**npm init –y**

npm i express

npm i ejs

npm i mongoose

\*\*\*GEMINI CONVERATTION\*\*\*\*

CAN U WRITE CODE FOR FULL WEB DEVELOPMENT PROJECT FOR A COMPUTER TRAINING INSTITUTE USING HTML, CSS, JAVA SCRIPT, NODE JS, NODE EXPRESS, NODE EJS AND MONGO DB WIT PROPER FILE STRUCTURE

**Crafting a Comprehensive Web Development Project for a Computer Training Institute**

**Understanding the Project Scope**

Before diving into the code, let's outline the core functionalities of our computer training institute website:

1. **Homepage:**
   * Hero section with institute's logo, tagline, and a prominent call-to-action (CTA) button.
   * About us section highlighting the institute's mission, vision, and values.
   * Featured courses section showcasing popular courses with brief descriptions and enrollment links.
   * Testimonials section displaying positive feedback from past students.
   * Contact us section with a contact form and contact information.
2. **Courses Page:**
   * A list of all available courses with detailed descriptions, duration, fees, and prerequisites.
   * A search bar to filter courses by category or keyword.
   * A course details page for each course, providing in-depth information and enrollment options.
3. **About Us Page:**
   * Detailed information about the institute's history, team, and achievements.
   * A team section showcasing the institute's faculty members with their profiles and contact information.
4. **Contact Us Page:**
   * A contact form to allow users to send inquiries and feedback.
   * Contact information, including address, phone number, and email address.
5. **Admin Dashboard:**
   * A dashboard for managing courses, students, and instructors.
   * Features for adding, editing, and deleting courses.
   * A student management system to track enrollments and progress.
   * An instructor management system to manage faculty and assign courses.

**Technology Stack**

* **Frontend:** HTML, CSS, JavaScript
* **Backend:** Node.js, Express.js
* **Templating Engine:** EJS
* **Database:** MongoDB

**File Structure**

project-name/

├── public/

│ ├── css/

│ │ ├── style.css

│ │ └── ...

│ ├── images/

│ │ ├── logo.png

│ │ └── ...

│ ├── js/

│ │ ├── script.js

│ │ └── ...

│ └── index.html

├── routes/

│ ├── courses.js

│ ├── about.js

│ ├── contact.js

│ └── admin.js

├── views/

│ ├── index.ejs

│ ├── courses.ejs

│ ├── about.ejs

│ ├── contact.ejs

│ └── admin/

│ ├── dashboard.ejs

│ ├── addCourse.ejs

│ └── ...

├── models/

│ ├── course.js

│ ├── user.js

│ └── ...

├── config/

│ └── database.js

├── app.js

└── package.json

**Code Implementation**

**Backend (Node.js, Express.js, MongoDB)**

JavaScript

// app.js

const express = require('express');

const mongoose = require('mongoose');

const ejs = require('ejs');

const app = express();

// Connect to MongoDB

mongoose.connect('mongodb://localhost/training\_institute')

.then(() => console.log('MongoDB connected'))

.catch(err => console.error('MongoDB connection error:', err));

// Set up EJS

app.set('view engine', 'ejs');

// Define routes

app.use('/', require('./routes/index'));

app.use('/courses', require('./routes/courses'));

app.use('/about', require('./routes/about'));

app.use('/contact', require('./routes/contact'));

app.use('/admin', require('./routes/admin'));

// Start the server

app.listen(3000, () => {

console.log('Server started on port 3000');

});

**Frontend (HTML, CSS, JavaScript)**

HTML

<!DOCTYPE html>

<html>

<head>

<title>Computer Training Institute</title>

<link rel="stylesheet" href="style.css">

</head>

<body>

</body>

</html>

**Key Considerations**

* **Security:** Implement proper security measures, including input validation, output encoding, and password hashing.
* **User Experience:** Design a user-friendly interface with intuitive navigation and clear calls to action.
* **Performance Optimization:** Optimize database queries, minimize HTTP requests, and leverage caching techniques.
* **Scalability:** Consider future growth and design the architecture to handle increased traffic and data.
* **Testing:** Thoroughly test the application to identify and fix bugs.

**Additional Features**

* **User Authentication and Authorization:** Implement user registration, login, and role-based access control.
* **Online Payment Integration:** Integrate a payment gateway to allow online course fee payments.
* **Email Notifications:** Send automated email notifications for course enrollments, reminders, and updates.
* **Real-time Chat:** Implement a chat feature for students to interact with instructors and peers.
* **E-learning Platform:** Integrate an e-learning platform for online course delivery and content management.

By following these guidelines and leveraging the power of the technology stack, you can create a robust and feature-rich website for your computer training institute.



CAN U TELL ME THE MAIN FILE NAME

The main file in a Node.js web application is typically named app.js. This file serves as the entry point for the application, handling the initialization of the server, routing, and middleware.

However, you can choose a different name if you prefer. The important thing is to configure your Node.js environment (e.g., package.json) to specify the correct entry point.

In the provided file structure, app.js is the primary file that starts the server and handles routing.



CAN U GIVE ME THE CODE OF APP.JS

JavaScript

const express = require('express');

const mongoose = require('mongoose');

const ejs = require('ejs');

const app = express();

// Connect to MongoDB

mongoose.connect('mongodb://localhost/training\_institute')

.then(() => console.log('MongoDB connected'))

.catch(err => console.error('MongoDB connection error:', err));

// Set up EJS

app.set('view engine', 'ejs');

// Define routes

app.use('/', require('./routes/index'));

app.use('/courses', require('./routes/courses'));

app.use('/about', require('./routes/about'));

app.use('/contact', require('./routes/contact'));

app.use('/admin', require('./routes/admin'));

// Start the server

app.listen(3000, () => {

console.log('Server started on port 3000');

});

This app.js file sets up the Express server, connects to the MongoDB database, configures EJS as the templating engine, and defines the routes for different parts of the website.

**Explanation:**

1. **Import necessary modules:**
   * express: For creating the web application.
   * mongoose: For interacting with the MongoDB database.
   * ejs: For rendering dynamic HTML templates.
2. **Create an Express app:**
   * const app = express(); initializes an Express application.
3. **Connect to MongoDB:**
   * mongoose.connect() establishes a connection to the specified MongoDB database.
4. **Set up EJS:**
   * app.set('view engine', 'ejs'); sets EJS as the default templating engine.
5. **Define routes:**
   * app.use() defines routes for different parts of the website. Each route is handled by a separate module in the routes directory.
6. **Start the server:**
   * app.listen(3000, ...) starts the server on port 3000 and logs a message to the console.

**Remember to:**

* Create the routes directory and define the route files (e.g., index.js, courses.js, about.js, contact.js, admin.js) to handle specific routes.
* Create the models directory to define Mongoose models for your data.
* Create the views directory to store your EJS templates.
* Configure your package.json file to specify the app.js file as the entry point.

By following these steps and customizing the code to your specific requirements, you can build a robust and feature-rich computer training institute website.



GIVE ROUTES CODE

**Here's a basic example of how you can structure your routes directory and the corresponding index.js file:**

JavaScript

// routes/index.js

const express = require('express');

const router = express.Router();

// Home page route

router.get('/', (req, res) => {

res.render('index', { title: 'Home' });

});

// About page route

router.get('/about', (req, res) => {

res.render('about', { title: 'About Us' });

});

// Contact page route

router.get('/contact', (req, res) => {

res.render('contact', { title: 'Contact Us' });

});

module.exports = router;

**Explanation:**

1. **Import necessary modules:**
   * express: For creating the router.
2. **Create a router:**
   * const router = express.Router(); initializes a new router.
3. **Define routes:**
   * **Home page route:**
     + router.get('/', ...) defines a GET route for the root path (/).
     + It renders the index.ejs template with the title "Home".
   * **About page route:**
     + router.get('/about', ...) defines a GET route for the /about path.
     + It renders the about.ejs template with the title "About Us".
   * **Contact page route:**
     + router.get('/contact', ...) defines a GET route for the /contact path.
     + It renders the contact.ejs template with the title "Contact Us".
4. **Export the router:**
   * module.exports = router; exports the router to be used in the app.js file.

**Additional Considerations:**

* **Dynamic Routes:** Use parameters to create dynamic routes, like /courses/:courseId.
* **POST Requests:** Handle form submissions using POST requests and middleware like body-parser.
* **Error Handling:** Implement error handling middleware to gracefully handle errors.
* **Authentication and Authorization:** Protect sensitive routes using authentication and authorization mechanisms.
* **Database Interactions:** Use Mongoose to interact with your MongoDB database and retrieve data for your views.

**Remember to create the corresponding EJS templates in the views directory:**

* index.ejs
* about.ejs
* contact.ejs

By following this approach and customizing it to your specific requirements, you can create a robust and flexible routing system for your computer training institute website.



give the index.ejs, about.ejs and contcat.ejs code

**index.ejs**

HTML

<!DOCTYPE html>

<html>

<head>

<title>Computer Training Institute</title>

<link rel="stylesheet" href="style.css">

</head>

<body>

<header>

<h1>Welcome to Our Training Institute</h1>

</header>

<main>

<section class="hero">

<h2>Learn New Skills, Achieve Your Goals</h2>

<p>Discover a wide range of courses to enhance your career.</p>

<a href="/courses" class="btn">Explore Courses</a>

</section>

<section class="about">

<h2>About Us</h2>

<p>We are dedicated to providing high-quality training and education...</p>

<a href="/about" class="btn">Learn More</a>

</section>

<section class="contact">

<h2>Contact Us</h2>

<p>Have any questions? Get in touch with us.</p>

<a href="/contact" class="btn">Contact Us</a>

</section>

</main>

<footer>

<p>&copy; 2023 Computer Training Institute</p>

</footer>

</body>

</html>

**about.ejs**

HTML

<!DOCTYPE html>

<html>

<head>

<title>About Us</title>

<link rel="stylesheet" href="style.css">

</head>

<body>

<header>

<h1>About Us</h1>

</header>

<main>

<p>Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.</p>

<h2>Our Mission</h2>

<p>Lorem ipsum dolor sit amet, consectetur adipiscing elit.</p>

<h2>Our Team</h2>

<ul>

<li>John Doe - CEO</li>

<li>Jane Smith - CTO</li>

<li>Mike Johnson - Instructor</li>

</ul>

</main>

<footer>

<p>&copy; 2023 Computer Training Institute</p>

</footer>

</body>

</html>

**contact.ejs**

HTML

<!DOCTYPE html>

<html>

<head>

<title>Contact Us</title>

<link rel="stylesheet" href="style.css">

</head>

<body>

<header>

<h1>Contact Us</h1>

</header>

<main>

<form>

<label for="name">Name:</label>

<input type="text" id="name" name="name" required>

<label for="email">Email:</label>

<input type="email" id="email" name="email" required>

<label for="message">Message:</label>

<textarea id="message" name="message" required></textarea>

<button type="submit">Send Message</button>

</form>

<h2>Contact Information</h2>

<p>

<strong>Address:</strong> 123 Main Street, Anytown, CA 12345<br>

<strong>Phone:</strong> (123) 456-7890<br>

<strong>Email:</strong> info@example.com

</p>

</main>

<footer>

<p>&copy; 2023 Computer Training Institute</p>

</footer>

</body>

</html>

**Remember to:**

* Create a styles.css file to style your HTML elements.
* Set up your database and models to store and retrieve information about courses, instructors, and students.
* Implement server-side logic to handle form submissions and database operations.
* Consider adding more sections, features, and customization to your website.



give style.css code

CSS

/\* style.css \*/

body {

font-family: Arial, sans-serif;

margin: 0;

padding: 0;

}

header {

background-color: #333;

color: #fff;

padding: 20px;

text-align: center;

}

main {

padding: 20px;

}

section {

margin-bottom: 20px;

}

h2 {

margin-bottom: 10px;

}

p {

margin-bottom: 10px;

}

a {

color: #333;

text-decoration: none;

}

a:hover {

text-decoration: underline;

}

.btn {

background-color: #333;

color: #fff;

padding: 10px 20px;

border-radius: 5px;

text-decoration: none;

}

.btn:hover {

background-color: #555;

}

footer {

background-color: #333;

color: #fff;

padding: 10px;

text-align: center;

}

This is a basic CSS style sheet to get you started. You can customize it further to match your specific design preferences.



give courses route code

JavaScript

const express = require('express');

const router = express.Router();

const Course = require('../models/Course'); // Assuming a Course model

// Route to display all courses

router.get('/', async (req, res) => {

try {

const courses = await Course.find();

res.render('courses', { courses });

} catch (err) {

console.error(err);

res.status(500).send('Server Error');

}

});

// Route to display a specific course

router.get('/:id', async (req, res) => {

try {

const course = await Course.findById(req.params.id);

res.render('course', { course });

} catch (err) {

console.error(err);

res.status(404).send('Course not found');

}

});

module.exports = router;

**Explanation:**

1. **Import necessary modules:**
   * express: For creating the router.
   * Course: The Mongoose model for courses.
2. **Route to display all courses:**
   * router.get('/'): Defines a GET route for the /courses path.
   * Course.find(): Fetches all courses from the database.
   * res.render('courses', { courses }): Renders the courses.ejs template, passing the courses array as a variable.
3. **Route to display a specific course:**
   * router.get('/:id'): Defines a GET route for a specific course ID.
   * Course.findById(req.params.id): Fetches the course with the specified ID.
   * res.render('course', { course }): Renders the course.ejs template, passing the course object as a variable.

**Remember to:**

* Create the Course Mongoose model to define the structure of your courses.
* Create the courses.ejs and course.ejs templates to display the course information.
* Handle potential errors and provide appropriate error messages.
* Consider adding more routes for specific functionalities like searching, filtering, or adding/editing courses.
* Implement proper error handling and input validation to ensure security and reliability.

By following these steps, you can create a robust and flexible routing system for your computer training institute website.



write code form models

JavaScript

// models/Course.js

const mongoose = require('mongoose');

const courseSchema = new mongoose.Schema({

name: {

type: String,

required: true

},

description: {

type: String,

required: true

},

duration: {

type: String,

required: true

},

fee: {

type: Number,

required: true

},

prerequisites: {

type: String

},

image: {

type: String

}

});

module.exports = mongoose.model('Course', courseSchema);

**Explanation:**

1. **Import Mongoose:**
   * const mongoose = require('mongoose'); imports the Mongoose library.
2. **Define the schema:**
   * const courseSchema = new mongoose.Schema({ ... }); creates a new Mongoose schema to define the structure of a course document.
   * The schema includes fields for:
     + name: The name of the course (required).
     + description: A brief description of the course (required).
     + duration: The duration of the course (required).
     + fee: The fee for the course (required).
     + prerequisites: Any prerequisites for the course (optional).
     + image: The URL of an image for the course (optional).
3. **Create the model:**
   * module.exports = mongoose.model('Course', courseSchema); creates a Mongoose model named "Course" based on the defined schema.

This model will be used by the courses routes to interact with the MongoDB database, such as creating new courses, retrieving existing courses, and updating course information.

**Remember to:**

* Create the models directory and save this file as Course.js.
* Import the Course model in your courses routes file.
* Use the model to interact with the database in your route handlers.

This is a basic example, and you can customize the schema to include additional fields as per your specific requirements.



code for models/user.js

JavaScript

const mongoose = require('mongoose');

const userSchema = new mongoose.Schema({

name: {

type: String,

required: true

},

email: {

type: String,

required: true,

unique: true

},

password: {

type: String,

required: true

},

role: {

type: String,

enum: ['admin', 'instructor', 'student'],

default: 'student'

}

});

module.exports = mongoose.model('User', userSchema);

**Explanation:**

1. **Import Mongoose:**
   * const mongoose = require('mongoose'); imports the Mongoose library.
2. **Define the schema:**
   * const userSchema = new mongoose.Schema({ ... }); creates a new Mongoose schema to define the structure of a user document.
   * The schema includes fields for:
     + name: The name of the user (required).
     + email: The email address of the user (required and unique).
     + password: The password of the user (required).
     + role: The role of the user (admin, instructor, or student), defaulting to 'student'.
3. **Create the model:**
   * module.exports = mongoose.model('User', userSchema); creates a Mongoose model named "User" based on the defined schema.

This model will be used to manage users in your application, including registration, login, and role-based access control.

**Remember to:**

* Create the models directory and save this file as User.js.
* Import the User model in your relevant routes and controllers.
* Implement proper password hashing and security measures to protect user data.
* Consider adding more fields to the schema as per your specific requirements, such as contact information, profile picture, etc.

This is a basic example, and you can customize the schema to fit the specific needs of your computer training institute application.



WRITE A CODE FOR ROUTES/CONTACT.JS , ABOUTUS.JS AND ADMIN.JS

Certainly, here are the code for routes/contact.js, routes/about.js, and routes/admin.js:

**routes/contact.js**

JavaScript

const express = require('express');

const router = express.Router();

const nodemailer = require('nodemailer'); // For sending emails

router.get('/', (req, res) => {

res.render('contact', { title: 'Contact Us' });

});

router.post('/', async (req, res) => {

const { name, email, message } = req.body;

try {

// Create a transporter for sending emails

const transporter = nodemailer.createTransport({

service: 'gmail', // Or your preferred email service

auth: {

user: 'your\_email@gmail.com',

pass: 'your\_email\_password'

}

});

// Define the email options

const mailOptions = {

from: email,

to: 'recipient\_email@example.com', // Replace with your recipient email

subject: `Message from ${name}`,

text: message

};

// Send the email

await transporter.sendMail(mailOptions);

res.render('contact', { title: 'Contact Us', success: true });

} catch (error) {

console.error(error);

res.status(500).send('Error sending email');

}

});

module.exports = router;

**routes/about.js**

JavaScript

const express = require('express');

const router = express.Router();

router.get('/', (req, res) => {

res.render('about', { title: 'About Us' });

});

module.exports = router;

**routes/admin.js**

JavaScript

const express = require('express');

const router = express.Router();

const authMiddleware = require('../middleware/auth'); // Assuming an auth middleware

// Protect admin routes with authentication middleware

router.use(authMiddleware);

// Admin dashboard route

router.get('/', (req, res) => {

res.render('admin/dashboard', { title: 'Admin Dashboard' });

});

// Route for managing courses (example)

router.get('/courses', (req, res) => {

// Logic to get a list of courses

res.render('admin/courses', { title: 'Manage Courses' });

});

// Add more routes for admin functionalities like adding/editing courses, managing users, etc.

module.exports = router;

**Explanation:**

**contact.js:**

* Handles GET and POST requests for the contact page.
* Renders the contact.ejs template for the GET request.
* On POST request, collects form data and sends an email using nodemailer.
* Renders the contact.ejs template with a success message if the email is sent successfully.

**about.js:**

* Handles the GET request for the /about route.
* Renders the about.ejs template.

**admin.js:**

* Uses the authMiddleware to protect all admin routes.
* Defines the dashboard route for the admin panel.
* Defines a route for managing courses (example).
* You can add more routes for other admin functionalities as needed.

**Important Notes:**

* **nodemailer:** Install the nodemailer package using npm install nodemailer.
* **authMiddleware:** Create an authMiddleware (in middleware/auth.js) to verify user authentication and authorization before accessing admin routes.
* **Email Configuration:** Replace your\_email@gmail.com and your\_email\_password with your actual Gmail credentials (or the credentials of your chosen email service).
* **Security:**
  + **Never hardcode sensitive information like email passwords directly in the code.** Use environment variables or a more secure method to store and access credentials.
  + Implement proper input validation and sanitization to prevent security vulnerabilities.
* **Error Handling:** Implement robust error handling to gracefully handle potential issues like email sending failures.

These routes provide a basic framework for your contact, about, and admin sections. You can customize them further to meet your specific requirements and add more advanced features.



write code : const authMiddleware = require('../middleware/auth'); // Assuming an auth middleware

JavaScript

// middleware/auth.js

const jwt = require('jsonwebtoken');

const authMiddleware = (req, res, next) => {

try {

const token = req.headers.authorization.split(' ')[1];

if (!token) {

return res.status(401).json({ message: 'Unauthorized' });

}

const decoded = jwt.verify(token, 'your\_secret\_key'); // Replace with your actual secret key

req.user = decoded;

next();

} catch (error) {

console.error(error);

return res.status(401).json({ message: 'Unauthorized' });

}

};

module.exports = authMiddleware;

**Explanation:**

1. **Import jsonwebtoken:**
   * const jwt = require('jsonwebtoken'); imports the jsonwebtoken library for working with JWTs.
2. **Define the authMiddleware function:**
   * This function takes req, res, and next as arguments.
   * It extracts the JWT token from the Authorization header in the request.
   * It verifies the token using jwt.verify() with your secret key.
   * If the token is valid, it attaches the decoded user information to the req.user object.
   * It calls next() to proceed to the next middleware or route handler.
   * If there's an error during token verification or if the token is missing, it returns an "Unauthorized" error.
3. **Export the middleware:**
   * module.exports = authMiddleware; exports the authMiddleware function to be used in other parts of your application.

**Key Points:**

* **Secret Key:**
  + Replace 'your\_secret\_key' with a strong, unique secret key.
  + This key is crucial for securely signing and verifying JWTs.
  + Keep your secret key confidential and never hardcode it directly in your code. Use environment variables or a secure secrets management solution.
* **Token Handling:**
  + The code assumes that the JWT is sent in the Authorization header with the format Bearer <token>.
  + Adjust this logic if your token is sent in a different manner.
* **Error Handling:**
  + The code includes basic error handling. You can customize it to provide more specific error messages or log errors to a file.
* **User Information:**
  + The decoded token typically contains user information such as ID, role, and other relevant data.
  + You can access this information in your route handlers using req.user.

**To use this middleware in your admin.js route:**

JavaScript

// routes/admin.js

const express = require('express');

const router = express.Router();

const authMiddleware = require('../middleware/auth');

router.use(authMiddleware);

// ... rest of your admin routes

This will ensure that all routes within the admin router require a valid JWT for access.

Remember to adapt this code to your specific needs and security requirements.



write code for courses.ejs

HTML

<!DOCTYPE html>

<html>

<head>

<title>Courses</title>

<link rel="stylesheet" href="style.css">

</head>

<body>

<header>

<h1>Our Courses</h1>

</header>

<main>

<ul class="course-list">

<% courses.forEach(course => { %>

<li>

<h2><%= course.name %></h2>

<p><%= course.description %></p>

<p><strong>Duration:</strong> <%= course.duration %></p>

<p><strong>Fee:</strong> <%= course.fee %></p>

<a href="/courses/<%= course.\_id %>" class="btn">View Details</a>

</li>

<% }); %>

</ul>

</main>

<footer>

<p>&copy; 2023 Computer Training Institute</p>

</footer>

</body>

</html>

**Explanation:**

* **Includes the necessary HTML elements:** head, body, header, main, and footer.
* **Uses EJS templating language:**
  + <% ... %> blocks are used for executing JavaScript code.
  + <%= ... %> blocks are used to output the values of JavaScript variables.
* **Iterates over courses:**
  + courses.forEach(course => { ... }); iterates over the array of courses passed from the courses route.
* **Displays course information:**
  + Displays the name, description, duration, and fee of each course.
  + Includes a link to the course details page using href="/courses/<%= course.\_id %>".

**To use this template:**

1. Save this code as courses.ejs in the views directory.
2. In the courses route, render this template by passing the array of courses to the res.render() method:

JavaScript

res.render('courses', { courses });

This will dynamically generate an HTML list of courses based on the data retrieved from the database.

**Note:**

* This is a basic example. You can customize it further by adding features like:
  + Course filtering and searching
  + Course sorting
  + Pagination
  + Images for each course
  + Enrollment buttons

Remember to adjust the CSS styles in style.css to match the desired appearance of the course list.



i think home page and navigation bar is missing

You're absolutely right! The provided code snippets are missing the crucial index.ejs file for the homepage and the navigation bar.

Here's an enhanced index.ejs with a basic navigation bar:

HTML

<!DOCTYPE html>

<html>

<head>

<title>Computer Training Institute</title>

<link rel="stylesheet" href="style.css">

</head>

<body>

<nav>

<ul>

<li><a href="/">Home</a></li>

<li><a href="/courses">Courses</a></li>

<li><a href="/about">About Us</a></li>

<li><a href="/contact">Contact</a></li>

<li><a href="/admin">Admin</a></li>

</ul>

</nav>

<header>

<h1>Welcome to Our Training Institute</h1>

</header>

<main>

<section class="hero">

<h2>Learn New Skills, Achieve Your Goals</h2>

<p>Discover a wide range of courses to enhance your career.</p>

<a href="/courses" class="btn">Explore Courses</a>

</section>

<section class="about">

<h2>About Us</h2>

<p>We are dedicated to providing high-quality training and education...</p>

<a href="/about" class="btn">Learn More</a>

</section>

<section class="contact">

<h2>Contact Us</h2>

<p>Have any questions? Get in touch with us.</p>

<a href="/contact" class="btn">Contact Us</a>

</section>

</main>

<footer>

<p>&copy; 2023 Computer Training Institute</p>

</footer>

</body>

</html>

**Explanation:**

* **Navigation Bar:**
  + A simple unordered list (<ul>) is used to create the navigation bar.
  + Each list item (<li>) contains an anchor tag (<a>) with the corresponding link (e.g., /courses, /about).
* **Home Page Content:**
  + The header, main, and footer sections are similar to the previous index.ejs but with minor adjustments for better presentation.
  + The hero section now includes a prominent call-to-action button to encourage users to explore the courses.

**Remember to:**

* **Style the navigation bar:** Add CSS styles to make the navigation bar visually appealing (e.g., background color, font color, spacing).
* **Consider user authentication:** If you have user authentication implemented, you might want to adjust the navigation bar to display different links for logged-in users (e.g., profile, logout).

This enhanced index.ejs provides a more complete and user-friendly homepage with a basic navigation bar.



rewrite the proper file structure according to our changes

Certainly, here's the revised file structure incorporating the changes we've made, including the navigation bar and improved index.ejs:

project-name/

├── public/

│ ├── css/

│ │ ├── style.css

│ │ └── ...

│ ├── images/

│ │ ├── logo.png

│ │ └── ...

│ ├── js/

│ │ ├── script.js

│ │ └── ...

│ └── index.html

├── routes/

│ ├── index.js

│ ├── courses.js

│ ├── about.js

│ ├── contact.js

│ ├── admin.js

│ └── auth.js

├── views/

│ ├── index.ejs

│ ├── courses.ejs

│ ├── course.ejs

│ ├── about.ejs

│ ├── contact.ejs

│ └── admin/

│ ├── dashboard.ejs

│ ├── courses.ejs

│ └── ...

├── models/

│ ├── Course.js

│ ├── User.js

│ └── ...

├── middleware/

│ ├── auth.js

├── config/

│ └── database.js

├── app.js

└── package.json

**Changes:**

* **middleware directory:** Added a middleware directory to hold the auth.js middleware file.
* **admin subdirectory in views:** Added a courses.ejs file within the admin subdirectory for managing courses in the admin panel.

This updated file structure better reflects the organization of our project and the inclusion of the authMiddleware.

I hope this revised structure is helpful!



write the js/script.js code

JavaScript

// js/script.js

// Example: Smooth scrolling to sections on click

const sections = document.querySelectorAll('section');

sections.forEach(section => {

const navLink = document.querySelector(`a[href="#${section.id}"]`);

navLink.addEventListener('click', (event) => {

event.preventDefault();

const targetId = section.getAttribute('id');

const targetPosition = section.offsetTop;

window.scrollTo({

top: targetPosition,

behavior: 'smooth'

});

});

});

// You can add more JavaScript functionality here, such as:

// - Interactive elements

// - Form validation

// - AJAX requests

// - Dynamic content updates

**Explanation:**

* **Smooth Scrolling:**
  + This code snippet demonstrates smooth scrolling to different sections of the page when their corresponding navigation links are clicked.
  + It iterates through each section on the page.
  + For each section, it finds the corresponding navigation link using its id.
  + When the link is clicked, it prevents the default behavior (page jump) and smoothly scrolls to the target section's position.
* **Further Enhancements:**
  + You can add more JavaScript functionality to your website as needed, such as:
    - Interactive elements (e.g., image sliders, carousels)
    - Form validation to ensure user input is correct
    - AJAX requests for fetching data from the server without full page reloads
    - Dynamically updating content based on user interactions

**Remember:**

* This is a basic example, and you can customize it to fit the specific needs of your computer training institute website.
* Include this script.js file in your index.html using a <script> tag:

HTML

<script src="js/script.js"></script>

This script.js file provides a starting point for adding interactive JavaScript functionality to your website.



write code for views/admin/daboard.ejs and courses.ejs

Certainly, here's the code for views/admin/dashboard.ejs and views/admin/courses.ejs:

**views/admin/dashboard.ejs**

HTML

<!DOCTYPE html>

<html>

<head>

<title>Admin Dashboard</title>

<link rel="stylesheet" href="/css/style.css">

</head>

<body>

<nav>

<ul>

<li><a href="/">Home</a></li>

<li><a href="/admin/dashboard">Dashboard</a></li>

<li><a href="/admin/courses">Courses</a></li>

</ul>

</nav>

<header>

<h1>Admin Dashboard</h1>

</header>

<main>

<h2>Welcome, Admin!</h2>

<p>This is the admin dashboard.</p>

<p>Here you can manage courses, students, and other administrative tasks.</p>

</main>

<footer>

<p>&copy; 2023 Computer Training Institute</p>

</footer>

</body>

</html>

**views/admin/courses.ejs**

HTML

<!DOCTYPE html>

<html>

<head>

<title>Manage Courses</title>

<link rel="stylesheet" href="/css/style.css">

</head>

<body>

<nav>

<ul>

<li><a href="/">Home</a></li>

<li><a href="/admin/dashboard">Dashboard</a></li>

<li><a href="/admin/courses">Courses</a></li>

</ul>

</nav>

<header>

<h1>Manage Courses</h1>

</header>

<main>

<h2>All Courses</h2>

<ul class="course-list">

<% courses.forEach(course => { %>

<li>

<h3><%= course.name %></h3>

<p><%= course.description %></p>

<a href="/admin/courses/<%= course.\_id %>/edit" class="btn">Edit</a>

<a href="/admin/courses/<%= course.\_id %>/delete" class="btn">Delete</a>

</li>

<% }); %>

</ul>

<a href="/admin/courses/new" class="btn">Add New Course</a>

</main>

<footer>

<p>&copy; 2023 Computer Training Institute</p>

</footer>

</body>

</html>

**Explanation:**

**dashboard.ejs:**

* Basic layout with a navigation bar, header, and main content.
* Displays a welcome message for the admin.
* Provides links to other admin sections (e.g., "Courses").

**courses.ejs:**

* Basic layout with a navigation bar, header, and main content.
* Displays a list of all courses.
* Includes links to edit and delete existing courses.
* Includes a link to add a new course.

**Remember to:**

* Create the corresponding routes and controllers for handling the "Add New Course," "Edit Course," and "Delete Course" functionalities.
* Implement proper input validation, error handling, and security measures for the admin panel.
* Customize the HTML and CSS to match your desired design and user interface.

These files provide a starting point for the admin dashboard and course management sections of your application. You can further enhance them with features like search, filtering, pagination, and more.



write code for config/database.js

JavaScript

// config/database.js

const mongoose = require('mongoose');

const dbURI = 'mongodb://localhost/training\_institute'; // Replace with your MongoDB URI

mongoose.connect(dbURI, { useNewUrlParser: true, useUnifiedTopology: true })

.then(() => console.log('MongoDB Connected'))

.catch(err => console.error(err));

module.exports = mongoose;

**Explanation:**

1. **Import Mongoose:**
   * const mongoose = require('mongoose'); imports the Mongoose library.
2. **Define MongoDB URI:**
   * const dbURI = 'mongodb://localhost/training\_institute'; defines the MongoDB URI.
     + Replace mongodb://localhost/training\_institute with your actual MongoDB connection string if you are using a different database or connection method (e.g., Atlas).
3. **Connect to MongoDB:**
   * mongoose.connect(dbURI, { ... }); establishes a connection to the MongoDB database using the provided URI.
     + The useNewUrlParser: true and useUnifiedTopology: true options are recommended for newer Mongoose versions.
4. **Handle connection events:**
   * The .then() block logs a success message if the connection is successful.
   * The .catch() block logs an error message if the connection fails.
5. **Export Mongoose:**
   * module.exports = mongoose; exports the Mongoose object, allowing other parts of the application to interact with the database.

**To use this file in your app.js:**

JavaScript

// app.js

const express = require('express');

const mongoose = require('./config/database'); // Import Mongoose from the config file

// ... rest of your app.js code ...

This database.js file centralizes your MongoDB connection logic, making it easier to manage and update.



write code for public/index.html

HTML

<!DOCTYPE html>

<html>

<head>

<title>Computer Training Institute</title>

<link rel="stylesheet" href="css/style.css">

</head>

<body>

<nav>

<ul>

<li><a href="/">Home</a></li>

<li><a href="/courses">Courses</a></li>

<li><a href="/about">About Us</a></li>

<li><a href="/contact">Contact</a></li>

<li><a href="/admin">Admin</a></li>

</ul>

</nav>

<header>

<h1>Welcome to Our Training Institute</h1>

</header>

<main>

<section class="hero">

<h2>Learn New Skills, Achieve Your Goals</h2>

<p>Discover a wide range of courses to enhance your career.</p>

<a href="/courses" class="btn">Explore Courses</a>

</section>

<section class="about">

<h2>About Us</h2>

<p>We are dedicated to providing high-quality training and education...</p>

<a href="/about" class="btn">Learn More</a>

</section>

<section class="contact">

<h2>Contact Us</h2>

<p>Have any questions? Get in touch with us.</p>

<a href="/contact" class="btn">Contact Us</a>

</section>

</main>

<footer>

<p>&copy; 2023 Computer Training Institute</p>

</footer>

<script src="js/script.js"></script>

</body>

</html>

**Explanation:**

* Includes the necessary HTML elements: head, body, header, main, and footer.
* Contains a basic navigation bar with links to different sections of the website.
* Includes sections for the hero, about us, and contact information.
* Includes a script tag to link the script.js file for any JavaScript functionality.

This index.html file serves as the main entry point for your website and provides the basic structure for the homepage. You can customize it further with images, videos, and other content to make it more visually appealing and engaging.

Remember to adjust the CSS styles in style.css to match your desired design and layout.



write the code for public/css/style.css

CSS

/\* public/css/style.css \*/

body {

font-family: sans-serif;

margin: 0;

padding: 0;

}

nav {

background-color: #333;

color: #fff;

overflow: hidden;

}

nav ul {

list-style: none;

margin: 0;

padding: 0;

}

nav li {

display: inline-block;

margin-right: 20px;

}

nav a {

color: #fff;

text-decoration: none;

padding: 15px 20px;

display: block;

}

nav a:hover {

background-color: #555;

}

header {

background-color: #f0f0f0;

text-align: center;

padding: 20px;

}

main {

padding: 20px;

}

section {

margin-bottom: 30px;

}

.hero {

background-color: #eee;

padding: 30px;

border-radius: 5px;

}

.btn {

background-color: #333;

color: #fff;

padding: 10px 20px;

border-radius: 5px;

text-decoration: none;

}

.btn:hover {

background-color: #555;

}

footer {

background-color: #333;

color: #fff;

text-align: center;

padding: 10px;

}

.course-list {

list-style: none;

padding: 0;

}

.course-list li {

margin-bottom: 20px;

padding: 15px;

border: 1px solid #ddd;

border-radius: 5px;

}

**Explanation:**

* **Basic Styling:**
  + Sets basic styles for the body element (font, margins, padding).
* **Navigation Bar:**
  + Styles the navigation bar with a dark background color and white text.
  + Styles the navigation links with appropriate spacing and hover effects.
* **Header:**
  + Styles the header section with a light background color and centered text.
* **Main Content:**
  + Adds padding to the main content area.
* **Sections:**
  + Adds spacing between sections.
* **Hero Section:**
  + Styles the hero section with a light background color and rounded corners.
* **Button:**
  + Styles the buttons with a dark background color, white text, and rounded corners.
* **Footer:**
  + Styles the footer with a dark background color and white text.
* **Course List:**
  + Styles the list of courses with appropriate spacing and borders.

**Remember:**

* This is a basic CSS style sheet. You can customize it further to match your desired design and branding.
* You can use CSS frameworks like Bootstrap or Materialize for more advanced styling and responsive design.

This style.css file provides a basic foundation for the visual appearance of your computer training institute website.



css file work ni kar rhi ya kuch aur issue hai

The provided CSS file (style.css) should be working correctly. However, if you're encountering issues where your styles aren't being applied, here are some common troubleshooting steps:

**1. Check File Paths and Linking:**

* **Correct Path:** Ensure the href attribute in your index.html file points to the correct location of your style.css file.
  + If style.css is in the same directory as index.html, the correct link would be:

HTML

<link rel="stylesheet" href="css/style.css">

* **File Existence:** Double-check that the style.css file exists in the public/css/ directory and that the file name is spelled correctly.
* **Case Sensitivity:** File and directory names are case-sensitive in some operating systems. Make sure the names in your HTML href attribute match the actual file and directory names exactly (including capitalization).

**2. Browser Cache:**

* **Clear Browser Cache:** Your browser might be caching an older version of the CSS file. Try clearing your browser's cache and hard reloading the page (usually by pressing **Ctrl + Shift + R** or **Cmd + Shift + R**).

**3. CSS File Errors:**

* **Syntax Errors:** Carefully review your style.css file for any syntax errors (e.g., missing semicolons, mismatched brackets). Even minor errors can prevent the CSS from being applied correctly.
* **Selector Specificity:** If you're using more specific selectors (e.g., #id, .class), make sure they are correctly targeting the HTML elements you want to style. More specific selectors have higher precedence.

**4. Browser Developer Tools:**

* **Inspect Element:** Use your browser's developer tools (usually accessible by right-clicking on an element and selecting "Inspect" or by pressing **F12**) to inspect the HTML and CSS.
  + Check if the CSS file is being loaded correctly in the "Network" or "Sources" tab.
  + Check the "Computed" tab to see which CSS rules are actually being applied to the element.
  + Look for any browser console errors that might be related to the CSS.

**5. Test in a Different Browser:**

* Try testing your website in a different browser (e.g., Chrome, Firefox, Edge) to see if the issue is specific to your current browser.

**Example of how to inspect in Chrome DevTools:**

1. Right-click on an element on the page that should be styled by the CSS.
2. Select "Inspect" from the context menu.
3. In the Elements panel, select the element you clicked on.
4. In the "Styles" tab, you can see the applied CSS rules and make changes to see how they affect the element.

If you've checked these points and are still having trouble, please provide the following information so I can assist you further:

* **Relevant code snippets:** The HTML code that includes the <link> tag for the CSS file, and the specific CSS rules that are not working.
* **Error messages:** Any error messages that appear in the browser console.
* **Browser and operating system:** The browser and operating system you are using.

I'm here to help you troubleshoot and get your CSS working correctly!