Your Project Title

Final Project Technical Report

SE/COM S 3190 – Construction of User Interfaces
Spring 2025

Team Members:
Cai Chen - achen23@iastate.edu
Megan Chng - mkchng@iastate.edu

May 11, 2025

1. Introduction

Recipes are hard to keep track of, whether it's in your notes app, brain, or even a physical book, it's difficult to keep track of everything. The purpose of this project is to streamline a way to store recipes in a simple yet pretty way. The expected users are everyone from all ages who enjoy cooking. Our goals for this project was to learn more about web development, specifically through typescript, react, next.js, material UI, mongoDB, express, and node.js. Our project was inspired by our own experiences with keeping tack of recipes.

2. Project Description

Explain major features, user flow, CRUD operations, and entities affected.

Major features:

- User Authentication: Creating/signing into an account & editing account information
- Recipes: Creating, Reading, Editing, Deleting recipes that are stored into the user's account.
- Grocery Mode: A way for users to shop specifically for certain recipes which also combines ingredients if multiple recipes have the same ones. (No CRUD)
- MarketPlace: A way for users to buy or make recipe books from other users. After checkout, the recipes get added to the users recipe page for them to use.
- Redirect: If a user isn't signed in and tries to access a page they aren't supposed to, it brings them to a redirect page that has a button to go back to the home page. (No CRUD)
- 404: A 404 page not found if a user accidentally types a page that doesn't exists into the URL (No CRUD)

3. File and Folder Architecture

The file structure is a basic frontend/, backend/, and documents setup. Here is the tree diagram with extra information:

```
.
├── Frontend/
│   ├── Public: all of our images
│   ├── Src: source files
│   ├── App: All of our individual pages, also stores the home page and 404 page/
│   │   ├── Auth: Authintication Page
│   │   ├── Grocery: Grocery feature page
│   │   ├── Profile: Page to edit your profile
│   │   ├── Recipes: File that stores all of the user's recipe/
│   │   │   ├── [id]: individual recipe page based off of the recipe ID
│   │   │   ├── add: A page to add a new recipe
│   │   │   ├── button components: place to hold reused components in the recipes section
│   │   │   ├── card components:
│   │   │   └── edit/[id]: Individual Recipe page that edits a recipe based off of it's id
│   │   └── Redirect: The redirect page
│   ├── Components: All of our components that get used accross multiple pages
│   ├── Context: All of our contexts to store data accross multiple pages (i.e user auth & cart)
│   ├── Lib: Our AXOIS API calls
│   └── Utils: A place for the redirect logic
├── Backend/
│   ├── src: source files
│   ├── api: api router for vercel deployment after this class is finished
│   ├── controllers: stores all of our controllers for recipes, users, and store
│   ├── models: stores all of our schemas for recipes, users, and store
│   ├── routes: stores our api routes/endpoints for recipes, users, and store
│   └── types: stores our typescript types for recipes, users, and store
└── Documents: Stores all our documents and videos
```

4. Code Explanation and Logic Flow

4.1. Frontend–Backend Communication

We handled API requests by containing all of our major request methods into a single “lib” directory. The calls themselves were made using the axios react library. When we need to make the calls in each page, we would just import the method and use it when necessary. For example in the recipes page, we imported the recipeApi from the lib directory. When the user first opens the page, it uses the recipeApi.getUserRecipes that to make a call that fetches all of the user’s recipes. The same is true for the rest of the components, though some only make a call when a button is pressed, like when creating/editing a recipe.

4.2. React Component Structure

We use components for cards that are used multiple times so we don’t have to rewrite the same code multiple times. For example the Navbar is constantly used for all of the pages, or the a recipe card since a user can have multiple recipes. The store uses a card and multiple modals to make the code easier to read and navigate. We use props and states all throughout the project. Going back to the store card, we use states for the each category of text, i.e name, description, cost, etc. We use props to help us display data from the database. For example the recipes use the ingredients, name, description, etc prop data to display the respective info.

4.3. Database Interaction

We ended up using mongoDB as our database. We wanted something simple and malleable that can store multiple ingredients and steps for a single recipe without being overly complex - so something NoSQL. The database itself is a single cluster separated into 3 schemas/models. They are the recipe model, user model, and the stores model. The user model has a connection to the recipe model that stores the _id of the recipe that the user has. The same is true for store and recipe. The data base is used to store all of our dynamic information such as these three models.

4.4. Code Snippets

Recipe Card: This card uses the recipe card data to make a prop that help’s display a preview card of the recipe.

```
export interface RecipeCardProps {
  recipeID: string;
  name: string;
  description: string;
  recipeTags?: string[];
  onClick?: (recipeID: string) => void;
}

export default function RecipeCard({
  recipeID,
  name,
  description,
  recipeTags = [],
  onClick,
}: RecipeCardProps) {
... ...
```

```
      <CardContent sx={{ flexGrow: 1 }}>
        <Typography variant="h6" component="div" sx={{ mb: 1 }}>
          {name}
        </Typography>
        <Typography
          variant="body2"
          color="text.secondary"
          sx={{
            mb: 2,
            height: 40,
            overflow: 'hidden',
            textOverflow: 'ellipsis',
          }}
        >
          {description.length > 100
            ? `${description.substring(0, 100)}...`
            : description}
        </Typography>

        {recipeTags && recipeTags.length > 0 && (
          <Box sx={{ display: 'flex', flexWrap: 'wrap', gap: 0.5, mb: 1 }}>
            {recipeTags.slice(0, 2).map((tag) => (
              <Chip
                key={tag}
                label={tag}
                size="small"
                sx={{ fontSize: '0.7rem' }}
              />
            ))}
            {recipeTags.length > 2 && (
              <Chip
                label={`+${recipeTags.length - 2}`}
                size="small"
                variant="outlined"
                sx={{ fontSize: '0.7rem' }}
              />
            )}
          </Box>
        )}
      </CardContent>
```

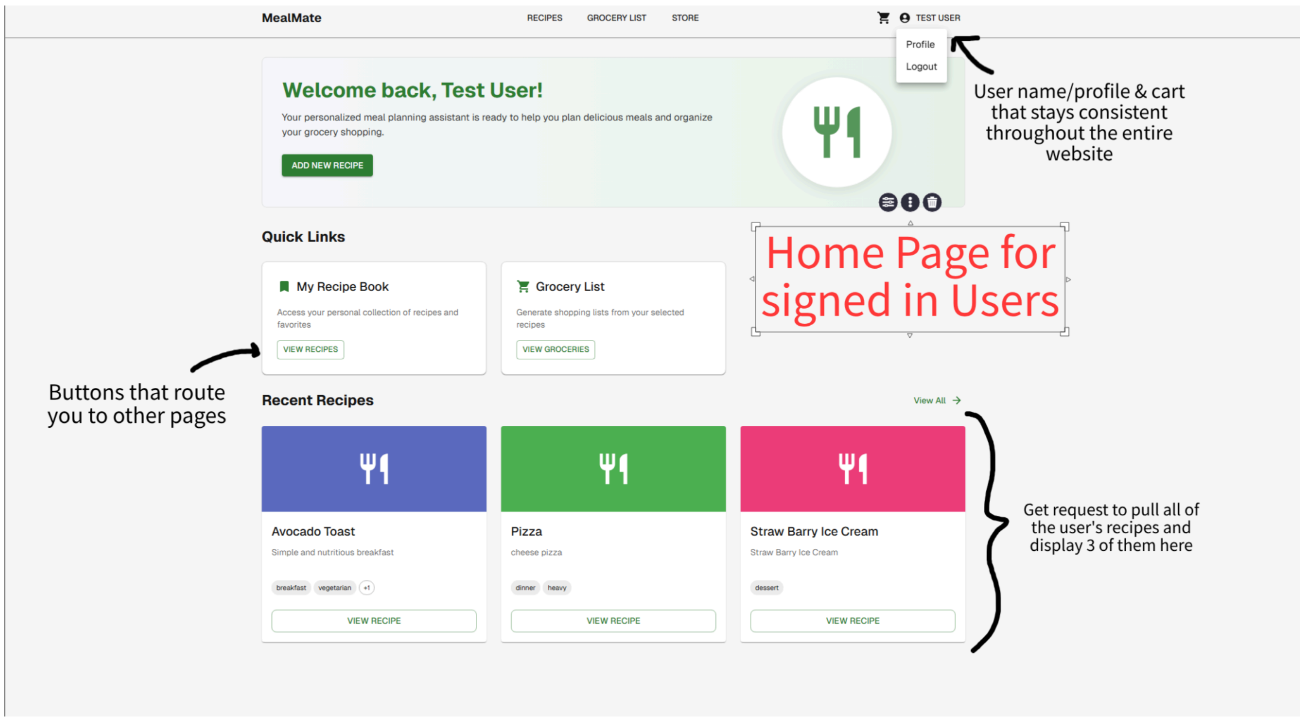
Store Controller: This snippet has 2 methods in it, one to get an item by it's ID and the other is to create a new item to store in the database.

```
  async getItemById(req: Request, res: Response) {
    try {
      const item = await StoreModel.findById(req.params.id);
      if (!item) {
        return res.status(404).json({ message: 'Store item not found' });
      }
      res.json(item);
    } catch (error) {
      console.error('Error fetching store item:', error);
      res.status(500).json({ message: 'Error fetching store item' });
    }
  }

  async createItem(req: Request, res: Response) {
    try {
      const itemData: MarketCreate = req.body;

      const newItem = new StoreModel({
        ...itemData,
        description: Array.isArray(itemData.description)
          ? itemData.description
          : [itemData.description],
      });

      const savedItem = await newItem.save();
      res.status(201).json(savedItem);
    } catch (error) {
      console.error('Error creating store item:', error);
      res.status(500).json({ message: 'Error creating store item' });
    }
  }
```

5. Web View Screenshots and Annotations

Home page for authenticated users

Description: This page is basically a user dashboard. It lets the user see quick links and recipes that they might go to.



Your personalized meal planning assistant is ready to help you plan delicious meals and organize your grocery shopping.

Access your personal collection of recipes and favorites

Generate shopping lists from your selected recipes

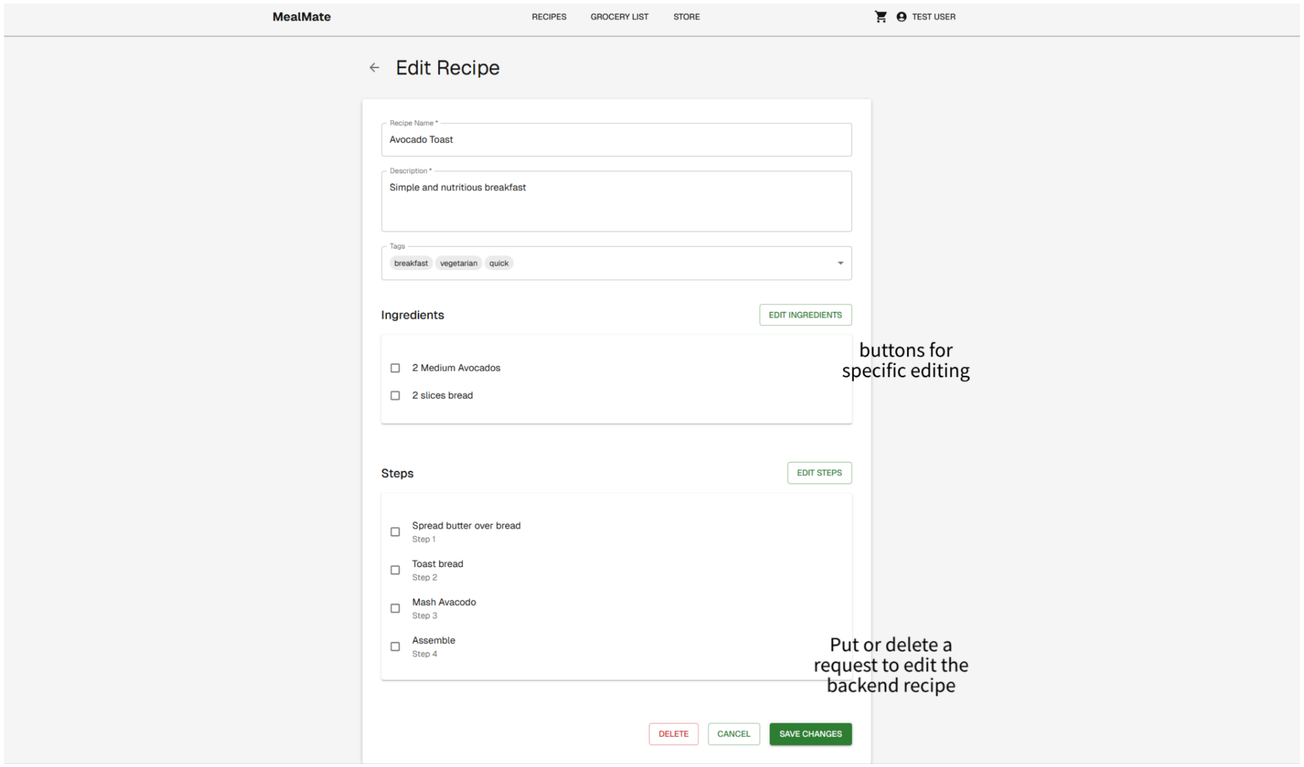
Simple and nutritious breakfast

cheese pizza

Straw Barry Ice Cream

Edit Recipe Page

Description: This page allows users to edit a recipe they made. It's exactly the same as creating a recipe though the forums are filled with the data from the recipe.



Recipe Name *: Avocado Toast

Description *: Simple and nutritious breakfast

Tags: breakfast, vegetarian, quick

Ingredients

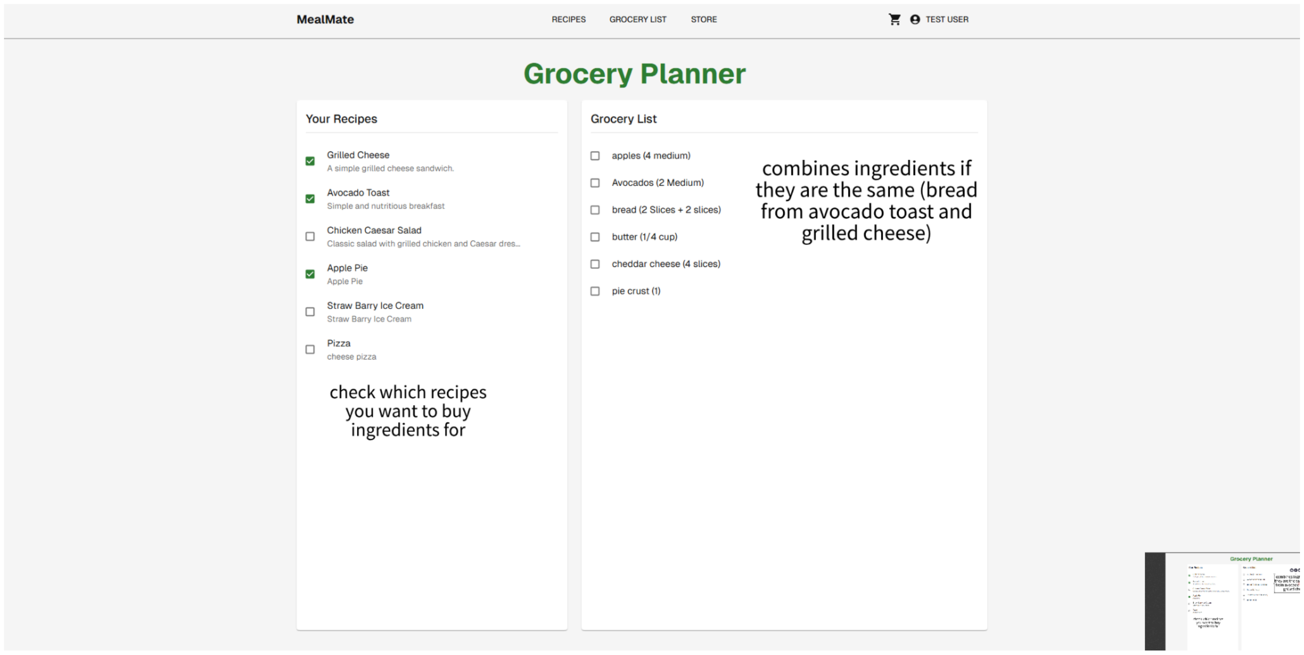
- 2 Medium Avocados
- 2 slices bread

Steps

- Spread butter over bread (Step 1)
- Toast bread (Step 2)
- Mash Avacodo (Step 3)
- Assemble (Step 4)

Edit Recipe Page

Description: This page allows the user to buy ingredients for recipes they want to make. It combines similar ingredients.


MealMate
RECIPES
GROCERY LIST
STORE
TEST USER
Grocery Planner
check which recipes you want to buy ingredients for
combines ingredients if they are the same (bread from avocado toast and grilled cheese)

Your Recipes

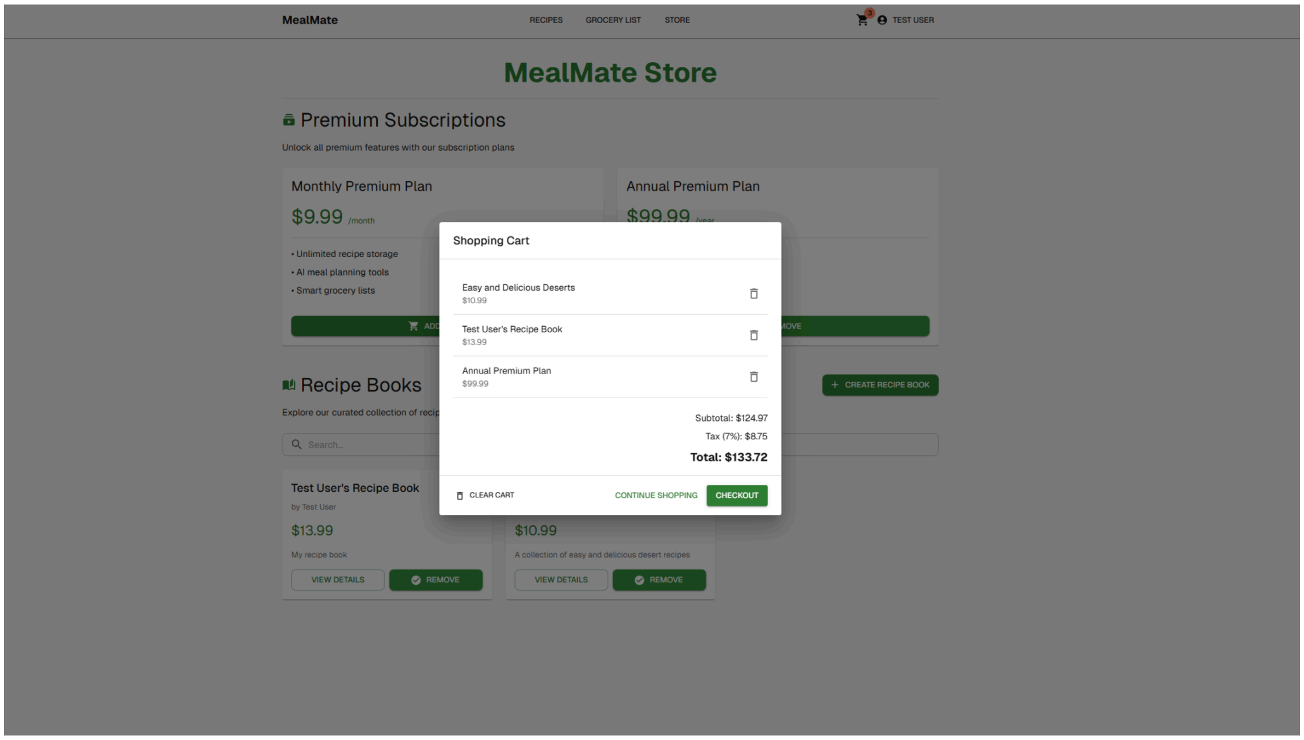
- [✓] Grilled Cheese — A simple grilled cheese sandwich.
- [✓] Avocado Toast — Simple and nutritious breakfast
- [ ] Chicken Caesar Salad — Classic salad with grilled chicken and Caesar dres...
- [✓] Apple Pie — Apple Pie
- [ ] Straw Barry Ice Cream — Straw Barry Ice Cream
- [ ] Pizza — cheese pizza

Grocery List

- apples (4 medium)
- Avocados (2 Medium)
- bread (2 Slices + 2 slices)
- butter (1/4 cup)
- cheddar cheese (4 slices)
- pie crust (1)

Store and Cart

Description: This page allows users to add recipe books or premium subscriptions to the user's cart. When the user goes through the card info and confirmation page, it will update their recipes with the new ones fro the recipe books.



Easy and Delicious Deserts
$10.99

Test User's Recipe Book
$13.99

Annual Premium Plan
$99.99

Subtotal: $124.97
Tax (7%): $8.75
Total: $133.72

6. Installation and Setup Instructions

1. clone Repo
2. cd into frontend directory
   a. run "**npm i**" in the terminal
3. cd into the backend directory
   a. run "**npm i**" in the terminal
4. create a .env file in the root of the backend directory
   a. copy paste the code below into the directory:

```
MONGODB_URI=mongodb+srv://achen2304:TzJp93B1SEpD4hTQ@cluster0.ztrxjtm.mongodb.
net/?retryWrites=true&w=majority&appName=Cluster0
PORT=8080
NODE_ENV=development
```

5. cd/stay in the backend directory
   a. run "**npm run dev:all**" in the terminal
6. go to localhost: http://localhost:3000/ to see the local environment
7. http://localhost:8080/ is the server

7. Contribution Overview

Feature	Contribution
Landing	Cai Chen
User Auth	Cai Chen
Recipes	Cai Chen
Grocery	Megan Chng
MarketPlace	Megan Chng
404 Page	Megan Chng
Redirect	Megan Chng

8. Challenges Faced

Time was a major issue with this project. Both of us were super busy with other projects and classes, which left us little time to implement everything we wanted. We had to drop alot of our initial features due to the time constraints.

Another issue was migrating the frontend to connect with the backend instead of placeholder data. When making the backend, we simplified it so it didn't have as much complicated and unnecessary connections, especially for a noSQL database. Our frontend was still using the complicated connections so we had trouble moving everything to a different data format.

9. Final Reflections

We learned alot from the project, main about web development and some basic backend functionalities. We learned about typescript, react, next.js, material UI, node.js, express, and mongoDB along with UI development, working on a team, and GitLab (and it's issues). Some things we could have done better was be more realistic about what we can implement. We didn't have much time for this project because of other classes so a lot of things went unimplemented.