**Execution steps**

1. cd E:\main\web\trade-trek\_project\backend

PS E:\main\web\trade-trek\_project\backend> node server.js

1. cd E:\main\web\trade-trek\_project\frontend

PS E:\main\web\trade-trek\_project\frontend> npm start

mern-tradespeople-app/

├── backend/ # Backend (Node.js + Express)

│ ├── config/ # Configuration files

│ │ ├── db.js # MongoDB connection setup

│ │ └── default.json # Default configuration (e.g., JWT secret, database URI)

│ ├── controllers/ # Controller functions for handling requests

│ │ ├── authController.js # Authentication (sign-up, login)

│ │ ├── userController.js # User-related functions

│ │ └── tradespersonController.js # Tradesperson-related functions

│ ├── models/ # Database schemas and models

│ │ ├── User.js # User model

│ │ ├── Tradesperson.js # Tradesperson model

│ │ ├── Booking.js # Booking model

│ │ └── Review.js # Review model

│ ├── routes/ # API route definitions

│ │ ├── authRoutes.js # Authentication routes

│ │ ├── userRoutes.js # User-specific routes

│ │ ├── tradespersonRoutes.js # Tradesperson-specific routes

│ │ └── bookingRoutes.js # Booking-related routes

│ ├── middlewares/ # Middleware functions

│ │ ├── authMiddleware.js # JWT verification, role checks

│ │ └── errorHandler.js # Error handling middleware

│ ├── utils/ # Utility functions

│ │ └── emailService.js # Functions for sending emails

│ ├── app.js # Express app setup

│ └── server.js # Server entry point

├── frontend/ # Frontend (React)

│ ├── public/ # Static assets

│ │ ├── index.html # HTML template

│ │ └── favicon.ico # Site favicon

│ ├── src/ # React application source

│ │ ├── components/ # Reusable components

│ │ │ ├── Navbar.js # Navbar component

│ │ │ ├── Footer.js # Footer component

│ │ │ ├── ServiceCard.js # Card component for service categories

│ │ │ └── Testimonial.js # Testimonial component

│ │ ├── pages/ # Page components

│ │ │ ├── Home.js # Home page (landing page)

│ │ │ ├── SignUp.js # Sign-up page with role selection

│ │ │ ├── Login.js # Login page

│ │ │ ├── UserDashboard.js # Dashboard for users

│ │ │ └── TradespersonDashboard.js # Dashboard for tradespeople

│ │ ├── services/ # API service files for data fetching

│ │ │ ├── authService.js # Auth-related API calls

│ │ │ ├── userService.js # User-related API calls

│ │ │ └── bookingService.js # Booking-related API calls

│ │ ├── contexts/ # Context providers for state management

│ │ │ ├── AuthContext.js # Context for authentication

│ │ │ └── UserContext.js # Context for user data

│ │ ├── hooks/ # Custom React hooks

│ │ │ └── useAuth.js # Hook for authentication logic

│ │ ├── App.js # Main React component

│ │ ├── index.js # Entry point for React

│ │ ├── styles/ # CSS and styling files

│ │ │ └── App.css # Global styles

│ │ └── tailwind.config.js # Tailwind CSS configuration (if using Tailwind)

├── .env # Environment variables (e.g., MongoDB URI, JWT secret)

├── .gitignore # Git ignore file

├── package.json # Project metadata and dependencies for backend

├── frontend/package.json # Project metadata and dependencies for frontend

├── README.md # Project documentation

└── Procfile # Heroku deployment file (if deploying to Heroku)

**Initial Homepage View for All Visitors (Users and Tradespeople)**

1. **Hero Section**:
   * **Main Headline**: A welcoming and clear tagline, like “Find Skilled Tradespeople Near You” or “Connecting You with Local Experts for Small Repairs,” which immediately communicates the website’s purpose.
   * **Search Bar**: A prominently placed search bar with placeholders such as “What service do you need?” and “Enter your location.” This allows visitors to start exploring right away by typing in a service (e.g., “Plumber”) and a location.
   * **Browse Services Button**: A quick “Browse Services” button below the search bar, allowing users to see available categories without entering specific search terms.
2. **Service Categories**:
   * **Popular Services Grid**: A visually appealing grid or icons representing popular service categories like “Plumbing,” “Electrical,” “Carpentry,” “Painting,” etc. Clicking on these categories will take visitors to a list of tradespeople offering those services.
   * **Hover Descriptions**: Each icon or tile may include a brief description when hovered over, such as “Qualified plumbers near you for quick fixes and repairs.”
3. **How It Works Section**:
   * **Simple Steps**: A brief, three-step guide explaining how the website works, which helps both users and tradespeople understand the process. For example:
     1. **Search for Services** – Find local professionals by searching by service and location.
     2. **Choose and Connect** – View profiles, read reviews, and reach out to the tradesperson that fits your needs.
     3. **Get the Job Done** – Book the service, leave a review, and complete the job!
   * **Visuals or Icons**: Each step could be accompanied by small icons or illustrations for a quick and engaging visual explanation.
4. **Benefits Section**:
   * **Why Choose Us?**: A section highlighting the platform’s benefits for both users and tradespeople. Examples might include:
     1. **For Users**: Easy access to trusted local professionals, verified profiles, and user reviews.
     2. **For Tradespeople**: A platform to grow their business, reach local clients, and showcase their expertise.
   * **Trust Indicators**: Icons or short statements emphasizing trust and quality, like “Verified Professionals,” “Transparent Reviews,” and “Easy Booking.”
5. **Testimonials or Featured Reviews**:
   * **User Feedback**: A carousel or static grid with a few testimonials from users who have successfully booked services and tradespeople who have grown their client base. This builds trust for new visitors and shows the platform’s value.
   * **High Ratings**: If applicable, showcase an average rating (e.g., “4.8/5 stars”) to highlight positive feedback from previous users.
6. **Call to Action for Sign Up**:
   * **Sign-Up Prompts**: Buttons encouraging both users and tradespeople to sign up, such as “Get Started as a User” and “Join as a Tradesperson.” These buttons could take visitors to a sign-up page where they choose their role.
   * **Benefits of Joining**: Small descriptions below each button explain the advantages of creating an account (e.g., “Find and save favorite tradespeople” for users, “List your services and connect with clients” for tradespeople).
7. **FAQ or Information Section**:
   * **Common Questions**: A FAQ dropdown or brief section addressing basic questions like “How do I book a service?” or “What happens after I request a service?”
   * **Additional Links**: Links to pages like “About Us,” “Contact Us,” and “Safety & Trust,” allowing visitors to learn more about the platform if they’re curious.
8. **Footer with Links**:
   * **Quick Links**: A footer at the bottom of the page with quick links to other parts of the site, such as “About,” “How It Works,” “FAQ,” “Privacy Policy,” and “Terms of Service.”
   * **Social Media Icons**: Icons linking to the platform’s social media pages, if applicable, to establish online presence and credibility.
   * **Contact Information**: A simple contact form or email link for inquiries and support.

Day 1: Project Setup Backend Setup: Initialize the backend with Node.js, Express, and MongoDB setup. Create the initial directory structure for backend. Configure MongoDB connection (config/db.js) and set up environment variables (.env). Set up basic Express server (server.js) and a health check route to confirm server setup. Frontend Setup: Initialize the React project inside the frontend directory. Set up basic directory structure and Tailwind CSS (if used). Configure basic routing using react-router-dom and create placeholder pages (Home, SignUp, Login).

**Here’s a detailed guide to help you complete Day 1: Project Setup for both the backend and frontend:**

**Backend Setup**

**1. Initialize the Backend**

1. **Create a new directory for the backend:**

mkdir backend && cd backend

1. **Initialize a Node.js project:**

npm init -y

1. **Install necessary packages:**

npm install express mongoose dotenv cors nodemon

**2. Create Directory Structure**

Inside the backend directory, organize the folders like this:

backend/

├── config/

│ └── db.js

├── models/

├── routes/

├── controllers/

├── middleware/

├── .env

├── server.js

└── package.json

**3. Set Up MongoDB Connection**

**config/db.js**

This file handles the MongoDB connection.

const mongoose = require('mongoose');

const connectDB = async () => {

try {

const conn = await mongoose.connect(process.env.MONGO\_URI, {

useNewUrlParser: true,

useUnifiedTopology: true,

});

console.log(`MongoDB Connected: ${conn.connection.host}`);

} catch (error) {

console.error(`Error: ${error.message}`);

process.exit(1); // Exit process with failure

}

};

module.exports = connectDB;

**4. Set Up Environment Variables**

Create a .env file in the root of the backend directory:

MONGO\_URI=your\_mongodb\_connection\_string

PORT=5000

**5. Set Up Express Server**

**server.js**

This file initializes the server and connects to the database.

const express = require('express');

const dotenv = require('dotenv');

const cors = require('cors');

const connectDB = require('./config/db');

// Load environment variables

dotenv.config();

// Initialize Express app

const app = express();

// Middleware

app.use(express.json());

app.use(cors());

// Database Connection

connectDB();

// Health Check Route

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

res.status(200).send('API is running...');

});

// Start the server

const PORT = process.env.PORT || 5000;

app.listen(PORT, () => {

console.log(`Server running on port ${PORT}`);

});

**6. Test the Backend**

1. Run the server using nodemon:

npx nodemon server.js

**Frontend Setup**

**1. Initialize React Project**

1. Create a new directory for the frontend:

mkdir frontend && cd frontend

1. Initialize a React project:

npx create-react-app .

1. Install react-router-dom for routing:

npm install react-router-dom

1. If using Tailwind CSS, install it:

npm install -D tailwindcss postcss autoprefixer

npx tailwindcss init

**2. Set Up Tailwind CSS (Optional)**

**tailwind.config.js**

Update the Tailwind configuration file to enable all templates:

/\*\* @type {import('tailwindcss').Config} \*/

module.exports = {

content: ["./src/\*\*/\*.{js,jsx,ts,tsx}"],

theme: {

extend: {},

},

plugins: [],

};

**src/index.css**

Replace the content with:

@tailwind base;

@tailwind components;

@tailwind utilities;

**3. Create Directory Structure**

Organize the frontend folders like this:

frontend/

├── src/

│ ├── components/

│ │ ├── Navbar.js

│ │ ├── Footer.js

│ ├── pages/

│ │ ├── Home.js

│ │ ├── SignUp.js

│ │ ├── Login.js

│ ├── App.js

│ ├── index.css

│ ├── index.js

└── package.json

**4. Set Up Basic Routing**

**App.js**

Set up routes using react-router-dom:

import React from 'react';

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

import Navbar from './components/Navbar';

import Footer from './components/Footer';

import Home from './pages/Home';

import SignUp from './pages/SignUp';

import Login from './pages/Login';

const App = () => {

return (

<Router>

<div className="flex flex-col min-h-screen">

<Navbar />

<main className="flex-grow">

<Routes>

<Route path="/" element={<Home />} />

<Route path="/signup" element={<SignUp />} />

<Route path="/login" element={<Login />} />

</Routes>

</main>

<Footer />

</div>

</Router>

);

};

export default App;

**5. Create Placeholder Pages**

**pages/Home.js**

import React from 'react';

const Home = () => {

return (

<div className="text-center mt-16">

<h1 className="text-4xl font-bold">Welcome to Our Platform</h1>

<p className="mt-4 text-lg">Find skilled tradespeople for your needs!</p>

</div>

);

};

export default Home;

**pages/Login.js**

import React from 'react';

const Login = () => {

return (

<div className="text-center mt-16">

<h1 className="text-3xl font-bold">Login Page</h1>

<p>Login form will go here.</p>

</div>

);

};

export default Login;

**6. Test the Frontend**

1. Start the React development server:

npm start

Day 2: User and Tradesperson Models & Authentication Backend: Create User and Tradesperson models with basic schema fields. Set up authentication routes in authRoutes.js for sign-up, login, and password reset. Implement JWT-based authentication and add role-based fields (User, Tradesperson). Add authController.js with signup, login, and JWT token generation functions. Frontend: Design simple Sign-Up and Login pages with forms and validation. Set up basic form handling and submit data to backend API

**Backend:**

1. **Create User and Tradesperson Models:**
   * **User Model (models/userModel.js):**

const mongoose = require('mongoose');

const userSchema = mongoose.Schema({

firstName: { type: String, required: true },

lastName: { type: String, required: true },

email: { type: String, required: true, unique: true },

password: { type: String, required: true },

userType: { type: String, enum: ['user', 'tradesperson'], required: true },

});

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

**Tradesperson Model (models/tradespersonModel.js):**

const mongoose = require('mongoose');

const tradespersonSchema = mongoose.Schema({

firstName: { type: String, required: true },

lastName: { type: String, required: true },

email: { type: String, required: true, unique: true },

phone: { type: String, required: true },

expertise: { type: String, required: true },

userType: { type: String, enum: ['user', 'tradesperson'], required: true },

});

module.exports = mongoose.model('Tradesperson', tradespersonSchema);

**Set Up Authentication Routes (routes/authRoutes.js):**

* Add routes for user authentication (sign-up, login).

const express = require('express');

const authController = require('../controllers/authController');

const router = express.Router();

router.post('/signup', authController.signUp);

router.post('/login', authController.login);

module.exports = router;

**Implement Authentication Logic (controllers/authController.js):**

* **SignUp and Login Functions with JWT**:

const bcrypt = require('bcryptjs');

const jwt = require('jsonwebtoken');

const User = require('../models/userModel');

const Tradesperson = require('../models/tradespersonModel');

// Sign-up function

const signUp = async (req, res) => {

const { firstName, lastName, email, password, userType } = req.body;

try {

const userExists = await User.findOne({ email });

if (userExists) return res.status(400).json({ message: 'User already exists' });

const hashedPassword = await bcrypt.hash(password, 10);

let newUser;

if (userType === 'user') {

newUser = new User({ firstName, lastName, email, password: hashedPassword, userType });

} else if (userType === 'tradesperson') {

newUser = new Tradesperson({ firstName, lastName, email, password: hashedPassword, userType });

}

const user = await newUser.save();

const token = jwt.sign({ id: user.\_id, userType: user.userType }, process.env.JWT\_SECRET, { expiresIn: '1h' });

res.status(201).json({ message: 'User created', token });

} catch (error) {

res.status(500).json({ message: error.message });

}

};

// Login function

const login = async (req, res) => {

const { email, password } = req.body;

try {

const user = await User.findOne({ email });

if (!user) return res.status(400).json({ message: 'User not found' });

const isMatch = await bcrypt.compare(password, user.password);

if (!isMatch) return res.status(400).json({ message: 'Invalid credentials' });

const token = jwt.sign({ id: user.\_id, userType: user.userType }, process.env.JWT\_SECRET, { expiresIn: '1h' });

res.status(200).json({ message: 'Login successful', token });

} catch (error) {

res.status(500).json({ message: error.message });

}

};

module.exports = { signUp, login };