Full Stack Development with MERN

Project Documentation format

1. Introduction

- **Project Title:** [Cafeteria Menu Display]
- **Team Members:** Pachipalu Kranthi Kumar Reddy[lead]
  Palannagari Tejasri
  Pathakotha Harinath
  Posa Jayavardhan

2. Project Overview

- **Purpose:** Cafeteria menu display is to inform customers of available food options, prices, and specials clearly and efficiently.

- **Features:** Dynamic menu updates, real-time pricing, nutritional info, visual appeal, and user-friendly interface

3. Architecture

- **Frontend:** Leverages component-based design to render dynamic menu items, handle state management, and enable real-time updates via APIs.
- **Backend:** Handles API routing, manages menu data with a database, and supports real-time updates and admin controls.
- **Database:** Collections like `menus`, `items`, and `status`, where interactions include CRUD operations to fetch, update, and display daily menu items dynamically on the front-end.

4. Setup Instructions

- **Prerequisites:** List software dependencies (e.g., Node.js, MongoDB).
- **Installation:** Step-by-step guide to clone, install dependencies, and set up the environment variables.

5. Folder Structure

- **Client:** `MenuNAME`, `MenuItem`, and `status`

- **Server:** Express with modular routes, controllers for handling logic, and a MongoDB database connection through Mongoose for managing cafeteria menu data.

6. Running the Application

- Provide commands to start the frontend and backend servers locally.
  - **Frontend:** npm start in the client directory.
  - **Backend:** node server.js in the server directory.

7. API Documentation

- `GET /menu`, `POST /menu`, `PUT /menu/:id`, `DELETE /menu/:id`, and optionally `GET /menu/category/:category`
- **GET /menu** – fetch all items, no params, returns `[{ id, name, price, category`

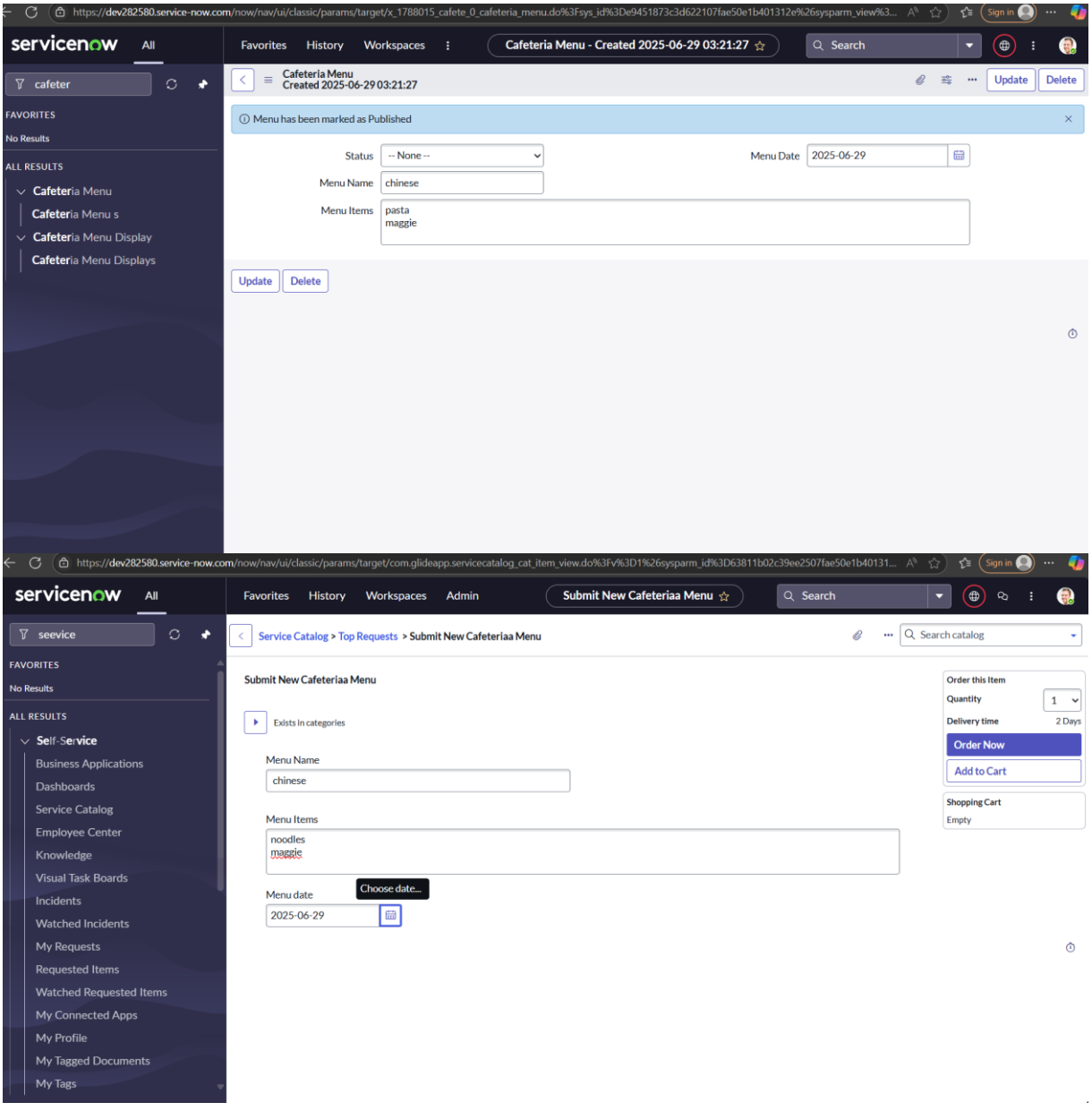
`}];`

POST /menu – create item, body: `{ name, price, category }`, returns `{ id, name, price, category };`

8. Authentication

- Are handled using JSON Web Tokens (JWT), where users log in to receive a token that must be included in request headers to access or modify protected routes.
- JWT tokens are generated upon login, stored on the client side (e.g., in localStorage), and sent in the `Authorization` header (`Bearer <token>`) with each request to validate user sessions and permissions.

9. User Interface



10. Testing

- Strategy includes unit tests with **Jest** for backend logic, **React Testing Library** for frontend component testing, and **Postman** or **Supertest** for API endpoint validation.

11. Screenshots or Demo

- https://drive.google.com/file/d/1cTKFq-hCNUZh-3sSqaW_0tmyV8BZ_s5L/view?usp=sharing

12. Known Issues

- Include occasional API delays, lack of input validation on some forms, and missing role-based access control which may allow unauthorized edits if tokens are exposed.

13. Future Enhancements

- Future improvements could include adding role-based access control, real-time menu updates with WebSockets, mobile app integration, and enhanced UI with filtering and search capabilities.