## Project Overview

The goal of this project was to create a job management dashboard with a Node.js backend and a React frontend. The project requirements included creating a REST API for managing job data and a frontend to interact with the API.

## Key Decisions and Approach

### Backend Development

1. **Technology Stack**:
    - **Node.js**: Chosen for its scalability and ease of use in building RESTful APIs.
    - **Express**: A lightweight framework for setting up the server and routing.
    - **TypeScript**: Used for type safety and better code quality.
    - **JSON File Storage**: Used to simulate a database for simplicity.
2. **Folder Structure**:
    - **src/models.ts**: Contains the data model and functions to read/write jobs from/to a JSON file.
    - **src/controllers.ts**: Contains the logic for handling API requests (CRUD operations).
    - **src/routes.ts**: Defines the API endpoints and links them to the corresponding controllers.
    - **src/index.ts**: Sets up the Express server and middleware.
3. **Data Handling**:
    - **Date Format**: Ensured all dates are stored and returned in the `2024-06-15T09:00:00Z` format for consistency.
    - **JSON Storage**: Used a JSON file (`data/jobs.json`) to store job data, making it easy to read and write job records.
4. **Security and Error Handling**:
    - Implemented basic error handling for operations like getting a job by ID, updating, and deleting jobs.
    - Used `cors` to handle cross-origin requests, allowing the frontend to communicate with the backend.

### Frontend Development

1. **Technology Stack**:
    - **React**: Chosen for its component-based architecture and ease of building interactive UIs.
    - **TypeScript**: Used for type safety and better code quality.
    - **React Router**: Used for navigation between different views (job list, job details, add/edit job).
2. **Component Structure**:
    - **JobList**: Displays a list of all jobs with options to edit and delete each job.
    - **JobDetail**: Displays detailed information about a selected job with an option to edit the job.

    - **JobForm**: Used for both adding new jobs and editing existing jobs, ensuring form data is consistent with the backend's requirements.
    - **Header**: Used for Showing the top header with the associated ProfitFill Logo.
3. **Styling**:
    - Used CSS for styling components, ensuring a consistent and responsive design.
    - Ensured buttons and form elements are aligned and properly styled for a better user experience.
4. **Date Handling**:
    - Ensured date inputs in forms are compatible with the `datetime-local` format required by the backend.
    - Converted dates to the correct format before sending them to the backend.

## Implementation Steps

### Backend

1. **Initialize Project**:
    - Set up a Node.js project with TypeScript, Express, and necessary dependencies.
    - Configured TypeScript and created basic folder structure.
2. **Create Models and Controllers**:
    - Defined job data model and functions for reading/writing JSON data.
    - Implemented CRUD operations in the controllers.
3. **Set Up Routing**:
    - Defined API routes and linked them to the appropriate controller functions.
4. **Handle Date Format**:
    - Ensured all dates are stored in the `2024-06-15T09:00:00Z` format.
    - Used `toISOString` for date conversions.

### Frontend

1. **Initialize Project**:
    - Set up a React project with TypeScript and necessary dependencies.
    - Configured React Router for navigation.
2. **Create Components**:
    - Implemented `JobList`, `JobDetail`, and `JobForm` components.
    - Ensured components handle CRUD operations correctly and communicate with the backend.
3. **Styling**:
    - Applied CSS styles to ensure a consistent and user-friendly design.
    - Used Flexbox for layout and alignment of buttons and form elements.
4. **Date Handling**:
    - Ensured date inputs and outputs are compatible with the backend's requirements.