# System Design Document

Project Name: RecipeConnect
Team: DevourDevs
Sprint 4

## 1. Introduction

This document outlines the system design for the implementation of RecipeConnect app. The system is developed using the MERN (MongoDB, Express.js, React, Node.js) stack and follows MVC architecture.

## 2. System Overview

The software will consist of the following key features:

- **User Authentication**: Users can register and login so that they can create and manage their recipes.
- **Recipe Browsing**: All users (both logged-in and non-logged-in) can view all recipes on the landing/home page. Users can view details about recipes, including ingredients, instructions, and related information.
- **Filtering and Searching:** Users can filter recipes based on dietary information, dish types, and cooking time. Users can also do a general search for recipes.
- **Recipe Rating and Review:** Logged-in users can rate meals or leave comments based on their experience.
- **Add Recipes to Favourite List:** Logged-in users can save a recipe to their favourite list
- **Ingredient-Based Search**: Logged-in users can input ingredients to get recipe suggestions.
- **Meal Planner:** Users can create, view, and manage their weekly meal plans. This feature will allow users to organize their meals for the week, helping them plan for their dietary needs and cooking schedule.
- **Nutritional Intake Analysis:** Users can upload a picture of their meal, and the app will analyze the nutritional content of the meal, including calories, proteins, fats, and other nutrients. The app will record this intake and use the information to track the user's daily intake.
- **AI Assistant Chat:** The app will feature an AI assistant through a chat interface. This assistant will provide step-by-step cooking guidance, help users learn new recipes, and answer cooking-related questions.
- **Voice Interaction with AI Assistant:** When logged in, users have an option to use their voice to interact with the AI Assistant instead of manually typing the messages. The system will process users' voice input and response with audio. When users speak to the assistant, the messages and responses will appear in chat history, allowing a continuous flow of conversations.

## 3. CRC cards:

### 3.1 UserModel Class (models/userModel.js)

| | |
|---|---|
| Class Name: **UserModel** | |
| Parent Class: None<br>Subclass: None | |
| Responsibilities:<br>● Define the schema for user data (username, password, email)<br>● Ensure email and username are unique<br>● Check for existing user with given username<br>● Hash password using bcrypt before saving users to database<br>● Validate user credentials during login (i.e. compare hashed password)<br>● Save user data to the database | Collaborators:<br>● AuthController - to handle user registration and login logic |

### 3.2 AuthController Class (controllers/authController.js)

| | |
|---|---|
| Class Name: **AuthController** | |
| Parent Class: None<br>Subclass: None | |
| Responsibilities:<br>● register: handle user registration, validate input, check if user with username and email exists<br>● login: validate credentials, generate JWT, send response (including JWT) to client, redirect to home page (with login status)<br>● me: checking the user's login status and provides information of the logged-in user (userId, username) | Collaborators:<br>● UserModel - to create a new user instance, provides methods to check if user exists, and validate user credentials<br>● AuthMiddleware - to handle authentication checks for protected routes. |

**3.3 AuthMiddleware Class (middlewares/authMiddleware.js)**

| Class Name: **AuthMiddleware** | |
|---|---|
| Parent Class: None<br>Subclass: None | |
| Responsibilities:<br>● Protect routes that require user authentication<br>● Verify the user's JWT token to ensure they are logged in<br>● Redirect users to login page if they are not logged in when performing actions requiring authorization | Collaborators:<br>● authController - manage the authentication process<br>● mealController - to ensure that actions like creating/editing recipes are only accessible by logged-in users. |

**3.4 MealModel Class (models/mealModel.js)**

| Class Name: **MealModel** | |
|---|---|
| Parent Class: None<br>Subclass: None | |
| Responsibilities:<br>● Define the schema for recipes | Collaborators:<br>● mealController - to handle requests for retrieving recipes and other recipe-related operations<br>● authMiddleware - to verify user's identity before allowing protected recipe features such as adding to favourites, recipe rating |

**3.5 MealController Class (controllers/mealController.js)**

| Class Name: **MealController** | |
|---|---|
| Parent Class: None<br>Subclass: None | |
| Responsibilities:<br>● Fetch recipe data from Spoonacular api | Collaborators:<br>● mealModel - interact with the database for operations on recipes |

| | |
|---|---|
| ● Return recipe data to client<br>● Handle recipe-related operations such as favourites, rating, filtering and search | ● authMiddleware - ensure only logged-in users can perform actions like rating or adding a recipe to favourite list<br>● favouriteModel - to manage favourite meals<br>● ratingModel - to manage meal ratings |

### 3.6 FavouriteModel Class (models/favouriteModel.js)

| Class Name: **FavouriteModel** | |
|---|---|
| Parent Class: None<br>Subclass: None | |
| Responsibilities:<br>● Define schema for user favourites<br>● Store favourite recipes for each user | Collaborators:<br>● mealModel - to manage adding and retrieving favourite recipes<br>● userModel - to associate users with their favourite recipes.<br>● authMiddleware - ensure only logged-in users can add recipes to favourite list |

### 3.7 RatingModel Class (models/ratingModel.js)

| Class Name: **RatingModel** | |
|---|---|
| Parent Class: None<br>Subclass: None | |
| Responsibilities:<br>● Define schema for recipe ratings<br>● Store total rating points and users who rated the recipe | Collaborators:<br>● mealController - to manage rating operations<br>● userModel - to associate users with their ratings.<br>● authMiddleware - ensure only logged-in users can rate a recipe |

**3.8 ImageController Class (controllers/imageControllers.js)**

| Class Name: **ImageController** | |
|---|---|
| Parent Class: None<br>Subclass: None | |
| Responsibilities:<br>● Handle image upload<br>● Process image through FoodNet model to recognize ingredients and nutritional content<br>● Use Roboflow API to detect ingredients in the uploaded image<br>● Delete uploaded image from server after processing | Collaborators:<br>● authMiddleware - ensure only logged-in users can access the feature |

**3.9 IntakeModel Class (models/intakeModel.js)**

| Class Name: **IntakeModel** | |
|---|---|
| Parent Class: None<br>Subclass: None | |
| Responsibilities:<br>● Define schema for user meal intake<br>● Store daily intake records for each user | Collaborators:<br>● userModel - to associate intake data with users<br>● authMiddleware - ensure only logged-in users can access the feature |

**3.10 IntakeController Class (controllers/intakeControllers.js)**

| Class Name: **IntakeController** | |
|---|---|
| Parent Class: None<br>Subclass: None | |
| Responsibilities:<br>● Save the nutritional intake data for a user<br>● Retrieve user’s intake history | Collaborators:<br>● intakeModel - to interact with intake data in database<br>● authMiddleware - ensure only logged-in users can access the feature |

### 3.11 PlannerModel Class (models/plannerModel.js)

| Class Name: **PlannerModel** | |
|---|---|
| Parent Class: None<br>Subclass: None | |
| Responsibilities:<br>● Define the schema for meal planners, associating each user with a meal plan for a specific date<br>● Store meal plan details, including breakfast, lunch, and dinner, by referencing meal IDs | Collaborators:<br>● userModel - to associate each meal plan with users<br>● mealModel - to reference meals for a plan<br>● authMiddleware - ensure only logged-in users can access the feature |

### 3.12 PlannerController Class (controllers/plannerControllers.js)

| Class Name: **PlannerController** | |
|---|---|
| Parent Class: None<br>Subclass: None | |
| Responsibilities:<br>● Create a new meal plan for the user with specified meals for breakfast, lunch, and dinner<br>● Retrieve the meal plan for a specific user and date<br>● Retrieve meals for the meal plan, providing details for breakfast, lunch, and dinner<br>● Update the meal plan for a user, modifying the meals for a specific date<br>● Delete a meal plan for a user | Collaborators:<br>● plannerModel - to interact with the meal plans stored in the database<br>● authMiddleware - ensure only logged-in users can access the feature |

### 3.13 AIAssistantController  Class (controllers/aiAssistantController.js)

| Class Name: **AIAssistantController** |
|---|

| Parent Class: None<br>Subclass: None | |
|---|---|
| Responsibilities:<br>● Generate step-by-step cooking instructions using AI based on provided recipe instructions.<br>● Handle real-time conversational interactions with the AI assistant, allowing users to ask cooking-related questions and receive responses | Collaborators:<br>● Google Gemini API - to generate responses for cooking instructions and chat interactions<br>● authMiddleware - ensure only logged-in users can access the feature |

**3.14 VoiceController  Class (controllers/voiceController.js)**

| Class Name: **VoiceController** | |
|---|---|
| Parent Class: None<br>Subclass: None | |
| Responsibilities:<br>● Handle voice input (an audio file)<br>● Transcribe the audio into text using Whisper AI<br>● Send transcribed test to AI Assistant and fetch a response<br>● Convert the AI Assistant's response into audio format<br>● Send the responses back to client via chat and audio | Collaborators:<br>● AIAssistantController - to process the transcribed text and generate a response<br>● authMiddleware - ensure only logged-in users can access the feature |

## 4. System Interaction with the Environment

### 4.1 Operating System:

- The system can be developed on Windows and macOS.
- Assumes the development environment supports Node.js for backend and React + Vite for frontend

### 4.2 Technology Stack:

- Frontend: Developed using React (JavaScript) and Vite as a build tool.
- Backend: Developed using Node.js and Express.js for handling API and client requests

- Database: Used local MongoDB for data storage. Relies on Spoonacular API for recipe data.
- External API: The app relies on:
  - Spoonacular API – Fetches recipes data. (https://spoonacular.com/food-api)
  - Google Gemini API – Powers the AI cooking assistant and chat agent. (https://aistudio.google.com)
  - Roboflow API – Detects ingredients from uploaded images for ingredient-based recipe suggestions. (https://universe.roboflow.com/recipevision/food-bxkvw)
  - FoodNet Model – Analyzes nutritional content of meals based on uploaded images for intake tracking.(https://github.com/Cheng-K/FoodNet)
  - Whisper AI – voice transcription feature (https://github.com/openai/whisper)
  - edge-tts – allows voice assistant to work on Microsoft Edge (https://github.com/rany2/edge-tts)

*Please see the updated installation step in README.md file for instructions to set up the database and run the app.*

## 5. System Architecture

### 5.1 Frontend (client-side):

- Handles UI, render pages for users to interact with
- Sends HTTP requests to the Backend (via axios or fetch) to retrieve data (recipes, user information)
- Displays content based on whether the user is logged in or not. For example:
  + Only logged-in users can use meal planner
  + All users (including non-logged-in) can see list of recipes on main pages

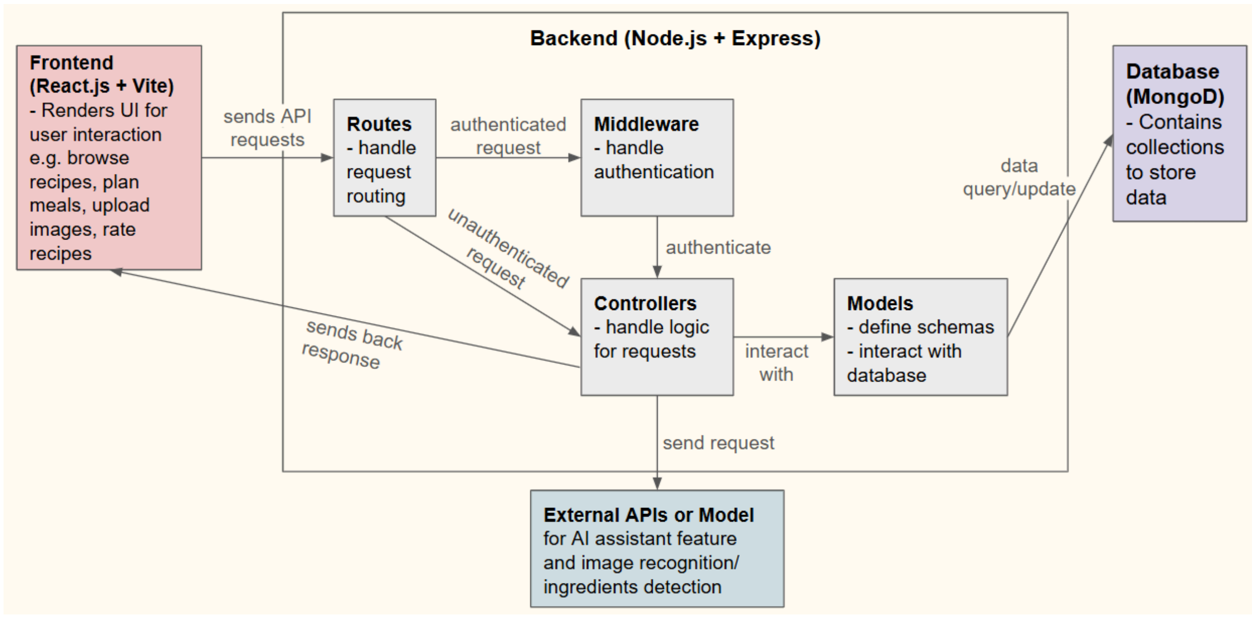
### 5.2 Backend (server-side):

- Manages logic for authentication, recipe operations, and serving API responses
- Authenticates users (with JWT and bcrypt) and verify requests for protected routes
- Interacts with the Database to retrieve and store user and recipe data.

### 5.3 Database:

- Stores user data and recipe content
- Provides data to the Backend via queries for things like retrieving recipes, user login logic.

### 5.4 Architecture Diagram:


Backend (Node.js + Express)
Frontend (React.js + Vite) - Renders UI for user interaction e.g. browse recipes, plan meals, upload images, rate recipes
sends API requests
Routes - handle request routing
authenticated request
Middleware - handle authentication
unauthenticated request
authenticate
Controllers - handle logic for requests
sends back response
interact with
Models - define schemas - interact with database
data query/update
Database (MongoD) - Contains collections to store data
send request
External APIs or Model for AI assistant feature and image recognition/ ingredients detection

## 6. System Decomposition:

**Frontend:**

- Pages/Components: storing pages and UI components used in the app.
- App.jsx handles pages routing
- Logout will be handled on client side by removing JWT from local storage.
- tests: contains front end test cases using vitest

**Backend:**

- controllers: Handles authentication, recipe retrieval, favourites, ratings, and other recipe-related operations.
- middleware:
  + Authentication Middleware: Verifies JWT token on protected routes (e.g., recipe rating).
- models: Defines structured data models for users and recipe-related features.
- tests: Contains the test cases for backend testing using Jest

**Database:**

- The database is **recipeconnect**

## 7. Error Handling and Exceptional Cases:

**Invalid user input during register:**

<u>Password length must be at least 8:</u> If this requirement is not met:

- Frontend will display an error message “Pass length must be at least 8” (or similar)

Confirmed password does not match provided password:
- Frontend will display an error message “Passwords do not match”

Username or email already taken:
- Backend will return 400 - Bad request
- Frontend will display an error message “Username/Email already existed”

**Authentication failure:**

Invalid JWT token: If the token is expired or invalid,
- Backend will return 401 - Unauthorized
- Frontend will redirect users to login page

Failed login: If credentials are incorrect during login,
- Backend will return 401 - Unauthorized
- Front end will display error message "Incorrect password or username"

**Database Errors:**

Connection Failures: If the backend cannot connect to database, backend will return 500 - Internal Server Error

Query Failure: If a query fails (e.g. no data found), backend will return 404 - Not found.

**Uncaught Errors:**

In the case of unexpected errors happening:
- Backend should provide a global error handler that will catch uncaught errors and return 500 - Internal Server Error
- Frontend should display appropriate error messages to users.

## 8. Backend controllers testing:

### 8.1 Running tests:

To run the backend controllers tests:

- Go to **server** folder
- Create **.env.test** file and put in the following:
```
PORT="3000"
MONGODB_URI="mongodb://127.0.0.1:27017/testrecipeconnect"
JWT_SECRET="ds9u2f383hf839"
NODE_ENV = "test"
SPOONACULAR_API_KEY="put your spoonacular api key here"
GEMINI_API_KEY = "put your gemini api key here"
ROBOFLOW_API_KEY = "put your roboflow api key here"
```
- You may have to create a new database **testrecipeconnect** in MongoDBCompass

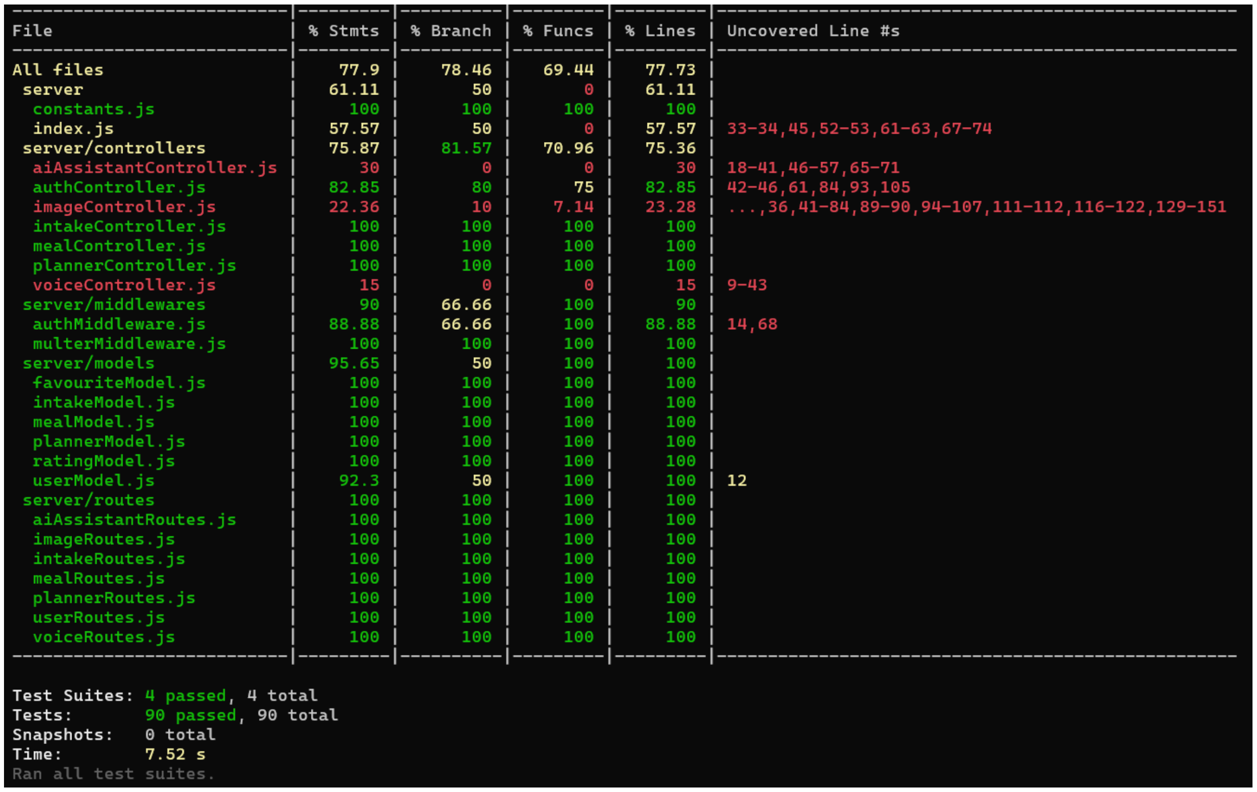
- To run the test: **npm test**
- To run the test with coverage information: **npm test -- --coverage**

### 8.2 Test Coverage

- Fully tested Controllers: authController, mealController, plannerController, intakeController
- Not fully tested controllers: imageController, aiAssistantController, voiceController (due to the complexity of the task and these controllers depend on calling external APIs to process; voice controller also depends on running an external Docker container. Therefore, we had to test their functionality manually)

### 8.3 Test results:

*More test cases may be added from the time of writing this document*



```
---------------------------|---------|----------|---------|---------|-------------------------------------------------
File                       | % Stmts | % Branch | % Funcs | % Lines | Uncovered Line #s
---------------------------|---------|----------|---------|---------|-------------------------------------------------
All files                  |    77.9 |    78.46 |   69.44 |   77.73 |
 server                    |   61.11 |       50 |       0 |   61.11 |
  constants.js             |     100 |      100 |     100 |     100 |
  index.js                 |   57.57 |       50 |       0 |   57.57 | 33-34,45,52-53,61-63,67-74
 server/controllers        |   75.87 |    81.57 |   70.96 |   75.36 |
  aiAssistantController.js |      30 |        0 |       0 |      30 | 18-41,46-57,65-71
  authController.js        |   82.85 |       80 |      75 |   82.85 | 42-46,61,84,93,105
  imageController.js       |   22.36 |       10 |    7.14 |   23.28 | ...,36,41-84,89-90,94-107,111-112,116-122,129-151
  intakeController.js      |     100 |      100 |     100 |     100 |
  mealController.js        |     100 |      100 |     100 |     100 |
  plannerController.js     |     100 |      100 |     100 |     100 |
  voiceController.js       |      15 |        0 |       0 |      15 | 9-43
 server/middlewares        |      90 |    66.66 |     100 |      90 |
  authMiddleware.js        |   88.88 |    66.66 |     100 |   88.88 | 14,68
  multerMiddleware.js      |     100 |      100 |     100 |     100 |
 server/models             |   95.65 |       50 |     100 |     100 |
  favouriteModel.js        |     100 |      100 |     100 |     100 |
  intakeModel.js           |     100 |      100 |     100 |     100 |
  mealModel.js             |     100 |      100 |     100 |     100 |
  plannerModel.js          |     100 |      100 |     100 |     100 |
  ratingModel.js           |     100 |      100 |     100 |     100 |
  userModel.js             |    92.3 |       50 |     100 |     100 | 12
 server/routes             |     100 |      100 |     100 |     100 |
  aiAssistantRoutes.js     |     100 |      100 |     100 |     100 |
  imageRoutes.js           |     100 |      100 |     100 |     100 |
  intakeRoutes.js          |     100 |      100 |     100 |     100 |
  mealRoutes.js            |     100 |      100 |     100 |     100 |
  plannerRoutes.js         |     100 |      100 |     100 |     100 |
  userRoutes.js            |     100 |      100 |     100 |     100 |
  voiceRoutes.js           |     100 |      100 |     100 |     100 |
---------------------------|---------|----------|---------|---------|-------------------------------------------------

Test Suites: 4 passed, 4 total
Tests:       90 passed, 90 total
Snapshots:   0 total
Time:        7.52 s
Ran all test suites.
```

## 9. Integration testing:

### 9.1 Running tests:

- Replicate the same test setup as backend controller testing
- cd to server and run ‘npm run start-cy’
- cd to root and run ‘npm run test’

### 9.2 Test Coverage

- Essential functionalities of user authentication journey, favourites, and more added after the writing of this document (test names will be self-explanatory)