**Full Stack Development with MERN**

**Project Documentation**

**1. Introduction**

* **Project Title:** “Food Ordering App”

You will also introduce the ”MERN stack”, explaining that it uses:

* MongoDB for the database.
* Express.js as the backend framework.
* React.js for the frontend interface.
* Node.js to handle server-side logic.

**2. Project Overview**

Food Hub is an innovative, full-stack food ordering application designed to revolutionize the way people order meals online. Built using modern web technologies, this platform aims to create a seamless connection between hungry customers and local restaurants, offering a user-friendly experience for both parties.

**Purpose:**

* Create a user-friendly platform for seamless online food ordering
* Develop an efficient system for restaurants to manage their menus and orders
* Aim to enhance the overall food ordering experience for both customers and restaurant owners
* Address pain points in existing food ordering systems

**Features:**

* Implement a robust user registration and authentication system
* Design comprehensive restaurant and menu browsing capabilities
* Develop an intuitive order placement process with real-time tracking
* Create a customizable restaurant dashboard for menu and order management
* Integrate real-time notifications for order updates
* Ensure responsive design for optimal user experience on both mobile and desktop devices
* Develop a rating and review system for restaurants and individual dishes.

**3. Architecture**

The Food Hub application follows a typical MERN (MongoDB, Express.js, React, Node.js) stack architecture, which provides a robust and scalable foundation for building modern web applications. Let's break down each component

**Frontend:**

1.Component Structure:

* Implement a modular component-based architecture using React.
* Utilize functional components with hooks for state management and side effects
* Create reusable UI components for common elements (buttons, forms, cards).
* Implement container components for managing data and state, separating concerns from presentational components

2. Routing:

* Utilize React Router for client-side routing
* Implement protected routes for authenticated users (e.g., user profile, order history)
* Use lazy loading for route components to improve initial load time

3. API Integration:

* Create a centralized API service using Axios for making HTTP requests to the backend
* Implement request/response interceptors for handling authentication tokens and errors.

4. Performance Optimization:

* Implement code splitting to reduce initial bundle size
* Use React.memo and use Memo for performance optimization where necessary
* Implement virtualization for long lists

**Backend:**

1.Server Setup:

* Use Express.js as the web application framework
* Implement middleware for parsing requests, handling CORS, and logging

2. API Structure:

* Design RESTful API endpoints following best practices
* Implement versioning for the API.
* Use controllers to handle the logic for each route.

3. Data Validation:

* Use a validation library like Joi or express-validator for input validation
* Implement middleware for validating request data before processing.

4. Security:

* Implement rate limiting to prevent abuse
* Use helmet middleware for setting various HTTP headers for security
* Implement CSRF protection for form submissions.

**Database:**

* **Data Schema**

1.User Schema:

* Fields: id, name, email, password (hashed), address, phone, created\_at, updated\_at
* Indexes: email (unique).

2.Restaurant Schema:

* Fields: id, name, description, address, phone, cuisine\_type, rating, operating\_hours, created\_at, updated\_at
* Indexes: name, cuisine\_type.

3. Menu Item Schema:

* Fields: id, restaurant\_id, name, description, price, category, image\_url, is\_available, created\_at, updated\_at
* Indexes: restaurant\_id, category.

4. Order Schema:

* - Fields: id, user\_id, restaurant\_id, items (array of menu items with quantity), total\_amount, status, delivery\_address, created\_at, updated\_at
* - Indexes: user\_id, restaurant\_id, status.

5. Review Schema:

* Fields: id, user\_id, restaurant\_id, rating, comment, created\_at
* Indexes: restaurant\_id, user\_id.
* **Database Interactions:**
* Use Mongoose ODM (Object Document Mapper) for interacting with MongoDB
* Implement data models corresponding to the schemas
* Use Mongoose middleware for data validation and pre/post-save operations
* Implement database transactions for operations that require atomicity (e.g., placing an order).

**4. Setup Instructions**

* **Prerequisites:**
  + Specify required Node.js version (compatible with React 18.3.1)
  + Include instructions for installing npm or yarn package manager
  + Provide guidance on setting up a local MongoDB instance or using a cloud-based solution like MongoDB Atlas
  + List any additional software requirements (e.g., Git for version control).
* **Installation:**
  + Provide step-by-step instructions for cloning the project repository from version control
  + Detail the process of installing project dependencies using npm or yarn
  + Guide users through setting up necessary environment variables for configuration
  + Include instructions for initializing the database with seed data if applicable
  + Provide commands for running the application in development mode

**5. Folder Structure**

* **Client:**
* **src/**: This is the main source directory for the React application. It contains all the code related to the frontend.
* **components/**: This directory holds reusable UI components that can be used throughout the application. Examples include buttons, input fields, navigation bars, and modals.
* **pages/**: This folder contains the various pages of the application, each representing a different view or route. For instance, there may be a Home page, a Restaurant Menu page, a Cart page, and a User Profile page.
* **services/**: This directory is responsible for handling API calls to the backend. It contains functions that interact with the server, such as fetching restaurant data, submitting orders, and managing user authentication.
* **App.js**: This is the main component that serves as the entry point for the React application.
* **index.js**: This file is the entry point for the React application, where the React app is rendered into the DOM
* **Server:**
  + **models/**: This directory contains the Mongoose models that define the schemas for the MongoDB collections. Each model corresponds to a specific entity in the application, such as User, Restaurant, and Order
  + **routes/**: This folder contains the Express route handlers that define the API endpoints. Each file typically corresponds to a specific resource
  + **controllers/**: This directory contains the controller functions that implement the business logic for each route. Controllers handle the processing of requests, interact with the models to retrieve or manipulate data, and send responses back to the client.
  + **middleware/** (optional): If the application uses middleware for tasks such as authentication, logging, or error handling, this folder can contain those middleware functions.
  + **config/**: This directory may contain configuration files, such as database connection settings or environment variable management.
  + **server.js**: This is the main entry point for the Node.js application. It initializes the Express server, sets up middleware, connects to the database, and defines the routes.
  + **package.json:** This file contains metadata about the project, including dependencies, scripts, and other configurations necessary for running the Node.js application.

**6. Running the Application**

* **Frontend:**
  + cd '.\Food Frenzy\'
  + cd .\project\
  + npm run dev
  + http://localhost:5173/
* **Backend:**
  + npm run seed
  + npm run dev
  + http://localhost:5000](http://localhost:5000)

**7. API Documentation**

**User Login**

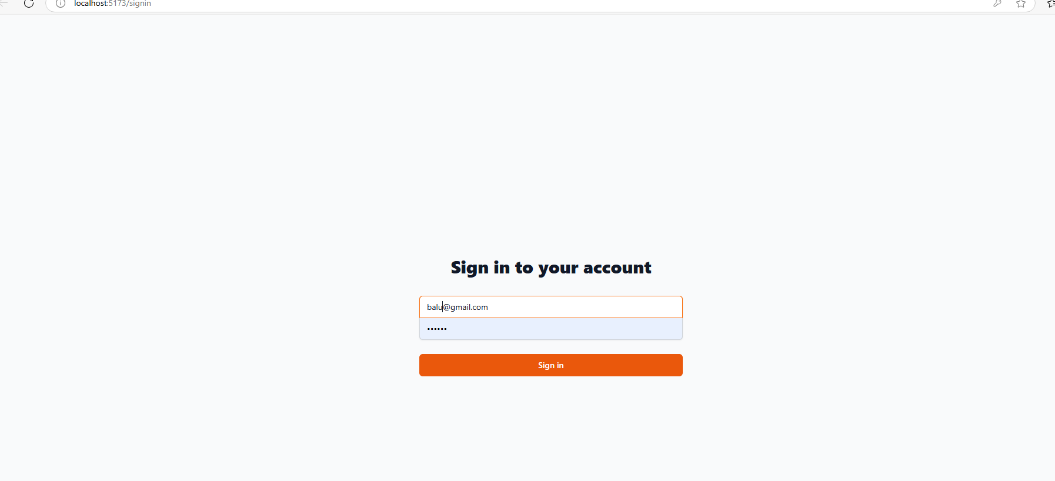
* Endpoint: POST /api/users/login
* Description: This endpoint authenticates a user and returns a token for session management.
* Request Body Parameters:
  + email (string): The email address of the user.
  + password (string): The password for the account.
* Example Request:
  + A user submits the following data to log in:
    - Email: "john@example.com"
    - Password: "securePassword123"

**8. Authentication**

Authentication is the process of verifying the identity of a user. In this project, authentication is primarily handled through a combination of username/email and password.

**Process:**

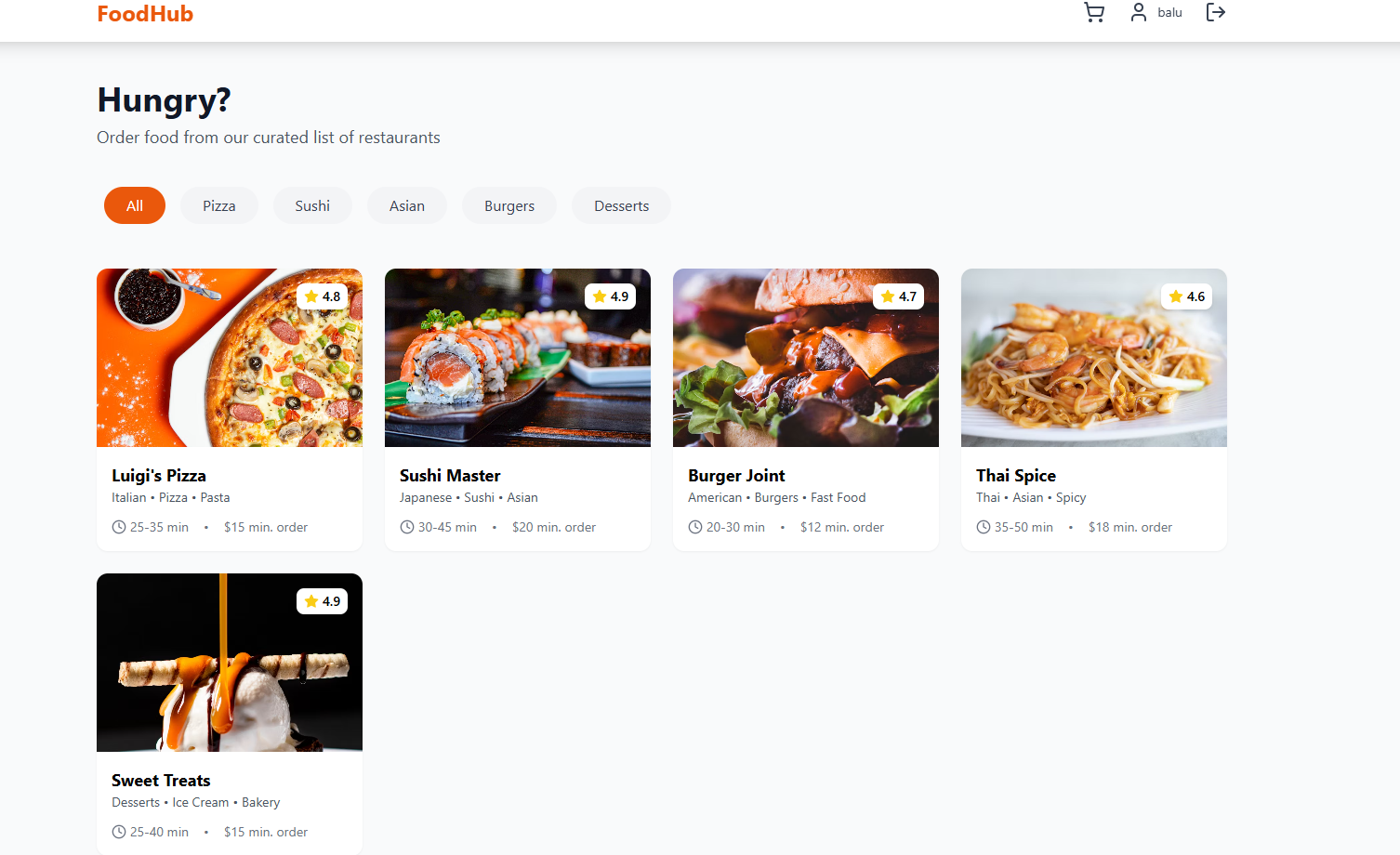
* **User Registration:** When a new user registers, they provide a username, email, and password. The password is securely hashed before being stored in the database to protect user credentials.
* **User Login:** During the login process, users submit their email and password. The backend verifies these credentials by comparing the provided password (after hashing) with the stored hash in the database.
* **Token Generation:** Upon successful authentication, the server generates a JSON Web Token (JWT). This token contains encoded user information and is signed to prevent tampering. The token is then sent back to the client.

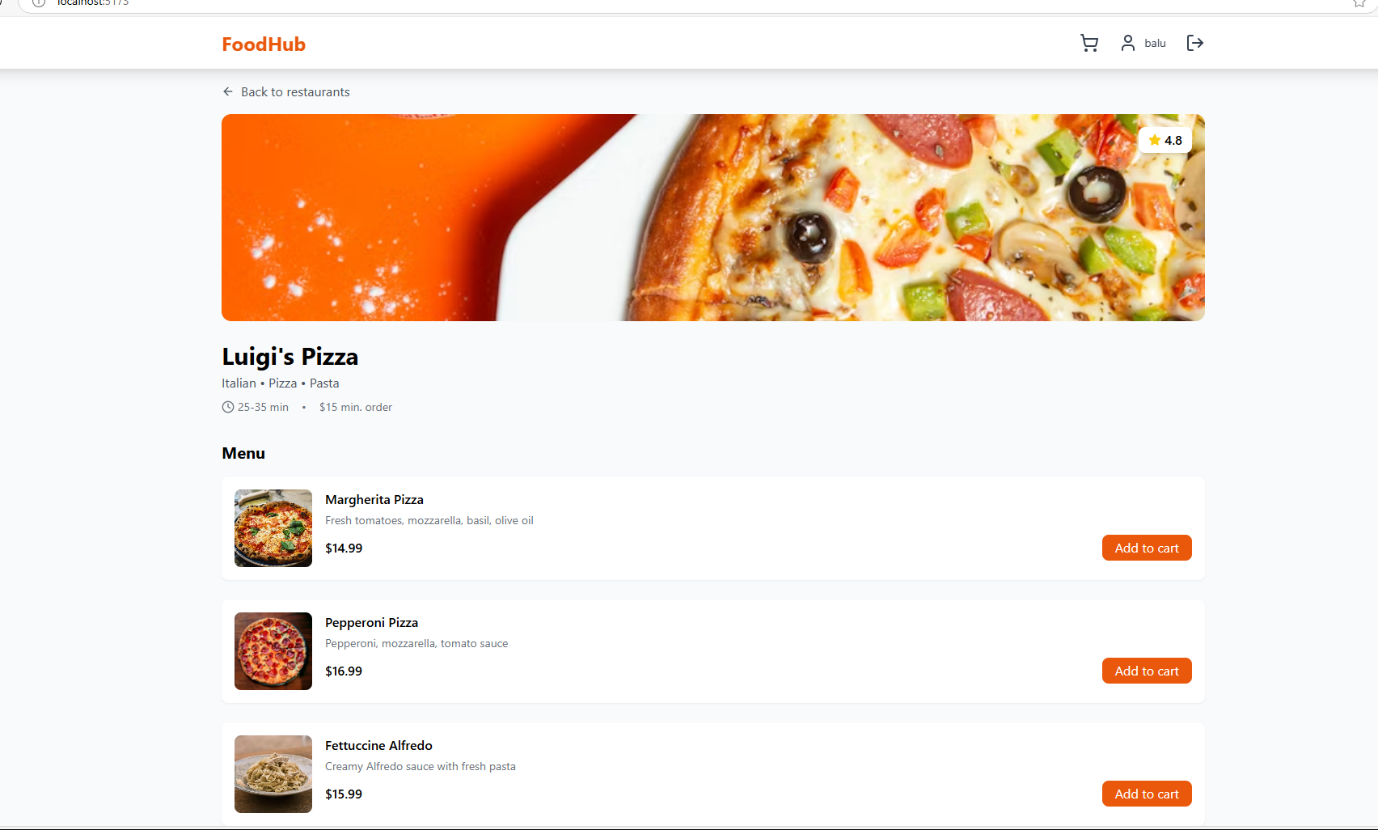
**9. User Interface**

**10. Testing**

* **Unit Testing**
  + Jest: A popular JavaScript testing framework that allows developers to write unit tests for both frontend and backend components. It provides a simple API for assertions and mocking.
  + Mocha/Chai: An alternative testing framework that can be used for unit testing, especially in Node.js applications.
* **Testing Types**
  + Postman: Used for manual functional testing of API endpoints, allowing testers to send requests and verify responses.
  + Cypress: Also used for functional testing in the context of E2E scenarios.
* **Continuous Integration and Continuous Deployment (CI/CD)**
  + GitHub Actions: A CI/CD tool that automates the testing and deployment process. It can be configured to run tests whenever code is pushed to the repository.
  + Travis CI or CircleCI: Alternative CI/CD tools that can also be integrated into the development workflow for automated testing and deployment.

**11. Screenshots or Demo**

****

****

**12. Known Issues**

* **Inconsistent Button Styles**
* Some buttons across the application have inconsistent styles, leading to a disjointed user experience.
* **Order Confirmation Email Delay**
* Users occasionally experience delays in receiving order confirmation emails after placing an order.
* **Slow Loading Times on Menu Page**
* The menu page can take longer to load during peak hours due to high traffic
* **High Memory Usage**
* The application may experience high memory usage on the server during heavy load, leading to potential crashes.

**13. Future Enhancements**

* **Advanced Search and Filtering Options**
* Implement advanced search features that allow users to filter restaurants and **Multiple Payment Options** options).
* **Order Scheduling**
* Allow users to schedule their orders for a future date and time, providing flexibility for meal planning.
* **Multiple Payment Options**
* Expand payment options to include digital wallets (e.g., Apple Pay, Google Pay) and cryptocurrency.
* **Social Sharing Options**
* Enable users to share their favorite meals or restaurant experiences on social media platforms.