**Trainer Management App**

**Project Overview**

**Tech Stack**: React (Frontend) – Express (Backend) – MongoDB (Database) – Node.js – Bootstrap (Styling)

**Description**

The Trainer Management App is designed to help MTD, a training and development company, manage a growing database of freelance trainers across India. This app centralizes trainer data and allows admins to easily perform CRUD (Create, Read, Update, Delete) operations, ensuring that MTD's trainer database is current and accurately reflects trainer availability, skillsets, and location preferences.

**Table of Contents**

1. [Project Structure](#project-structure)
2. [Database Design](#database-design)
3. [API Endpoints](#api-endpoints)
4. [Frontend Components](#frontend-components)
5. [Backend Architecture](#backend-architecture)
6. [Additional Functionalities](#additional-functionalities)
7. [Setup and Installation](#setup-and-installation)

**1. Project Structure**

trainer-management-app/

── backend/

── controllers/

── models/

── routes/

── server.js

──.env

── frontend/

── public/

── src/

── components/

── App.jsx

── index.html

── package.json

* **backend/**: Contains the Node.js and Express.js API server
  + **controllers/**: Handles business logic for CRUD operations
  + **models/**: Mongoose models for MongoDB schema
  + **routes/**: API routes for trainer CRUD operations
* **frontend/**: Contains the React application for UI
  + **components/**: Reusable UI components
  + **src/**: Full-page components (e.g., Add Trainer, Edit Trainer)

**2. Database Design**

**Collections:** trainers

|  |  |  |
| --- | --- | --- |
| **Field** | **Type** | **Description** |
| \_id | ObjectId | Unique identifier for each trainer |
| trainer\_name | String | Name of the trainer |
| trainer\_location | String | Location of the trainer |
| trainer\_skills | [String] | List of skills (e.g., Java, C++, Python) |
| trainer\_phone | String | Contact phone number |

**3. API Endpoints**

**Base URL:** http://localhost:3000

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Method** | **Endpoint** | **Description** | **Request Body** | **Response** |
| GET | /api/trainers | Fetch all trainers | - | Array of trainers |
| GET | /api/trainers/:id | Fetch specific trainer | - | Trainer details |
| POST | /api/trainers | Add new trainer | { trainer\_name, trainer\_location, trainer\_skills, trainer\_phone } | Created trainer object |
| PUT | /api/trainers/:id | Update trainer by ID | { trainer\_name, trainer\_location, trainer\_skills, trainer\_phone } | Updated trainer object |
| DELETE | /api/trainers/:id | Delete trainer by ID | - | Success message with status |

**4. Frontend Components**

**Pages**

1. **HomePage** (/)
   * Displays a list of all trainers with add, edit and delete buttons.
2. **Add Trainer Page** (/add-trainer)
   * Contains a form for adding a new trainer with fields: trainer\_name, trainer\_location, trainer\_skills, trainer\_phone.
3. **Edit Trainer Page** (/edit-trainer/:id)
   * Form pre-filled with existing trainer data for updating.

**Components**

* **GetTrainers**: Component to display list of trainers.
* **AddTrainers**: Reusable form for adding.
* **EditTrainers**: Component to for editing trainers
* **Navbar**: Navigation bar with links to Home, Add Trainer.

**Routes Configuration (App.jsx)**

import React, { useState, useEffect } from 'react';

import { BrowserRouter as Router, Routes, Route, Link } from 'react-router-dom';

import './App.css';

import AddTrainer from '../components/AddTrainer';

import GetTrainer from '../components/GetTrainer';

import EditTrainer from '../components/EditTrainer';

import axios from 'axios';

function App(){

 return (

    <Router>

      <div className="container">

        <h1>Trainer Management</h1>

        <nav>

          <Link to="/">Trainer List</Link> | <Link to="/add-trainer">Add Trainer</Link>

          <link to="/hooks">Practice Hooks</link>

        </nav>

        <Routes>

          <Route path="/" element={<GetTrainer trainers={trainers} onDeleteTrainer={handleDeleteTrainer} />} />

          <Route path="/add-trainer" element={<AddTrainer onAddTrainer={handleAddTrainer} />} />

          <Route path="/edit-trainer/:id" element={<EditTrainer trainers={trainers} onUpdateTrainer={handleUpdateTrainer} />} />

        </Routes>

      </div>

    </Router>

  );

}

export default App;

**5. Backend Architecture**

1. **Controllers**: Functions that handle CRUD operations for trainers, such as getAllTrainers, getTrainerById, createTrainer, updateTrainer, and deleteTrainer.
2. **Routes**: Define routes and attach respective controller functions.
3. **Server Setup (server.js)**:
   * Initializes Express server, connects to MongoDB using Mongoose, cors and sets up API routes.

**Example Controller (TrainerController.js)**

const Trainer = require('../models/trainers');

exports.createTrainer = async (req, res) => {

try {

const trainer = new Trainer(req.body);

const savedTrainer = await trainer.save();

res.status(201).json(savedTrainer);

} catch (error) {

res.status(400).json({ message: error.message });

}

};

**6. Additional Functionalities**

* **Validation**: Use validation on both frontend (React) and backend (Express/Mongoose).
* **Styling**: Add Bootstrap for a user-friendly interface.

**7. Setup and Installation**

**Prerequisites**

* Node.js and npm
* MongodbCompass and Mongodb Community Server and mongosh installed and running

**Steps**

1. **Backend Setup**:
   * + cd backend
     + npm install mongodb mongoose express dotenv
2. **Environment Variables**:
   * + Create a .env file in the backend/ folder with:

PORT=3000

MONGO\_URI=mongodb://localhost:27017/trainer\_db

1. **Frontend Setup**:

cd frontend

npm create vite@latest “trainer\_app”

cd trainer\_app

npm install

1. **Running the Application**:
   * Start backend server:

nodemon server.js

* + Start frontend server:

npm run dev

The app should be running locally at http://localhost:3000.