Internship Assessment Task: "The Digital Diner" - Mini Restaurant Ordering System

Scenario

"The Digital Diner" is a small, popular restaurant looking to improve its customer experience by allowing users to browse their menu and place simple pickup orders online. They need a basic web application to facilitate this. You have been tasked with building a prototype of this system.

Objective

Develop a full-stack web application using the MERN + PostgreSQL stack (MongoDB, Express, React, Node.js) with PostgreSQL integrated for specific data. The application should allow users to view the menu, add items to a cart, and place a simplified order. The final frontend application should be deployed on Netlify.

Core Requirements

Menu Display

Users should be able to see a list of available menu items, potentially categorized (e.g., Appetizers, Main Courses, Desserts, Drinks).

Shopping Cart

Users should be able to add items from the menu to a shopping cart. They should be able to view the cart contents and the total price. (Removing items/updating quantity is a bonus).

Order Placement

Users should be able to "place" an order. For simplicity, this will involve providing basic contact information (Name, Phone Number) and submitting the cart contents. No payment processing is required.

Order Confirmation/History (Basic)

After placing an order, display a confirmation message. A simple view to see past orders associated with a phone number would be a good addition. (Full user authentication is optional but demonstrates stronger skills).

Technical Requirements & Tasks

Database Design (Thinking & Implementation)

Requirement: Design the database schemas needed for this application. Critically, you must use both MongoDB and PostgreSQL.

Task:

- Decide which data belongs in MongoDB and which belongs in PostgreSQL. Justify your choice in your documentation (e.g., README file). Consider the nature of the data (structured vs. unstructured, relations, query patterns).
- Suggestion: You might consider using MongoDB for the menu items (flexible schema, potentially nested details) and PostgreSQL for user/order information (structured, relational).
- Implement the schemas/models using appropriate tools (e.g., Mongoose for MongoDB, an ORM like Sequelize or node-postgres for PostgreSQL).

Backend API Development (Node.js & Express)

Requirement: Create a RESTful API to serve data to the frontend and handle business logic.

Task:

- Set up a Node.js/Express server.
- Connect to both your MongoDB and PostgreSQL databases.
- Implement API endpoints for:
    - Fetching all menu items (potentially by category).
    - Fetching a single menu item's details (if needed).
    - Creating a new order (receiving cart data and user info).
    - Fetching orders based on user identifier (e.g., phone number).
    - (Bonus) Endpoints for adding/editing menu items (can be tested via Postman/curl if no admin UI is built).
- Implement basic data validation for incoming requests.
- Ensure proper error handling and response statuses.

Frontend Development (React)

Requirement: Build a user-friendly interface using React.

Task:

- Set up a React application (e.g., using Create React App or Vite).
- Create components for:
    - Displaying menu items.
    - The shopping cart view.
    - The order placement form.

    - Order confirmation/history view.
- Implement client-side routing (e.g., using React Router) if needed for different views.
- Connect the frontend components to the backend API endpoints you created (use fetch or axios).
- Manage application state effectively (e.g., using Context API, Redux Toolkit, Zustand, or component state for cart management).
- Focus on functionality over complex styling, but ensure it's usable.

Deployment (Netlify)

Requirement: Deploy the frontend application.

Task:

- Deploy your React frontend application to Netlify.
- Ensure the deployed frontend can successfully communicate with your backend API. (Your backend will need to be running somewhere accessible - e.g., Heroku free tier, Render, Fly.io, or provide clear instructions in the README on how to run it locally for testing the deployed frontend).
- Make sure Cross-Origin Resource Sharing (CORS) is configured correctly on your backend to allow requests from your deployed Netlify URL.

Evaluation Criteria

- **Database Design**: Logical separation of data between MongoDB and PostgreSQL, clear justification, well-structured schemas.
- **API Quality**: Well-designed, RESTful endpoints; proper use of HTTP methods and status codes; data validation; error handling.
- **Code Quality**: Clean, readable, well-organized code (both frontend and backend); meaningful variable names; comments where necessary.
- **Functionality**: Does the application meet the core requirements? Is it working as expected?
- **React Implementation**: Proper use of components, state management, API integration, and hooks.
- **Problem Solving**: How you approached the tasks, tackled potential issues, and justified design decisions (especially the DB choice).
- **Deployment**: Successful deployment of the frontend to Netlify and connectivity to the backend.
- **Git Usage**: Clear commit history and logical commits in the provided repository.

Submission

Provide a link to a Git repository (e.g., GitHub, GitLab) containing your complete source code (frontend and backend).

Include a comprehensive README.md file in the repository root that includes:

- Clear instructions on how to set up and run the backend locally (including database setup).
- Your justification for the MongoDB vs. PostgreSQL database design choices.
- A list of the API endpoints created.
- A link to the deployed frontend application on Netlify.
- Any assumptions made or challenges faced.
- Using AI code tools are encouraged but proper understanding of the code is needed.

Good luck! We are excited to see what you build.

There is no timeline, but the interviews will be scheduled on a first come first basis.