System Design Document

Project Name: RecipeConnect
Team: DevourDevs
Sprint 1

1. Introduction

This document outlines the system design for the RecipeConnect app in Sprint 1.
The system is developed using the MERN (MongoDB, Express.js, React, Node.js) stack and follows the MVC architecture.

2. System Overview

The software in sprint 1 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.
- **Recipe Post Creation & Interactions**: Logged-in users can create/edit/delete recipes.

3. CRC cards:

3.1 UserModel Class (models/userModel.js)

<table>
<tr><td colspan="2">Class Name: UserModel</td></tr>
<tr><td colspan="2">Parent Class: None
Subclass: None</td></tr>
<tr><td>Responsibilities:
Define the schema for user data (username, password, email)
Ensure email and username are unique
Check for existing user with given email, username
Hash password using bcrypt before saving users to database
Validate user credentials during login (i.e. compare hashed password)
Save user data to the database</td><td>Collaborators:
AuthController - to handle user high-level registration and login logic</td></tr>
</table>

3.2 AuthController Class (controllers/authController.js)

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

3.3 AuthMiddleware Class (middlewares/authMiddleware.js)

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

3.4 RecipeModel Class (models/recipeModel.js)

Class Name: **RecipeModel**	
Parent Class: None Subclass: None	
Responsibilities: ● Define the schema for recipes ● Save recipes to the database ● Handle direct interaction related to recipes with the database	Collaborators: ● recipeController - to handle high-level CRUD operations related to recipes ● authMiddleware - to verify user's identity before allowing recipe creation/editing/deletion

3.5 recipeController Class (controllers/recipeController.js)

Class Name: **RecipeController**	
Parent Class: None Subclass: None	
Responsibilities: ● Handle high-level recipe creating/editing/deletion/retrieving ● Ensure that only logged-in users can perform CRUD on their own recipes ● Fetch and return recipes to client for display	Collaborators: ● recipeModel - interact with the database for CRUD operations on recipes ● authMiddleware - ensure only logged-in users can perform actions like creating or editing recipes. ● userModel - associate recipes with the user who created/owned them.

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.

Please see the 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 manage their recipes
  + All users (including non-logged-in) can see a list of recipes on the home page.

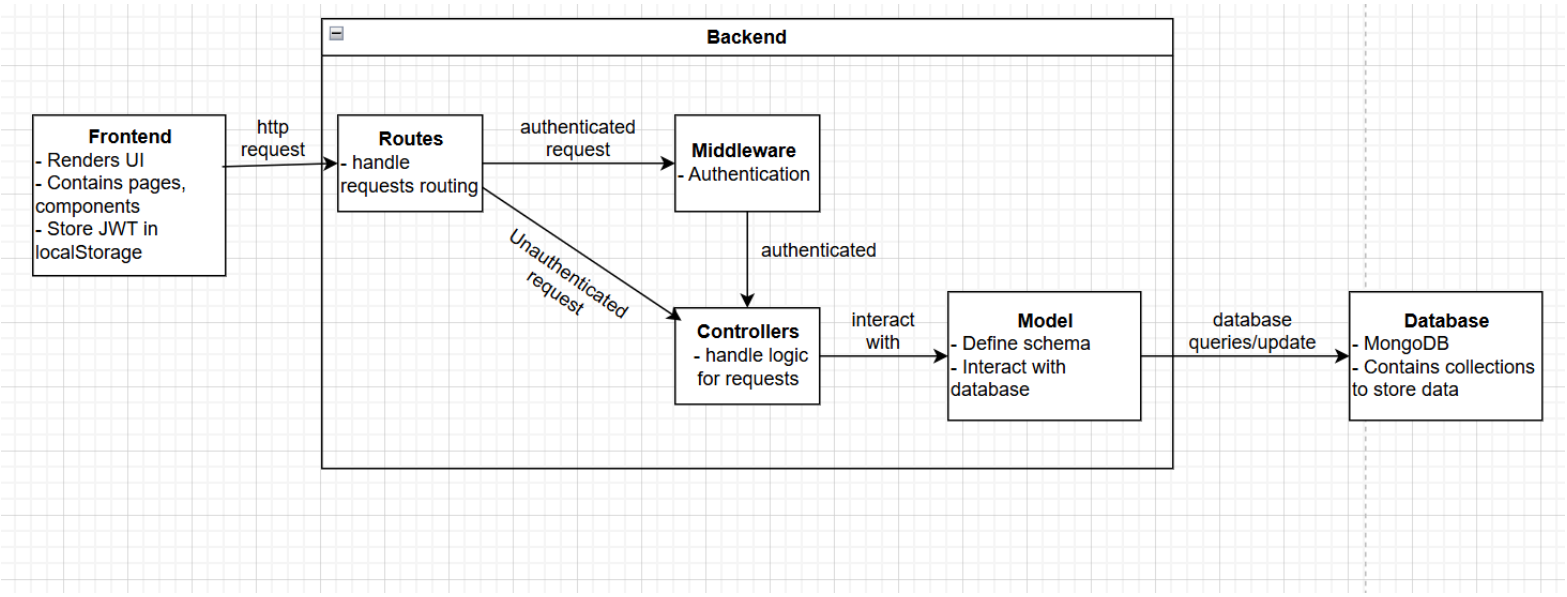
5.2 Backend (server-side):

- Manages logic for authentication, recipe CRUD operation, 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 information.

5.4 Architecture Diagram:



6. System Decomposition:

Frontend:

- Pages/Components:
  + Home (or Landing) Page: Displays a list of recipes for both logged-in and guest users.
  + Login Page: After a successful login, store JWT in browser's local storage.
  + Signup Page: Register new users
  + Recipe Management Page: Allows logged-in users to create, update, and delete their recipes.
- App.jsx handles pages routing and checking login status
- Logout will be handled on client side by removing JWT from local storage.

Backend:

- Controllers:
  + Auth Controller: Handles user registration, login, and JWT token generation.
  + Recipe Controller: Handles CRUD operations for recipes.
- Middleware:
  + Authentication Middleware: Verifies JWT token on protected routes (e.g., recipe creation or editing).
- Models:
  + User Model:
    - Define schema for users and handle interaction between the model and database.
  + Recipe Model:

- Define schema for recipes and handle interaction between the model and database.

Database:
- The database is "recipeconnect"
- Collections within the database:
  + Users: Stores hashed passwords, emails, and other profile data.
  + Recipes: Stores recipe names, ingredients, instructions, and the author, etc…

7. Error Handling and Exceptional Cases:
Invalid user input during register:
Password length must be at least 8: 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.
- Frontend should display appropriate messages

Query Failure: If a query fails (e.g. no data found),
- Backend will return 404 - Not found
- Frontend should display appropriate messages or redirect users to a ‘Page 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.