

DELIZIA

A Project Report

AJMI M(IRP003-HQ24AUG13)

AKHIL V(IRP003-HQ24AUG31)

DIVYA SREEKALA(IRP003-HQ24AUG30)

GOWRI N(IRP003-HQ24AUG42)

PRECIOUS P P(IRP003-HQ24AUG28)

Abstract

Delizia is a recipe-searching web application built using the MERN stack, offering users the ability to search for recipes by name or by selecting multiple ingredients from a multi-select dropdown. The frontend, developed with React.js, provides an intuitive and responsive interface, triggering API calls to the backend, built with Node.js and Express.js, which processes these search queries. Recipes are fetched from both a MongoDB database and external APIs, and displayed with details such as ingredients, cooking steps, preparation time, and images. Features include ingredient-based search, a recipe details page, the ability to save/favorite recipes, and suggestions/recommendations based on user preferences. Additionally, users can copy cooking steps and descriptions to their clipboard and use a text-to-audio feature to read out cooking instructions. Future enhancements include search by image and search by recipe using a camera, expanding the search capabilities even further.

Acknowledgement

We take this opportunity to express our deepest sense of gratitude and sincere thanks to everyone who helped us to complete this work successfully. We express our sincere thanks to faculties of ICT Academy of Kerala Thiruvananthapuram for providing us this Internship program with all the necessary facilities and support. We would like to express my sincere gratitude to Miss. Remya U.L., Senior Knowledge Officer FULL STACK DEVELOPMENT MERN and Miss. Mridula Reghunath, Senior Knowledge Officer, FULL STACK DEVELOPMENT MERN, ICT Academy of Kerala Thiruvananthapuram for the support and co-operation. We would like to place on record our sincere gratitude to the TRACKGENESIS team for providing the Internship and guiding us throughout the project. Finally We thank our family, and friends who contributed to the successful fulfilment of this Internship project.

AJMI M

AKHIL V

DIVYA SREEKALA

GOWRI N

PRECIOUS P P

Contents

Abstract	i
Acknowledgement	ii
List of Figures	v
1 Introduction	1
2 Requirements	3
2.1 Hardware Requirements	3
2.2 Software Requirments	3
2.2.1 Frontend:	3
2.2.2 Backend:	4
2.3 Functional Requirements	4
2.4 Non-Functional Requirements	4
3 Design	7
3.1 Project Planning	8
3.2 Flow Diagram	9
3.3 Folder Structure	9
3.4 Development Environment	10
4 Frontend Implementation	14
4.1 Technology Stack	14
4.2 Features	14
4.3 App Functionality	15
4.3.1 User Functionality	15

4.3.2	Admin Functionality	16
4.4	Components	16
5	Backend Implementation	17
5.1	Technology Stack	17
5.2	Features	17
5.3	Backend API	19
5.3.1	Authentication and User Management:	19
5.3.2	Recipe Management	19
5.3.3	Search and Filtering	20
5.3.4	Wishlist Management	20
5.3.5	External API Management	21
6	Database Implementation	22
7	Deployment	24
8	Results	25
9	Discussion	29
10	Conclusion	31
10.1	Future Enhancement	31
11	References	33

List of Figures

3.1	DFD	9
3.2	Frontend folder	9
3.3	Backend folder	10
6.1	userdata model	22
6.2	recipedata model	23
6.3	wishlistdata model	23
8.1	Home Page	25
8.2	Footer Page	26
8.3	Login	26
8.4	Signup	26
8.5	Search page	27
8.6	Recipe Page	27
8.7	Wishlist Page	27
8.8	About us Page	28
8.9	Admin Dashboard	28

Chapter 1

Introduction

In today's fast-paced and convenience-driven world, food plays a central role in our daily lives, serving as a source of nourishment, creativity, and connection. However, the challenge of discovering new recipes that align with our available ingredients, preferences, and time constraints can often feel overwhelming. With busy schedules and a growing desire for efficient meal planning, there is a need for a smart, user-friendly tool that simplifies the cooking experience.

Introducing the Delizia App – an innovative platform designed to revolutionize the way users explore, plan, and prepare meals. Built using the powerful MERN stack, the Recipe App seamlessly combines technology and culinary inspiration to offer a transformative cooking experience tailored to every individual's needs. The platform empowers users to search for recipes effortlessly, whether by entering a recipe name or selecting specific ingredients they already have at home. By bridging the gap between pantry staples and delicious meals, the Recipe App enhances creativity in the kitchen and minimizes food waste, all while making cooking more accessible and enjoyable.

With the Delizia App, users can embark on a culinary journey where meal preparation is both efficient and exciting. The app leverages an extensive database of recipes, enabling users to explore diverse dishes across various cuisines, categories, and dietary preferences. Whether you're searching for a quick dinner idea, a healthy meal, or a way to use up leftover ingredients, the Recipe App provides tailored results to match your input. Its intuitive user interface, powered by React.js, ensures a seamless experience, while the robust backend, built with Node.js and MongoDB,

guarantees accurate, real-time search results.

More than just a recipe search tool, the Deliziq App fosters a deeper connection between individuals and their culinary creativity. By offering smart search capabilities, and an immersive user experience, the Recipe App redefines how users approach cooking and meal planning. Join us on this exciting journey to make cooking a hassle-free, inspiring, and joyful part of your everyday life with the Recipe App – your ultimate companion in the kitchen.

The Delizia App is designed to be a user-friendly and scalable platform for both Admins and Users. It consists of two modules: Admin and User, offering features like recipe management, and ingredient-based search. The front end is built with React for a responsive interface, while the back end uses Node.js, Express, and MongoDB for efficient data handling. APIs enable seamless front-end and back-end communication, ensuring real-time interaction. The modular design and clear folder structure ensure maintainability and future scalability.

Chapter 2

Requirements

2.1 Hardware Requirements

- **Processor:** Minimum Intel Core i3 or equivalent (Recommended: i5 or higher).
- **RAM:** Minimum 4 GB (Recommended: 8 GB or higher for smoother performance).
- **Storage:** At least 50 GB free disk space for codebase, databases, and dependencies.
- **Monitor:** 1080p resolution for better UI/UX testing.
- **Network:** Stable internet connection for dependency installations, API testing, and deployment.

2.2 Software Requirements

2.2.1 Frontend:

- **React.js:** For building the user interface.

Libraries/Dependencies: React Router, Axios, Material-UI/Bootstrap (optional for styling).

- **Node.js (LTS):** Required to run React development server.

- **Browser:** Google Chrome, Firefox, or Edge for testing UI.

2.2.2 Backend:

- **Node.js (LTS):** To run the backend server.
- **Express.js:** For handling server-side logic and API endpoints.
- **Mongoose:** For managing MongoDB interactions.

2.3 Functional Requirements

- **User Profile Management:**
 - Include features for users to save their favorite recipes and preferences.
- **Ingredient-Based Recipe Search:**
 - Enable users to input available ingredients.
 - Provide a list of all recipes that can be made using the given ingredients.
- **Recipe Details:**
 - Display detailed steps for each recipe.
 - Include information about preparation time, cooking time, and nutritional value.
 - Allow users to upload their own recipes with images and descriptions.

2.4 Non-Functional Requirements

- **Performance:**
 - **Response Time:** The app should respond to user inputs, such as searching for recipes or filtering by ingredients, within 1-2 seconds to ensure a smooth user experience.

- **Scalability:** The system should handle an increasing number of users and recipe uploads without performance degradation.
- **Reliability:** The app should maintain high reliability, minimizing crashes and ensuring that recipes and user data are always accessible.
- **Security:**
 - **Data Encryption:** Encrypt all user data, such as account details and saved recipes, to protect against unauthorized access.
 - **User Authentication:** Implement secure login mechanisms to protect user accounts.
 - **Authorization Controls:** Ensure that only authorized users can upload or edit recipes.
- **Compatibility:**
 - **Cross-Platform Compatibility:** Ensure the app functions seamlessly across multiple platforms (e.g., iOS, Android) and devices (phones, tablets).
 - **Browser Compatibility:** If a web version is available, ensure it is compatible with popular browsers like Chrome, Firefox, and Safari.
- **User Interface (UI) and User Experience (UX):**
 - **Intuitive Design:** The app should have a clean, user-friendly interface that makes it easy to search, filter, and upload recipes.
 - **Accessibility:** Ensure the app is accessible to users with disabilities by following standards like WCAG.
 - **Consistency:** Maintain consistent UI elements and styling across all sections of the app.
- **Availability:**
 - **Uptime:** Aim for high availability with minimal downtime, ensuring that users can access recipes at any time.
- **Regulatory Compliance:**

- **Data Privacy:** Comply with relevant data protection regulations to safeguard user information.

Chapter 3

Design

The Delizia App is designed to be a user-friendly, scalable platform catering to both Admins and Users, focused on providing a seamless recipe discovery and management experience. The app features two primary modules: the Admin module, which allows for recipe management (add, edit, delete, list), and the User module, which offers an ingredient-based recipe search, and the ability to save/favorite recipes. The front-end of the app is built using React, ensuring a dynamic and responsive interface, while the back-end leverages Node.js, Express, and MongoDB for efficient data handling, real-time interactions, and easy scalability. APIs facilitate smooth communication between the front-end and back-end, enabling real-time data retrieval, search functionalities, and text-to-audio integration for cooking steps. The app follows a modular structure with a well-organized folder system to ensure maintainability and scalability, allowing easy updates and future feature additions. This architecture supports future growth, such as integrating user-generated content, social sharing, advanced nutritional information, and multi-language support. The design prioritizes flexibility, ensuring the platform can evolve alongside user needs and the growing recipe database.

3.1 Project Planning

Below is the tentative timeline and milestone breakdown for this project:

Week	Tasks	Timeline	Milestone
1 - Project Planning and Front-End Development			
Week 1	<ul style="list-style-type: none"> - Define project scope and features. - Set up Git repository. - Create wireframes/mockups. - Initialize React project and organize folder structure. - Configure React Router. - Build UI components (Navbar, Footer, Forms). - Create and style pages (Sign Up, Login, Homepage). 	Nov 20 - Nov 26	Project plan ready, structured front-end setup with partially styled UI components.
2 - Back-End Development and Database Setup			
Week 2	<ul style="list-style-type: none"> - Initialize Node.js and Express. - Design API routes (CRUD operations). - Set up MongoDB and Mongoose schemas. - Implement API endpoints. - Test APIs with Postman. - Handle errors and enable CORS. 	Nov 27 - Dec 3	Functional back-end with a connected database and tested API endpoints.
3 - Full-Stack Integration and Core Functionalities			
Week 3	<ul style="list-style-type: none"> - Connect front end to back end using Axios/Fetch. - Implement login, signup, and data fetching. - Set up JWT authentication and secure private routes. - Manage state with Context API or Redux. - Add loaders, error handlers, and conditional rendering. 	Dec 4 - Dec 10	Fully integrated application with core functionalities like user authentication and data interaction working end-to-end.
4 - Advanced Features, Testing, Deployment, and Documentation			
Week 4	<ul style="list-style-type: none"> - Add search, sorting, or pagination. - Implement user roles and optimize performance. - Write tests for critical functions. - Deploy back end (Heroku/Render) and front end (Netlify/Vercel). - Document API and project structure. - Prepare demo and final presentation. 	Dec 4 - Dec 10	Fully integrated application with core functionalities like user authentication and data interaction working end-to-end.

3.2 Flow Diagram

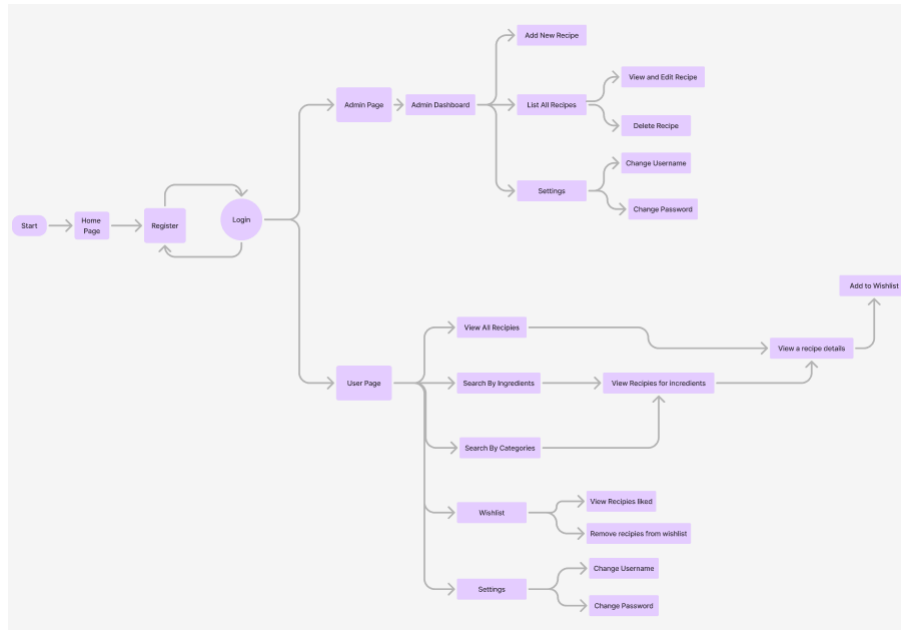


Figure 3.1: DFD

3.3 Folder Structure

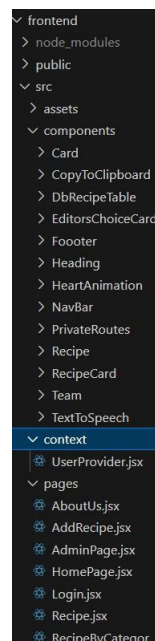


Figure 3.2: Frontend folder

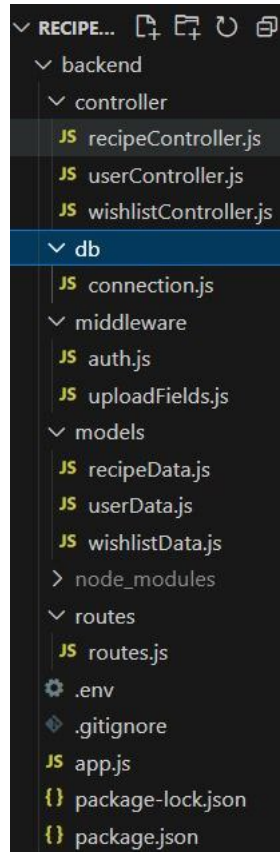


Figure 3.3: Backend folder

3.4 Development Environment

The development environment for the Job Portal project consists of the following key components to ensure a smooth and efficient workflow:

1. Code Editor/IDE:

- Visual Studio Code (VS Code): A highly popular, lightweight code editor with support for various programming languages, version control integration, and extensions for React, Node.js, MongoDB, and other technologies used in the project. VS Code also offers features like IntelliSense, debugging tools, and an integrated terminal that helps streamline the development process.

2. Version Control:

- Git: Git is used for version control, allowing the development team to track changes, collaborate effectively, and maintain a history of all modifications made to the codebase.

3. Frontend:

- **React.js:** A JavaScript library for building user interfaces, particularly single-page applications (SPA). React's component-based architecture makes it easy to manage the UI and state across different sections of the job portal.
- **Material-UI:** A React component library that provides a rich set of pre-styled UI components, such as buttons, forms, grids, and tables. Material-UI ensures a consistent and responsive design across devices.
- **Axios:** A promise-based HTTP client used for making API requests between the frontend and backend.
- **React Router:** A library used for managing navigation between pages in the React application.

4. Backend:

- **Node.js:** A JavaScript runtime built on Chrome's V8 JavaScript engine. Node.js is used for building the server-side logic of the Job Portal, handling HTTP requests, user authentication, and data processing.
- **Express.js:** A minimalist web framework for Node.js that simplifies routing and middleware management. Express.js handles requests from the frontend and responds with appropriate data from the MongoDB database.
- **MongoDB:** A NoSQL database used for storing user data, job listings, applications, and other dynamic content. MongoDB allows for flexibility in handling unstructured data and is well-suited for applications that require quick scaling.
- **Mongoose:** An ODM (Object Document Mapper) for MongoDB that provides a schema-based solution to model the application data, making it easier to work with MongoDB collections in a more structured way.

5. Authentication:

- **JWT (JSON Web Tokens):** A token-based authentication mechanism used for securing user sessions. JWT is used for ensuring that both recruiters and job seekers are authenticated before they can access certain parts of the application.

- Bcrypt.js: A library used for hashing passwords before they are stored in the database, ensuring secure user authentication.

6. Development Tools:

- Postman: A popular tool used for testing APIs during development. Postman allows developers to send requests to the backend API endpoints, view responses, and check for correct data formatting and behavior.
- Nodemon: A utility that automatically restarts the Node.js server whenever changes are made to the backend code, helping speed up the development process by eliminating the need for manual server restarts.

7. Package Manager:

- npm (Node Package Manager): npm is used for managing dependencies and scripts in the project. It allows easy installation and management of libraries, frameworks, and tools needed for both the frontend and backend.

8. Database Management Tools:

- MongoDB Atlas: A cloud-based service for managing MongoDB databases. MongoDB Atlas provides a secure and scalable environment for hosting and managing the database in production.

9. Testing:

- Postman: supports automated testing by allowing developers to write test scripts in JavaScript that are executed after each API request. These scripts can validate the response status code, response body, and even response time, ensuring the backend behaves correctly under various conditions.

10. Deployment:

- Vercel: A cloud platform for deploying frontend applications. Vercel is used to deploy the React application with continuous integration, automatic deployments from Git repositories, and instant global scalability.

11. Development Workflow:

- Agile Methodology: The development process is typically carried out in iterative cycles (sprints) with regular feedback loops, ensuring that the features are continuously improved and refined based on user needs.
- CI/CD Pipelines: Continuous Integration and Continuous Deployment are implemented for the project to automate the testing and deployment process. Tools like GitHub Actions or CircleCI are used to automate the build, test, and deployment pipeline.

Chapter 4

Frontend Implementation

4.1 Technology Stack

- React.js: Frontend framework for building a dynamic user interface.
- Material-UI: For styling components and implementing responsive design.
- Redux: For state management.
- Axios: For handling HTTP requests.
- React Router: For managing routes and navigation.

4.2 Features

1. User Interface:

- A responsive and user-friendly design that adapts to various screen sizes.
- Intuitive navigation with clear labels and instructions.

2. Authentication:

- Signup and Login pages for users and adminss.
- Role-based redirection (e.g., users to their dashboard, admins to their profile).

3. Recipes Listings:

- Dynamic display of recipes with search and filter functionalities.
- Pagination for better user experience with a large number of recipes.

4.3 App Functionality

General

- **Registration (Admin and User):** Users and Admins can register with their respective roles.
- **Login:** Secure login for users and admins.

4.3.1 User Functionality

- **Ingredient-Based Recipe Search:** Users can search recipes using a combination of ingredients. The search fetches results from both external APIs and the app's database.

Recipe Details Page: Displays comprehensive recipe information, including:

- List of ingredients.
- Step-by-step cooking instructions.
- Preparation time and cooking time.

Save/Favorite Recipes: Users can save or mark recipes as favorites for easy access later.

- Recipes marked as favorites.

Description and Cooking Steps - Copy to Clipboard: Allows users to copy recipe descriptions or cooking steps with a single click.

Text-to-Audio for Cooking Steps: Converts cooking steps into audio instructions for hands-free cooking.

4.3.2 Admin Functionality

- **Recipe Data Management:**
 - **Add New Recipe:** Admins can create new recipe entries, specifying all required details.
 - **Edit Recipe:** Update existing recipe details.
 - **Delete Recipe:** Remove recipes from the database.
 - **List Recipe:** View all recipes in a manageable format with sorting/filtering options.

4.4 Components

1. Home Page:

- Highlights the latest recipes and provides quick access to key sections.

2. Login/Signup:

- User-friendly forms for both users and admins with validation and error handling.

3. Recipe Details Page:

- Displays detailed recipe information, including ingredients, cooking steps, preparation time.
- Includes options to copy cooking steps to the clipboard and text-to-audio functionality.

4. User Dashboard:

- Displays a list of saved/favorite recipes and allows users to search for recipes based on ingredients.

5. Admin Dashboard:

- Provides recipe management capabilities such as adding, editing, deleting, and listing recipes.

Chapter 5

Backend Implementation

5.1 Technology Stack

- Node.js: Backend runtime environment.
- Express.js: Framework for building RESTful APIs.
- MongoDB: NoSQL database for storing recipes, user data, and wishlists.
- Mongoose: ODM library for defining schemas and interacting with MongoDB.
- JWT (JSON Web Token): For secure authentication and authorization.
- Bcrypt.js: For password hashing and secure storage.

5.2 Features

User Authentication

- Secure login and signup for Admin and Users using JWT-based authentication.
- Password hashing with bcrypt to ensure secure storage of user credentials.

Recipe Management

- CRUD operations for recipes, allowing Admin to:

- Add new recipes.
 - Edit or delete existing recipes.
 - List all available recipes.
- Recipes are stored in MongoDB using a schema defined in `recipeData.js`.

Wishlist Management

- Users can save their favorite recipes to a wishlist.
- Wishlist data is handled using a dedicated schema

Search and Filter

- API for searching recipes based on ingredients or categories.
- Integration of external APIs for ingredient-based recipe searches.

Related Recipes Based on Recipe Category

- Related recipes are determined by categorizing recipes into various groups such as cuisine type, meal type, or dietary restrictions.
- The related recipes of a given recipe are found by searching for other recipes that match the same category, providing users with suggestions of similar dishes.

Text-to-Speech

- Text-to-speech is achieved by using the browser's `SpeechSynthesis` API, which converts the text content of a target element into speech, while allowing control over play, pause, resume, speed, and highlighting the currently spoken word.

Clipboard Functionality

- The clipboard API (`navigator.clipboard.writeText`) is used to copy text to the clipboard.

5.3 Backend API

5.3.1 Authentication and User Management:

- `router.post('/register', userRegistration)`
 - Purpose: Registers a new user.
 - Details: Accepts user details (e.g., name, email, password) and creates a new account. Stores the user data in the database.
- `router.post('/login', login)`
 - Purpose: Logs a user into the app.
 - Details: Verifies user credentials, generates a JWT token for authentication, and returns it to the user.
- `router.get('/view-profile', authMiddleware('user'), viewProfile)`
 - Purpose: Retrieves the logged-in user's profile details.
 - Details: Requires authentication using the `authMiddleware`. Returns the user's stored information.

5.3.2 Recipe Management

- `router.post('/add',authMiddleware('admin'), addRecipe)`
 - Purpose: Adds a new recipe to the database.
 - Details: Requires authentication.Accepts recipe data (e.g., name, ingredients, steps) from the admin and stores it in the database.
- `router.get('/list-recipes', listRecipes)`
 - Purpose: Lists all recipes in the database and random recipes from the spoonacular api.
 - Details: Returns a collection of recipes from both api.
- `router.get('/dbRecipe', authMiddleware('admin'), dbRecipe)`

- Purpose: Lists all recipes in the database and random recipes from the spoonacular api.
- Details: Returns a collection of recipes from both api.
- router.get('/recipe-details/:id', getRecipeById)
 - Purpose: Fetches details of a specific recipe by its ID.
 - Details: Returns complete details, including ingredients, instructions, and images for recipe in database or in spoonacular.
- router.put('/edit-recipe/:id', authMiddleware('admin'), updateRecipe)
 - Purpose: Edits an existing recipe.
 - Details: Requires authentication. Admin updates the recipe's details using its ID.
- router.delete('/delete/:id', authMiddleware('admin'), deleteRecipe)
 - Purpose: Deletes a specific recipe by its ID.
 - Details: Requires authentication. Admin removes the recipe from the database.

5.3.3 Search and Filtering

- router.get('/search-recipes', getRecipeByIngredients)
 - Purpose: Finds recipes based on given ingredients.
 - Details: Accepts a list of ingredients as a query parameter and returns recipes that can be made using those ingredients from the database and spoonacular.
- router.get('/recipe-by-category', getRecipeByCategory)
 - Purpose: Fetches recipes by category (e.g., drink, dessert).
- router.get('/popular-recipes', getRecipesPopular)
 - Purpose: Retrieves a list of popular recipes tagged by the admin.
- router.get('/recipe-by-name', getRecipeByName)
 - Purpose: Searches for a recipe by its name in the database.

5.3.4 Wishlist Management

- router.post('/add-wishlist', authMiddleware('user'), addWishlistRecipe)

- Purpose: Adds a recipe to the user’s wishlist.
- Details: Requires authentication. Stores the recipe ID in the user’s wishlist.
- `router.get('/wishlist-recipes', authMiddleware('user'), listWishlistRecipe)`
 - Purpose: Lists all recipes in the user’s wishlist.
 - Details: Requires authentication. Fetches all recipes added to the wishlist.

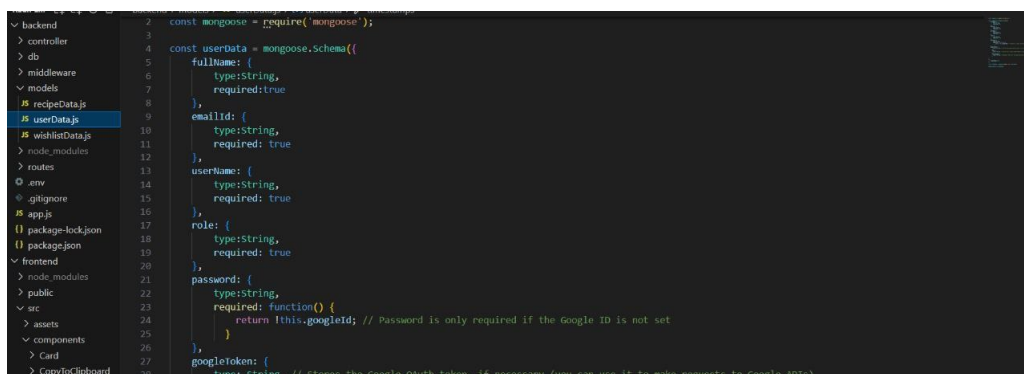
5.3.5 External API Management

- <https://api.spoonacular.com/recipes/random>
 - Purpose: Get random recipes from the spoonacular api.
- <https://api.spoonacular.com/recipes/id/information>
 - Purpose: Fetches details of a specific recipe by its ID from spoonacular api.
- <https://api.spoonacular.com/recipes/findByIngredients>
 - Purpose: Find recipes that either maximize the usage of ingredients at hand or minimize the ingredients that don’t currently have.

Chapter 6

Database Implementation

The database for Delizia is implemented using MongoDB Atlas, a cloud-based solution offering scalability and reliability. It organizes data into collections for entities like Users, Recipes, Favorites, and Ingredients, ensuring efficient storage and retrieval. MongoDB Atlas provides features like automated backups, data redundancy, and real-time synchronization, which enhance both performance and reliability. The flexible schema of MongoDB allows for easy handling of various data types, such as recipe details, user profiles, and ingredient lists. With MongoDB Atlas, Delizia can scale efficiently as the app grows, providing a seamless experience for users as they search for, save recipes. Additionally, its cloud-based nature ensures high availability and easy maintenance.

The image shows a code editor with a file explorer on the left and code on the right. The file explorer lists various files and folders, including 'backend', 'controller', 'db', 'middleware', 'models', 'recipeData.js', 'userData.js', 'wishlistData.js', 'node_modules', 'routes', '.env', '.gitignore', 'app.js', 'package-lock.json', 'package.json', 'frontend', 'node_modules', 'public', 'src', 'assets', 'components', 'Card', and 'CopyToClipboard'. The 'userData.js' file is selected. The code in the editor defines a Mongoose schema for a user, with fields for fullName, emailId, username, role, password, and googleToken. The password field has a custom required function that checks if the Google ID is not set. The googleToken field is a string type with a comment explaining its purpose.

```
1 const mongoose = require('mongoose');
2
3
4 const userData = mongoose.Schema({
5   fullName: {
6     type: String,
7     required: true
8   },
9   emailId: {
10    type: String,
11    required: true
12  },
13  username: {
14    type: String,
15    required: true
16  },
17  role: {
18    type: String,
19    required: true
20  },
21  password: {
22    type: String,
23    required: function() {
24      return !this.googleId; // Password is only required if the Google ID is not set
25    }
26  },
27  googleToken: {
28    type: String, // Stores the Google OAuth token, if necessary (you can use it to make requests to Google APIs)
```

Figure 6.1: userdata model

```

✓ RECIPE... backend > models > JS recipeData.js > ...
2 const mongoose = require("mongoose");
3
4 const recipeData = mongoose.Schema({
5   title: {
6     type:String,
7     required:true
8   },
9   category: {
10    type:[String],
11    default:[],
12    required:true
13  },
14  descriptions: {
15    type:String,
16    required:true
17  },
18  instructions: {
19    type:String,
20    required:true
21  },
22  analyzedInstructions: [{
23    number: {type:Number},
24    step:{type:String}
25  }],
26  vegetarian: {
27    type:Boolean,
28    required:true

```

Figure 6.2: recipedata model

```

✓ backend 3 const mongoose = require("mongoose");
4
5 const wishlistModel = mongoose.Schema({
6   userId: {
7     type:String,
8     required:true
9   },
10  savedRecipes: [
11    {
12      recipeId:(type:String),
13      title:(type:String),
14      image:(type:String),
15      category:(type:[String],default:[]),
16      description:(type:String),
17      vegetarian:(type:Boolean),
18      cookingTime:(type:Number),
19      isExternal:(type:Boolean)
20    }
21  ],
22  timestamps: true
23 }
24
25 const wishlistData = mongoose.model('wishlist',wishlistModel);
26
27
28
29

```

Figure 6.3: wishlistdata model

Chapter 7

Deployment

Deploying a project using AWS involves leveraging various AWS services to host and manage the frontend, backend, and database components. The frontend can be deployed on AWS S3 for static hosting, optionally accelerated using CloudFront for faster global delivery and HTTPS support. The backend can be hosted on an EC2 instance, configured with Node.js or another server runtime, and secured using NGINX as a reverse proxy. For data storage, a managed database like RDS (Relational Database Service) or DynamoDB can be used, ensuring scalability and security. DNS management is handled via Route 53, linking a custom domain to the deployed services. Monitoring and scaling can be managed using CloudWatch and Auto-Scaling Groups, making the deployment reliable, efficient, and ready to handle varying traffic demands.

Chapter 8

Results

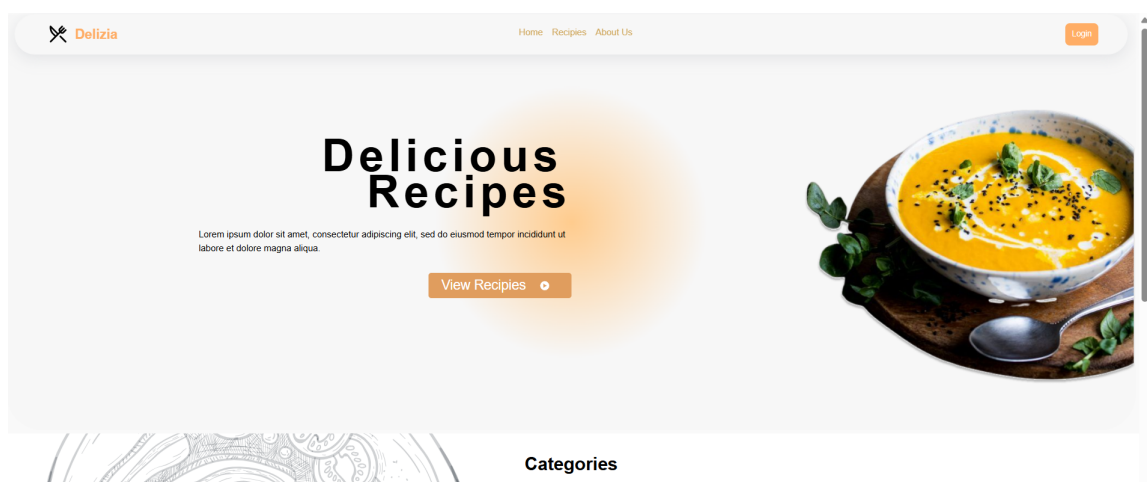


Figure 8.1: Home Page

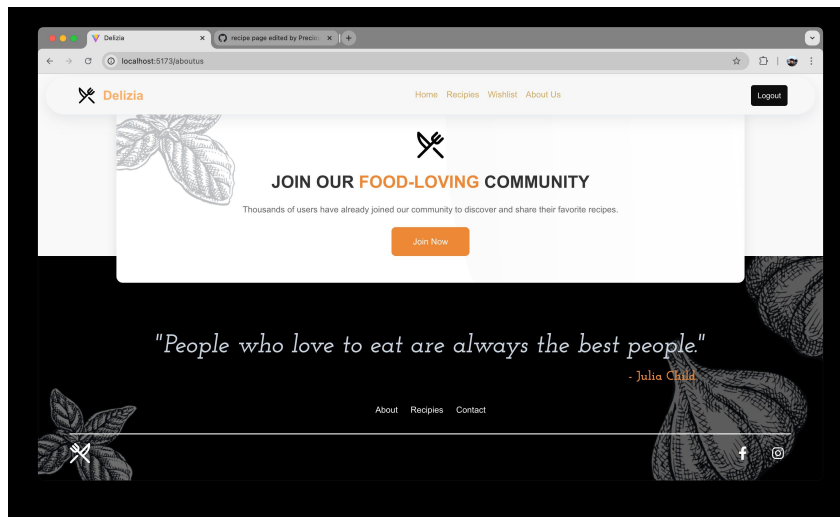


Figure 8.2: Footer Page

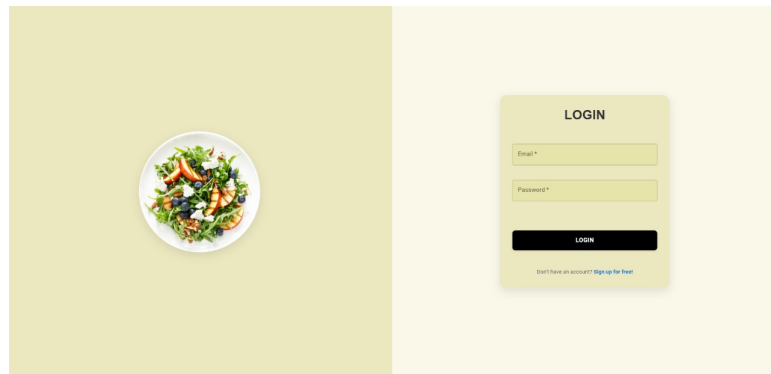


Figure 8.3: Login

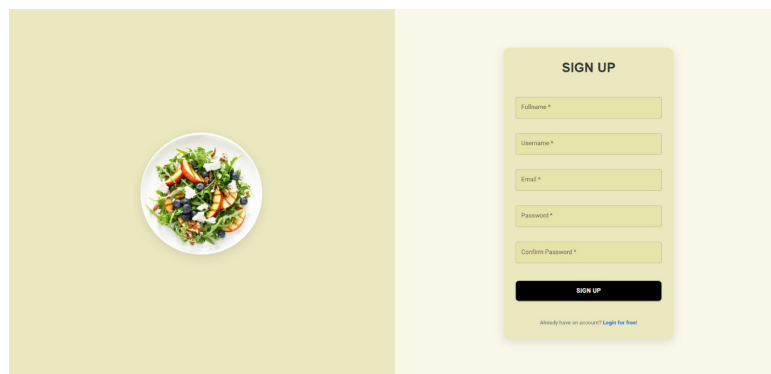


Figure 8.4: Signup

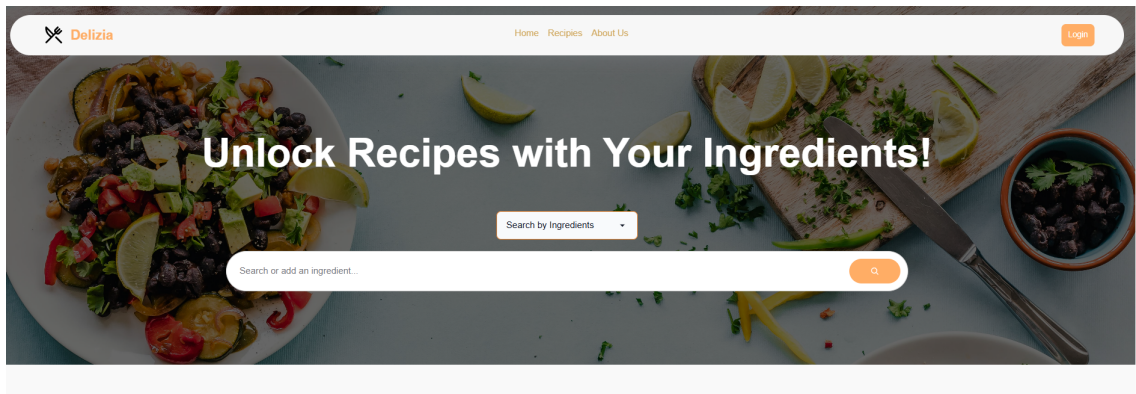


Figure 8.5: Search page

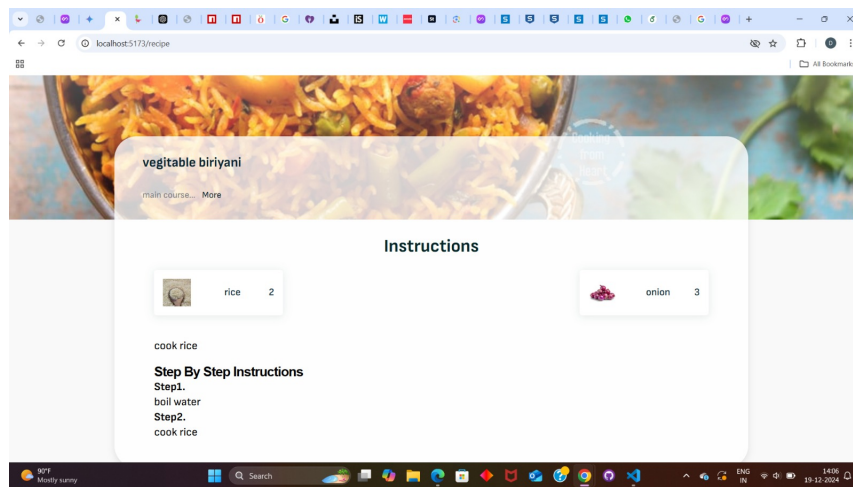


Figure 8.6: Recipe Page

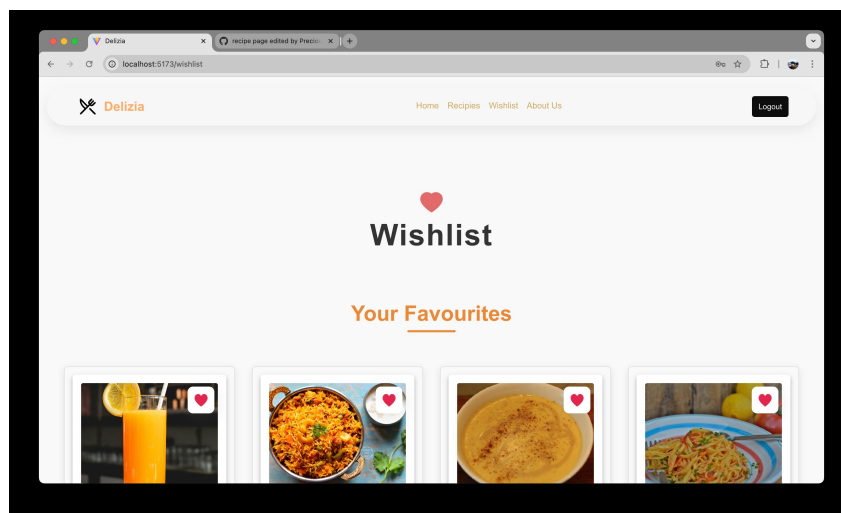


Figure 8.7: Wishlist Page

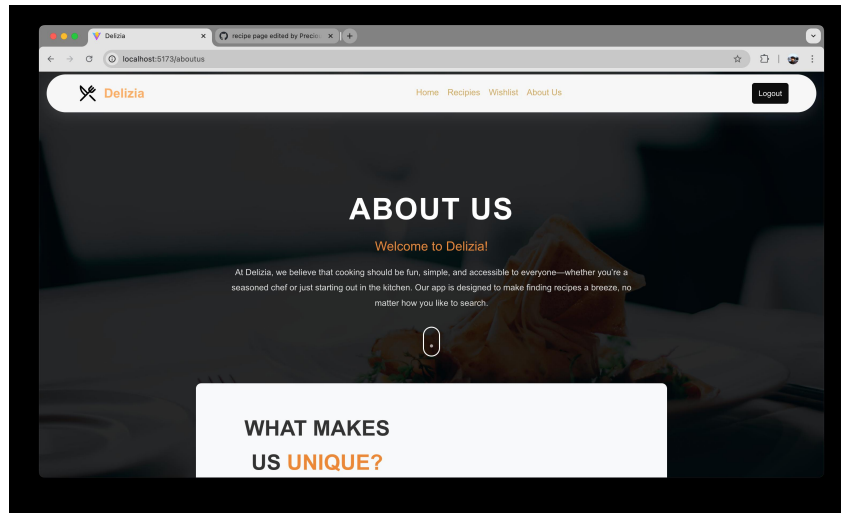


Figure 8.8: About us Page

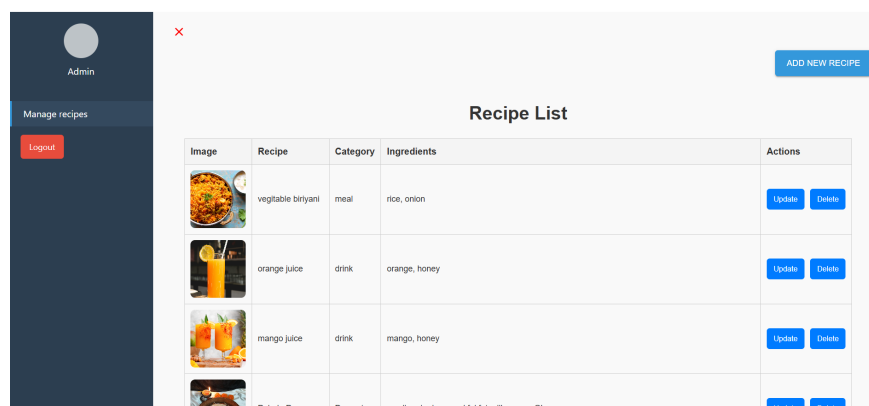


Figure 8.9: Admin Dashboard

Chapter 9

Discussion

The Delizia is designed to cater to the needs of two key user roles: Admin and Users, with distinct functionalities and user experiences tailored to their specific requirements. For Admins, the platform provides tools to manage recipe data, including adding, editing, deleting, and listing recipes. Users, on the other hand, can search for recipes using ingredients, view detailed recipe descriptions, save their favorite recipes, and use features like text-to-audio for cooking steps and copy-to-clipboard for recipe instructions. The platform focuses on providing an intuitive and engaging experience for both roles.

The backend is developed using Node.js and MongoDB, enabling efficient data storage and retrieval with robust CRUD operations. Secure access is ensured through JWT-based authentication, with password hashing implemented using bcrypt to maintain data privacy and user security. The architecture is designed to be scalable and maintainable, incorporating middleware and modular APIs for efficient role-based access control and functionality management.

The frontend, built with React.js, is designed with a focus on responsive design principles, ensuring seamless usability across a variety of devices, from desktops to mobile phones. Material-UI is used to create a visually appealing and consistent interface, while tools like Redux and Axios manage state and API communication efficiently. Features such as ingredient-based recipe search, and dynamic recipe detail pages enhance the overall user experience, making the platform intuitive and easy to navigate.

The application integrates advanced features, such as text-to-audio functionality for cooking steps, ingredient-based search using external APIs, and copy-to-clipboard for sharing recipe instructions. These features are designed to make cooking more convenient and enjoyable for users.

Looking ahead, the project has the potential to incorporate advanced features such as AI-driven recipe recommendations, guided cooking experiences with video or voice assistance, and community-driven functionalities like user-uploaded recipes and recipe reviews. These enhancements aim to transform the platform into a comprehensive and interactive space for discovering, sharing, and managing recipes.

This Recipe Making Application is designed to provide a seamless and efficient recipe discovery experience for all users, with scalability and user-centric design at its core. It lays a solid foundation for future enhancements, aspiring to become a reliable platform for culinary enthusiasts.

Chapter 10

Conclusion

Our recipe app combines innovative functionalities to enhance the meal planning and cooking experience. With features like user registration (for both Admins and Users), secure login, the app ensures ease of access and personalization. Users can search for recipes based on available ingredients through both external APIs and the app's database, with detailed recipe pages that provide information such as ingredients, cooking steps, prep time, and more. The app also allows users to save or favorite recipes, and utilize convenient tools like copying recipe descriptions and cooking steps to the clipboard. Additionally, it offers a text-to-audio feature for step-by-step instructions. For admins, the app includes a comprehensive recipe data management system that allows for adding, editing, deleting, and listing recipes, providing full control over the recipe database. This seamless integration of functionality aims to make cooking more enjoyable, efficient, and accessible, empowering users to create meals that fit their tastes and dietary preferences.

10.1 Future Enhancement

The future enhancement of our project can involve incorporating advanced technologies and features to further improve user experience, streamline operations, and adapt to changing trends in the digital landscape. Here are some potential enhancements to consider:

- **Search by Image:** Users can upload or take a picture of an ingredient or dish,

and the app will recognize it and suggest relevant recipes. This feature will use image recognition technology to identify ingredients or meals, allowing users to discover new recipes without manually entering ingredients.

- **Search by Recipe Using Camera:** Users can scan recipe cards or images of meals with their camera to instantly retrieve step-by-step instructions and ingredient lists. This will make it easy for users to access recipes they find in books, magazines, or on social media platforms.
- **Nutritional Analysis and Meal Planning:** Introduce a feature that provides nutritional information for each recipe, including calorie count, macronutrients, and other important details.
- **Recipe Scaling and Servings Adjustments:** Allow users to easily scale recipes based on the number of servings needed, adjusting ingredient quantities accordingly. This feature will help users cook for different group sizes, whether it's a single meal or a large gathering.
- **Augmented Reality (AR) Cooking Instructions:** Introduce AR features to provide step-by-step visual instructions for cooking, helping users visualize the cooking process with interactive overlays. This would guide users through complex recipes or unfamiliar techniques in an immersive and intuitive way.
- **Meal Prep and Storage Tips:** Provide tips on how to store leftover ingredients or cooked meals to reduce waste and extend shelf life. Include guidance on meal prep and make-ahead recipes to help users save time during the week.
- **Longitudinal Tracking and Insights:** Develop features that track mental health trends over time, offering insights into progress or potential triggers. Visualization tools can help users understand their mental health journey.
- **Crisis Intervention and Support:** Include emergency contact information, hotlines, or crisis intervention tools for users in immediate need of assistance.
- **Feedback and Improvement Loop:** Encourage user feedback and iterate on the app based on user suggestions and experiences, ensuring continual improvement and relevance.

Chapter 11

References

<https://cloudinary.com/>

<https://mui.com/material-ui/>

<https://www.geeksforgeeks.org/>

<https://www.w3schools.com/>

<https://nodejs.org/en/docs/>

<https://expressjs.com/>

<https://www.mongodb.com/docs/>

<https://mongoosejs.com/docs/>

<https://jwt.io/introduction/>

<https://github.com/kelektiv/node.bcrypt.js/>

<https://www.mongodb.com/docs/atlas/>

<https://cloud.google.com/text-to-speech/docs>

<https://clipboardjs.com/>

<https://spoonacular.com/food-api/docs>

<https://dribbble.com/search/recipe-app>

<https://www.chakra-ui.com/docs/get-started/installation>