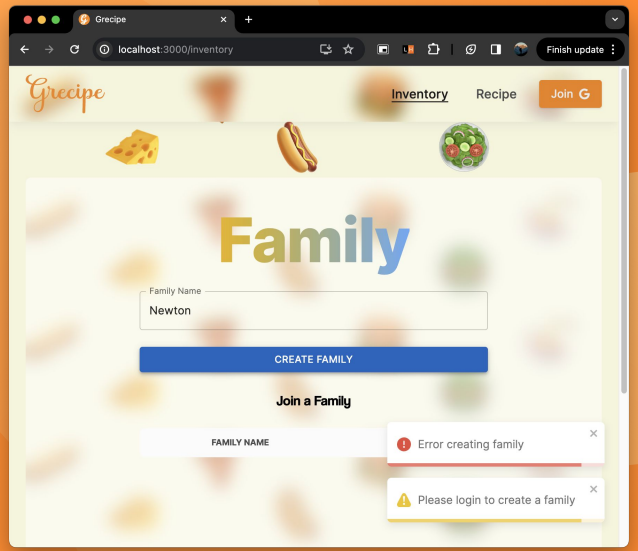
Robustness Tests



1. User must login to generate recipes

When the user is selecting a receipt to upload, they are only able to add Images (.png, .jpg)

This ensures that we are not receiving incorrectly formatted documents that the program can't process



2. User must login to generate recipes

When the user tries to create a family, they must be logged in

If the user tries to create a family without being logged in, the request will be refused and an error will appear

Conclusion

- Grecipe tackles the difficulty in meal preparation by aggregating groceries and simplifying recipe access.
- Receipt upload, inventory management, and personalized recipe recommendations.
- Tailored recipes, updated recommendations, and robust data handling.
- Implements catch-block method and schema validation for data integrity.
- Utilized Visual Studio Code, Postman, MongoDB Compass, DevTools, GitHub, and Lucidchart.
- Ensures correctness, robustness, and efficiency for both family heads and members.

With these measures in place, Grecipe offers users a dependable platform for efficient meal planning and grocery management.

THANKS!