Project RoadMap:

# CareerCraft - AI-Powered Career Building Assistant

# 1. Project Overview

**Title**: CareerCraft - AI-Powered Career Building Assistant

**Goal**: Build a fully responsive web platform that empowers students, graduates, and professionals by using AI tools to shape and support their career journey.

# 2. Real-World Problem

Many job seekers, especially in underrepresented regions, face confusion and lack of proper guidance when making career decisions. They often struggle with resume creation, skill development, and understanding career paths. CareerCraft aims to solve this with an intelligent, step-by-step AI-powered solution.

# 3. Key Features / Objectives

* **Responsive Web Platform** adaptable to all devices.
* **AI Resume Builder**: Generate professional resumes from raw user data.
* **Career Path Recommender**: Suggest potential job roles and growth paths.
* **Skill Gap Analyzer**: Analyze current skills and suggest improvements.
* **Mock Interview Simulator**: Role-based AI interviews for practice.
* **Portfolio Builder**: Allow users to build a live, shareable digital portfolio.
* **Progress Tracker**: Monitor learning, job applications, and interviews.

**4. Website Page Structure & Features**

# 1. Home Page

* Hero section with tagline & CTA (Get Started)
* Brief about CareerCraft features
* Testimonials or success stories
* Navigation to all major sections

# 2. About Us Page

* Project vision & mission
* Problem we're solving
* Team or contributors info

# 3. Resume Builder Page

* Form to collect raw user data (education, experience, skills)
* AI-based resume generation preview
* Downloadable or editable format

# 4. Career Recommender Page

* Input user goals/interests
* AI suggests possible career paths
* Shows required skills & market trends

# 5. Skill Gap Analyzer Page

* Input user skills
* Compare with selected job roles
* Suggest personalized learning roadmap

# 6. Mock Interview Page

* Select job role/domain
* Simulated AI questions & answers
* Feedback and scoring system

# 7. Portfolio Builder Page

* Personal dashboard to add projects, achievements
* Auto-design portfolio template options
* Export/share live link

# 8. Progress Tracker Page

* Visual display of user's growth
* Tracks job applications, skills, interview attempts

# 9. Contact Us Page

* Form for feedback, support
* Social/media links

# 10. Login / Signup Page

* Email/social login
* Onboarding steps for new users

# 11. Dashboard (Post-login)

* Central panel for accessing tools (Resume, Career Path, Portfolio, etc.)
* Notifications and AI suggestions

**5. Technologies to be Used (Frontend Only)**

* **React 19** (Vite setup)
* **Tailwind CSS** for styling
* **React Router** for navigation

# • Styled Components (if required)

• **LocalStorage** for saving progress/data

# 6. Non-Functional Requirements

* Fully responsive design (mobile-first)
* Smooth user experience and navigation
* Clear UI/UX for users with no technical background
* Modular and reusable components

# 7. Future Scope (Optional)

* Backend integration with Node.js & MongoDB
* AI models using OpenAI or similar services
* User authentication & data persistence
* Admin dashboard to manage user data

**✅ PART 1: How will I handle AI integration as a Frontend-only Developer?**

Since you're not building the backend or AI logic, you will **simulate** AI behavior during development.

**💡 Strategy: Fake AI Responses for Now**

You'll use **dummy data** and **mock responses** from fake APIs to simulate how AI would work.

| **AI Feature** | **How to Simulate in Frontend** |
| --- | --- |
| **Resume Generator** | When user submits data, you show a pre-written, AI-enhanced resume preview (from static JSON). |
| **Skill Gap Analyzer** | Use a fake job description + static skill list → show what skills are "missing" based on simple logic. |
| **Interview Simulator** | Load dummy Q&A pairs and randomly generate "AI feedback". |
| **Career Roadmap** | Show pre-defined skills and future milestones for selected roles. |
| **Portfolio Builder** | Just show a user-filled profile page (AI won't be used here heavily). |
| **Weekly Tracker** | Show dummy graphs with random progress data. |
| **Admin Panel** | Use static data to show users or insights. |

👉 In real apps, these would come from AI services like OpenAI, Gemini, or your custom backend.

**🧠 Tools for Mocking AI APIs:**

| **Tool** | **Usage** |
| --- | --- |
| msw (Mock Service Worker) | Fake backend calls to simulate AI responses |
| json-server | Run a fake local API server with your dummy data |
| fetch or axios | Use these to connect to mocked endpoints (e.g., /api/generate-resume) |
| Zustand or Context | To manage the mock data across multiple pages |

**✅ PART 2: What should I check at the end to confirm my app works?**

Here’s a checklist to verify your **frontend-only app is functional and complete**:

**✅ 1. UI Functionality**

* All buttons, forms, toggles, tabs work as expected.
* All screens (Resume, Interview, Portfolio, etc.) are **navigable and responsive**.

**✅ 2. Routing**

* Navigation between pages works via React Router.
* There is a 404 or fallback route.

**✅ 3. State Management**

* Zustand stores are used for: auth, resume data, filters, and toggles.
* State persists during user interaction (and optionally via localStorage).

**✅ 4. Mock Data Simulation**

* Dummy API data is loaded into all relevant components.
* AI features show realistic output (even if not real).

**✅ 5. Mobile Responsiveness**

* App looks good on desktop **and** mobile screens.

**✅ 6. Dark Mode / UI Settings**

* If included, dark mode or theme toggle works across pages.

**✅ 7. Deployment Readiness**

* App is clean of console errors.
* No unused files/components.
* GitHub repo is clean with README.
* App can be deployed to Vercel or Netlify for testing.

**✅ Updated Feature Roadmap (Frontend Only)**

Here’s the **final roadmap**, now including **real AI integration** (from frontend) and **Dark Mode support**:

**🧠 Core Features (Frontend)**

| **Feature** | **Description** | **AI Involved?** |
| --- | --- | --- |
| ✅ Resume Generator | User enters data, AI rewrites/beautifies it | ✅ OpenAI API |
| ✅ Skill Gap Analyzer | Compare user skills vs job description | ✅ OpenAI API |
| ✅ Interview Simulator | Role-based Q&A, AI gives questions + feedback | ✅ OpenAI API |
| ✅ Career Roadmap | AI suggests future goals + milestones | ✅ OpenAI API |
| ✅ Portfolio Builder | Public user profile with achievements | ❌ |
| ✅ Weekly Career Tracker | Charts showing progress (skills, goals) | ❌ |
| ✅ Dark Mode Toggle | Theme toggle across site (light/dark) | ❌ |
| ✅ Admin Panel (if time) | View users, insights (dummy frontend only) | ❌ |

**⚙️ Tech Additions for These New Requirements:**

**🧠 AI Integration (Frontend-only)**

You’ll **use OpenAI API** directly in frontend **but securely**:

| **What to Do** | **Tool** |
| --- | --- |
| Use fetch/axios to send prompts to OpenAI | fetch() |
| Keep API Key hidden during dev by .env (Vite) | .env |
| [Optional for deployment] Use a mock server or Vercel Edge Functions later | Vercel |

**Mock first, then switch to real OpenAI when tested.**  
I’ll help you simulate first → then do real AI.

**🌙 Dark Mode Setup**

Use:

* zustand or localStorage to manage dark mode preference
* tailwind.config.js to enable dark mode:

js

CopyEdit

// tailwind.config.js

module.exports = {

darkMode: 'class', // or 'media'

// ...

}

Create a toggle button:

jsx

CopyEdit

<button onClick={toggleDarkMode}>Toggle Theme</button>

Apply dark styles using:

html

CopyEdit

<div className="bg-white text-black dark:bg-gray-900 dark:text-white">

**🧠 Where You’ll Use AI (Keep This Tracked)**

| **Page/Feature** | **AI Needed** | **API Call** |
| --- | --- | --- |
| Resume Page | Yes | Send user's data as prompt |
| Skill Analyzer | Yes | Send skills + job description |
| Interview Simulator | Yes | Send role + questions |
| Career Roadmap | Yes | Send current skills, get future steps |
| Portfolio Page | No | Show saved data only |
| Tracker Charts | No | Visuals based on local data |
| Admin Panel | No | Static/dummy data only |
| Dark Mode | No | Zustand/localStorage only |

**🔁 Final Updated Tech Stack (Frontend Only)**

| **Area** | **Tool** |
| --- | --- |
| UI | React 19 + Tailwind CSS + styled-components |
| State Management | Zustand |
| Routing | React Router |
| Notifications | react-toastify |
| AI | OpenAI API (via fetch) |
| Theming | Tailwind + Zustand + localStorage |
| Storage | localStorage (for bugs, profile, filters, etc.) |
| Charts | Recharts or Chart.js |
| Code Quality | ESLint + Prettier (optional) |

**✅ Summary of Your Next Steps**

1. ✅ Finalize all UI screens in Figma
2. ✅ Build React components for each feature
3. ✅ Set up Zustand for state: dark mode, profile, etc.
4. ✅ Mock AI responses → test UI
5. ✅ Add real OpenAI API call in one feature (e.g., resume)
6. ✅ Add dark mode toggle
7. ✅ Connect localStorage to persist theme and user data

**✅ Where Will You Use TanStack Query in CareerCraft?**

Since you're simulating AI/backend with **mock data**, TanStack Query will help you organize and simulate real-time behavior like fetching from APIs.

**🧠 Use Cases in Your Project:**

| **📄 Page** | **💬 What to Fetch** | **📦 From Where** | **📌 Query Name** |
| --- | --- | --- | --- |
| Resume Builder | Pre-filled user data or template preview | JSON or local mock API | useResumeData |
| Skill Gap Analyzer | Job description or role data | JSON or mock endpoint | useJobData |
| Interview Simulator | Role-based Q&A pairs | JSON or mock API | useInterviewQA |
| Career Roadmap | Skill steps based on role | JSON or mock | useCareerMap |
| Progress Tracker | Weekly data for charts | JSON or static | useProgressData |
| Admin Panel (Optional) | List of users, insights | JSON/static | useAdminStats |

**✅ How Will You Use TanStack Query?**

Here’s how to structure usage in your frontend project:

**1. 🔧 Install TanStack Query**

bash

CopyEdit

npm install @tanstack/react-query

**2. 🧠 Setup the QueryClient in main.jsx**

jsx

CopyEdit

import { QueryClient, QueryClientProvider } from '@tanstack/react-query';

const queryClient = new QueryClient();

<React.StrictMode>

<QueryClientProvider client={queryClient}>

<App />

</QueryClientProvider>

</React.StrictMode>

**3. 📦 Create a Hook for Each Feature**

Use useQuery to fetch mock data (can be JSON files or hardcoded in a function).

**Example (Interview Q&A):**

jsx

CopyEdit

import { useQuery } from '@tanstack/react-query';

const fetchInterviewData = async () => {

const res = await fetch('/mock/interview-qa.json'); // local file or endpoint

return res.json();

};

export const useInterviewQA = () => {

return useQuery(['interviewData'], fetchInterviewData);

};

Now in your component:

jsx

CopyEdit

const { data, isLoading, isError } = useInterviewQA();

if (isLoading) return <p>Loading...</p>;

if (isError) return <p>Error loading Q&A</p>;

return (

<div>

{data.questions.map(q => (

<p key={q.id}>{q.question}</p>

))}

</div>

);

**✅ Where to Store Mock Data?**

* Place mock JSON in /public/mock/ or in a folder like src/data/mock/
* Alternatively, simulate delays with setTimeout in your fake fetch functions

**✅ Summary: TanStack Query Use in Your Frontend**

| **Feature** | **Hook** | **Fetches** | **Benefit** |
| --- | --- | --- | --- |
| Resume Builder | useResumeData | Templates, previews | Simulates AI preview |
| Interview | useInterviewQA | Questions & answers | Dynamic Q&A |
| Skill Gap | useJobData | Job roles, required skills | Show gaps |
| Career Map | useCareerMap | Suggested steps | Simulate AI path |
| Tracker | useProgressData | Weekly dummy data | Feed Chart.js |
| Admin | useAdminStats | Dummy user list | Simulate control panel |