**ACTIONS FOLDER:**

1. Actions/cover-letter.js:

"use server";

import { db } from "@/lib/prisma";

import { auth } from "@clerk/nextjs/server";

import { GoogleGenerativeAI } from "@google/generative-ai";

// Ensure the API key is set

if (!process.env.GEMINI\_API\_KEY) {

  throw new Error("GEMINI\_API\_KEY is not configured");

}

const genAI = new GoogleGenerativeAI(process.env.GEMINI\_API\_KEY);

const model = genAI.getGenerativeModel({ model: "gemini-1.5-flash" });

export async function generateCoverLetter(data) {

  const { userId } = await auth();

  if (!userId) throw new Error("Unauthorized");

  const user = await db.user.findUnique({

    where: { clerkUserId: userId },

  });

  if (!user) throw new Error("User not found");

  const prompt = `

    Write a professional cover letter for a ${data.jobTitle} position at ${data.companyName}.

    About the candidate:

    - Industry: ${user.industry}

    - Years of Experience: ${user.experience}

    - Skills: ${user.skills?.join(", ")}

    - Professional Background: ${user.bio}

    Job Description:

    ${data.jobDescription}

    Requirements:

    1. Use a professional, enthusiastic tone

    2. Highlight relevant skills and experience

    3. Show understanding of the company's needs

    4. Keep it concise (max 400 words)

    5. Use proper business letter formatting in markdown

    6. Include specific examples of achievements

    7. Relate candidate's background to job requirements

    Format the letter in markdown.

  `;

  try {

    const result = await model.generateContent(prompt);

    console.log("Generative AI full response:", result);

    // Extract text from the response

    const content = result.response.text().trim();

    if (!content || content === "$@1") {

      console.error("Received invalid content:", result);

      throw new Error("Generated content is empty or invalid");

    }

    // Save the generated cover letter to the database.

    // Removed "status" from the create call so the default value is used.

    const coverLetter = await db.coverLetter.create({

      data: {

        userId: user.id,

        content,

        jobDescription: data.jobDescription,

        companyName: data.companyName,

        jobTitle: data.jobTitle,

      },

    });

    return coverLetter;

  } catch (error) {

    console.error("Error generating cover letter:", error.message);

    throw new Error("Failed to generate cover letter" + error.message);

  }

}

export async function getCoverLetters() {

  const { userId } = await auth();

  if (!userId) throw new Error("Unauthorized");

  const user = await db.user.findUnique({

    where: { clerkUserId: userId },

  });

  if (!user) throw new Error("User not found");

  return await db.coverLetter.findMany({

    where: { userId: user.id },

    orderBy: { createdAt: "desc" },

  });

}

export async function getCoverLetter(id) {

  const { userId } = await auth();

  if (!userId) throw new Error("Unauthorized");

  const user = await db.user.findUnique({

    where: { clerkUserId: userId },

  });

  if (!user) throw new Error("User not found");

  return await db.coverLetter.findUnique({

    where: { id, userId: user.id },

  });

}

export async function deleteCoverLetter(id) {

  const { userId } = await auth();

  if (!userId) throw new Error("Unauthorized");

  const user = await db.user.findUnique({

    where: { clerkUserId: userId },

  });

  if (!user) throw new Error("User not found");

  return await db.coverLetter.delete({

    where: { id, userId: user.id },

  });

}

1. ACTIONS/dashboard.js:

"use server";

import { db } from "@/lib/prisma";

import { auth } from "@clerk/nextjs/server";

import { GoogleGenerativeAI } from "@google/generative-ai";

import { redirect } from "next/navigation"; // Import the redirect helper

const genAI = new GoogleGenerativeAI(process.env.GEMINI\_API\_KEY);

const model = genAI.getGenerativeModel({ model: "gemini-1.5-flash" });

export const generateAIInsights = async (industry) => {

  if (!industry) throw new Error("Industry is required for generating AI insights");

  const prompt = `

    Analyze the current state of the ${industry} industry and provide insights in ONLY the following JSON format without any additional notes or explanations:

    {

      "salaryRanges": [

        { "role": "string", "min": number, "max": number, "median": number, "location": "string" }

      ],

      "growthRate": number,

      "demandLevel": "HIGH" | "MEDIUM" | "LOW",

      "topSkills": ["skill1", "skill2"],

      "marketOutlook": "POSITIVE" | "NEUTRAL" | "NEGATIVE",

      "keyTrends": ["trend1", "trend2"],

      "recommendedSkills": ["skill1", "skill2"]

    }

    IMPORTANT: Return ONLY the JSON. No additional text, notes, or markdown formatting.

    Include at least 5 common roles for salary ranges.

    Growth rate should be a percentage.

    Include at least 5 skills and trends.

  `;

  const result = await model.generateContent(prompt);

  const response = result.response;

  const text = response.text();

  const cleanedText = text.replace(/```(?:json)?\n?/g, "").trim();

  return JSON.parse(cleanedText);

};

export async function getIndustryInsights() {

  const { userId } = await auth();

  if (!userId) throw new Error("Unauthorized");

  const user = await db.user.findUnique({

    where: { clerkUserId: userId },

    include: {

      industryInsight: true,

    },

  });

  if (!user) throw new Error("User not found");

  // If the user's industry is not set, redirect to the /onboarding page.

  if (!user.industry) {

    redirect("/onboarding");

  }

  // If no industry insights exist for the user, generate and store them.

  if (!user.industryInsight) {

    const insights = await generateAIInsights(user.industry);

    const industryInsight = await db.industryInsight.create({

      data: {

        industry: user.industry,

        ...insights,

        nextUpdate: new Date(Date.now() + 7 \* 24 \* 60 \* 60 \* 1000),

      },

    });

    return industryInsight;

  }

  return user.industryInsight;

}

1. Actions/interview.js:

"use server";

import { db } from "@/lib/prisma";

import { auth } from "@clerk/nextjs/server";

import { GoogleGenerativeAI } from "@google/generative-ai";

const genAI = new GoogleGenerativeAI(process.env.GEMINI\_API\_KEY);

const model = genAI.getGenerativeModel({ model: "gemini-1.5-flash" });

export async function generateQuiz() {

  const { userId } = await auth();

  if (!userId) throw new Error("Unauthorized");

  const user = await db.user.findUnique({

    where: { clerkUserId: userId },

    select: {

      industry: true,

      skills: true,

    },

  });

  if (!user) throw new Error("User not found");

  const prompt = `

    Generate 10 technical interview questions for a ${

      user.industry

    } professional${

    user.skills?.length ? ` with expertise in ${user.skills.join(", ")}` : ""

  }.

    Each question should be multiple choice with 4 options.

    Return the response in this JSON format only, no additional text:

    {

      "questions": [

        {

          "question": "string",

          "options": ["string", "string", "string", "string"],

          "correctAnswer": "string",

          "explanation": "string"

        }

      ]

    }

  `;

  try {

    const result = await model.generateContent(prompt);

    const response = result.response;

    const text = response.text();

    const cleanedText = text.replace(/```(?:json)?\n?/g, "").trim();

    const quiz = JSON.parse(cleanedText);

    return quiz.questions;

  } catch (error) {

    console.error("Error generating quiz:", error);

    throw new Error("Failed to generate quiz questions");

  }

}

export async function saveQuizResult(questions, answers, score) {

  const { userId } = await auth();

  if (!userId) throw new Error("Unauthorized");

  const user = await db.user.findUnique({

    where: { clerkUserId: userId },

  });

  if (!user) throw new Error("User not found");

  const questionResults = questions.map((q, index) => ({

    question: q.question,

    answer: q.correctAnswer,

    userAnswer: answers[index],

    isCorrect: q.correctAnswer === answers[index],

    explanation: q.explanation,

  }));

  // Get wrong answers

  const wrongAnswers = questionResults.filter((q) => !q.isCorrect);

  // Only generate improvement tips if there are wrong answers

  let improvementTip = null;

  if (wrongAnswers.length > 0) {

    const wrongQuestionsText = wrongAnswers

      .map(

        (q) =>

          `Question: "${q.question}"\nCorrect Answer: "${q.answer}"\nUser Answer: "${q.userAnswer}"`

      )

      .join("\n\n");

    const improvementPrompt = `

      The user got the following ${user.industry} technical interview questions wrong:

      ${wrongQuestionsText}

      Based on these mistakes, provide a concise, specific improvement tip.

      Focus on the knowledge gaps revealed by these wrong answers.

      Keep the response under 2 sentences and make it encouraging.

      Don't explicitly mention the mistakes, instead focus on what to learn/practice.

    `;

    try {

      const tipResult = await model.generateContent(improvementPrompt);

      improvementTip = tipResult.response.text().trim();

      console.log(improvementTip);

    } catch (error) {

      console.error("Error generating improvement tip:", error);

      // Continue without improvement tip if generation fails

    }

  }

  try {

    const assessment = await db.assessment.create({

      data: {

        userId: user.id,

        quizScore: score,

        questions: questionResults,

        category: "Technical",

        improvementTip,

      },

    });

    return assessment;

  } catch (error) {

    console.error("Error saving quiz result:", error);

    throw new Error("Failed to save quiz result");

  }

}

export async function getAssessments() {

  const { userId } = await auth();

  if (!userId) throw new Error("Unauthorized");

  const user = await db.user.findUnique({

    where: { clerkUserId: userId },

  });

  if (!user) throw new Error("User not found");

  try {

    const assessments = await db.assessment.findMany({

      where: {

        userId: user.id,

      },

      orderBy: {

        createdAt: "asc",

      },

    });

    return assessments;

  } catch (error) {

    console.error("Error fetching assessments:", error);

    throw new Error("Failed to fetch assessments");

  }

}

1. Actions/resume.js:

"use server";

import { db } from "@/lib/prisma";

import { auth } from "@clerk/nextjs/server";

import { GoogleGenerativeAI } from "@google/generative-ai";

const genAI = new GoogleGenerativeAI(process.env.GEMINI\_API\_KEY);

const model = genAI.getGenerativeModel({ model: "gemini-1.5-flash" });

export async function generateQuiz() {

  const { userId } = await auth();

  if (!userId) throw new Error("Unauthorized");

  const user = await db.user.findUnique({

    where: { clerkUserId: userId },

    select: {

      industry: true,

      skills: true,

    },

  });

  if (!user) throw new Error("User not found");

  const prompt = `

    Generate 10 technical interview questions for a ${

      user.industry

    } professional${

    user.skills?.length ? ` with expertise in ${user.skills.join(", ")}` : ""

  }.

    Each question should be multiple choice with 4 options.

    Return the response in this JSON format only, no additional text:

    {

      "questions": [

        {

          "question": "string",

          "options": ["string", "string", "string", "string"],

          "correctAnswer": "string",

          "explanation": "string"

        }

      ]

    }

  `;

  try {

    const result = await model.generateContent(prompt);

    const response = result.response;

    const text = response.text();

    const cleanedText = text.replace(/```(?:json)?\n?/g, "").trim();

    const quiz = JSON.parse(cleanedText);

    return quiz.questions;

  } catch (error) {

    console.error("Error generating quiz:", error);

    throw new Error("Failed to generate quiz questions");

  }

}

export async function saveQuizResult(questions, answers, score) {

  const { userId } = await auth();

  if (!userId) throw new Error("Unauthorized");

  const user = await db.user.findUnique({

    where: { clerkUserId: userId },

  });

  if (!user) throw new Error("User not found");

  const questionResults = questions.map((q, index) => ({

    question: q.question,

    answer: q.correctAnswer,

    userAnswer: answers[index],

    isCorrect: q.correctAnswer === answers[index],

    explanation: q.explanation,

  }));

  // Get wrong answers

  const wrongAnswers = questionResults.filter((q) => !q.isCorrect);

  // Only generate improvement tips if there are wrong answers

  let improvementTip = null;

  if (wrongAnswers.length > 0) {

    const wrongQuestionsText = wrongAnswers

      .map(

        (q) =>

          `Question: "${q.question}"\nCorrect Answer: "${q.answer}"\nUser Answer: "${q.userAnswer}"`

      )

      .join("\n\n");

    const improvementPrompt = `

      The user got the following ${user.industry} technical interview questions wrong:

      ${wrongQuestionsText}

      Based on these mistakes, provide a concise, specific improvement tip.

      Focus on the knowledge gaps revealed by these wrong answers.

      Keep the response under 2 sentences and make it encouraging.

      Don't explicitly mention the mistakes, instead focus on what to learn/practice.

    `;

    try {

      const tipResult = await model.generateContent(improvementPrompt);

      improvementTip = tipResult.response.text().trim();

      console.log(improvementTip);

    } catch (error) {

      console.error("Error generating improvement tip:", error);

      // Continue without improvement tip if generation fails

    }

  }

  try {

    const assessment = await db.assessment.create({

      data: {

        userId: user.id,

        quizScore: score,

        questions: questionResults,

        category: "Technical",

        improvementTip,

      },

    });

    return assessment;

  } catch (error) {

    console.error("Error saving quiz result:", error);

    throw new Error("Failed to save quiz result");

  }

}

export async function getAssessments() {

  const { userId } = await auth();

  if (!userId) throw new Error("Unauthorized");

  const user = await db.user.findUnique({

    where: { clerkUserId: userId },

  });

  if (!user) throw new Error("User not found");

  try {

    const assessments = await db.assessment.findMany({

      where: {

        userId: user.id,

      },

      orderBy: {

        createdAt: "asc",

      },

    });

    return assessments;

  } catch (error) {

    console.error("Error fetching assessments:", error);

    throw new Error("Failed to fetch assessments");

  }

}

1. Actions/user.js:

"use server";

import { db } from "@/lib/prisma";

import { auth } from "@clerk/nextjs/server";

import { generateAIInsights } from "./dashboard";

export async function UpdateUser(data){

    const {userId} = await auth();

    if(!userId) throw new Error('Unauthorized');

    const user = await db.user.findUnique({

        where: {

         clerkUserId: userId,

        },

    });

    if(!user) throw new Error('User not found');

    try{

       const result = await db.$transaction(

        async(tx)=>{

            let industryInsight=await tx.industryInsight.findUnique({

                where:{

                    industry: data.industry,

                }

            });

             if(!industryInsight){

                 const insights = await generateAIInsights(data.industry);

                              industryInsight = await db.industryInsight.create({

                                data: {

                                    industry: data.industry,

                                    ...insights,

                                    nextUpdate: new Date(Date.now() + 7\*24\*60\*60\*1000),

                                }

                              });

             }

             const updatedUser = await tx.user.update({

                where:{

                   id:user.id,

                },

                data:{

                    industry: data.industry,

                    experience: data.experience,

                    bio: data.bio,

                    skills: data.skills,

                },

             });

           return{updatedUser,industryInsight};

        },{timeout: 10000,}

       );

       return {success:true , ...result};

    }catch(error){

        console.error("Error updating user and industry",error.message);

          throw new Error('Failed to update profile' + error.message);

    }

}

export async function getUserOnboardingStatus(){

    const {userId} = await auth();

    if(!userId) throw new Error('Unauthorized');

    const user = await db.user.findUnique({

        where: {

         clerkUserId: userId,

        },

    });

    if(!user) throw new Error('User not found');

    try{

        const user = await db.user.findUnique({

            where:{

            clerkUserId: userId,

            },

            select:{

              industry: true,

            },

        });

        return{

            isOnboarded: !!user?.industry,

        };

    }catch(error){

        console.error("Error checking onboarding status",error.message);

        throw new Error('Failed to check onboarding status');

    }

}

**App folder:**

1. App/auth/sign-in/page.jsx:

import React from "react";

import { SignIn } from "@clerk/nextjs";

const Page = () => {

 return <SignIn />;

};

export default Page;

1. App/auth/sign-up/page.jsx:

import React from "react";

import { SignUp } from "@clerk/nextjs";

const Page = () => {

return <SignUp />;

};

export default Page;

1. App/auth/layout.js:

import React from "react";

const AuthLayout = ({children}) =>{

return <div className="flex justify-center pt-40">{children}</div>

};

export default AuthLayout;

**app/main folder**

1. App/main/ai-cover-letter/components/cover-letter-generator.jsx:

"use client";

import { useState } from "react";

import { useForm } from "react-hook-form";

import { zodResolver } from "@hookform/resolvers/zod";

import { toast } from "sonner";

import { Loader2 } from "lucide-react";

import { Button } from "@/components/ui/button";

import {

Card,

CardContent,

CardDescription,

CardHeader,

CardTitle,

} from "@/components/ui/card";

import { Input } from "@/components/ui/input";

import { Label } from "@/components/ui/label";

import { Textarea } from "@/components/ui/textarea";

import { generateCoverLetter } from "@/actions/cover-letter";

import useFetch from "@/hooks/use-fetch";

import { coverLetterSchema } from "@/app/lib/schema";

import { useEffect } from "react";

import { useRouter } from "next/navigation";

export default function CoverLetterGenerator() {

const router = useRouter();

const {

register,

handleSubmit,

formState: { errors },

reset,

} = useForm({

resolver: zodResolver(coverLetterSchema),

});

const {

loading: generating,

fn: generateLetterFn,

data: generatedLetter,

} = useFetch(generateCoverLetter);

// Update content when letter is generated

useEffect(() => {

if (generatedLetter) {

toast.success("Cover letter generated successfully!");

router.push(`/ai-cover-letter/${generatedLetter.id}`);

reset();

}

}, [generatedLetter]);

const onSubmit = async (data) => {

try {

await generateLetterFn(data);

} catch (error) {

toast.error(error.message || "Failed to generate cover letter");

}

};

return (

<div className="space-y-6">

<Card>

<CardHeader>

<CardTitle>Job Details</CardTitle>

<CardDescription>

Provide information about the position you're applying for

</CardDescription>

</CardHeader>

<CardContent>

<form onSubmit={handleSubmit(onSubmit)} className="space-y-4">

{/\* Form fields remain the same \*/}

<div className="grid grid-cols-2 gap-4">

<div className="space-y-2">

<Label htmlFor="companyName">Company Name</Label>

<Input

id="companyName"

placeholder="Enter company name"

{...register("companyName")}

/>

{errors.companyName && (

<p className="text-sm text-red-500">

{errors.companyName.message}

</p>

)}

</div>

<div className="space-y-2">

<Label htmlFor="jobTitle">Job Title</Label>

<Input

id="jobTitle"

placeholder="Enter job title"

{...register("jobTitle")}

/>

{errors.jobTitle && (

<p className="text-sm text-red-500">

{errors.jobTitle.message}

</p>

)}

</div>

</div>

<div className="space-y-2">

<Label htmlFor="jobDescription">Job Description</Label>

<Textarea

id="jobDescription"

placeholder="Paste the job description here"

className="h-32"

{...register("jobDescription")}

/>

{errors.jobDescription && (

<p className="text-sm text-red-500">

{errors.jobDescription.message}

</p>

)}

</div>

<div className="flex justify-end">

<Button type="submit" disabled={generating}>

{generating ? (

<>

<Loader2 className="mr-2 h-4 w-4 animate-spin" />

Generating...

</>

) : (

"Generate Cover Letter"

)}

</Button>

</div>

</form>

</CardContent>

</Card>

</div>

);

}

1. App/main/ai-cover-letter/components/cover-letter-list.js:

"use client";

import { useRouter } from "next/navigation";

import { format } from "date-fns";

import { Edit2, Eye, Trash2 } from "lucide-react";

import { toast } from "sonner";

import {

Card,

CardContent,

CardDescription,

CardHeader,

CardTitle,

} from "@/components/ui/card";

import { Button } from "@/components/ui/button";

import {

AlertDialog,

AlertDialogAction,

AlertDialogCancel,

AlertDialogContent,

AlertDialogDescription,

AlertDialogFooter,

AlertDialogHeader,

AlertDialogTitle,

AlertDialogTrigger,

} from "@/components/ui/alert-dialog";

import { deleteCoverLetter } from "@/actions/cover-letter";

export default function CoverLetterList({ coverLetters }) {

const router = useRouter();

const handleDelete = async (id) => {

try {

await deleteCoverLetter(id);

toast.success("Cover letter deleted successfully!");

router.refresh();

} catch (error) {

toast.error(error.message || "Failed to delete cover letter");

}

};

if (!coverLetters?.length) {

return (

<Card>

<CardHeader>

<CardTitle>No Cover Letters Yet</CardTitle>

<CardDescription>

Create your first cover letter to get started

</CardDescription>

</CardHeader>

</Card>

);

}

return (

<div className="space-y-4">

{coverLetters.map((letter) => (

<Card key={letter.id} className="group relative ">

<CardHeader>

<div className="flex items-start justify-between">

<div>

<CardTitle className="text-xl gradient-title">

{letter.jobTitle} at {letter.companyName}

</CardTitle>

<CardDescription>

Created {format(new Date(letter.createdAt), "PPP")}

</CardDescription>

</div>

<div className="flex space-x-2">

<AlertDialog>

<Button

variant="outline"

size="icon"

onClick={() => router.push(`/ai-cover-letter/${letter.id}`)}

>

<Eye className="h-4 w-4" />

</Button>

<AlertDialogTrigger asChild>

<Button variant="outline" size="icon">

<Trash2 className="h-4 w-4" />

</Button>

</AlertDialogTrigger>

<AlertDialogContent>

<AlertDialogHeader>

<AlertDialogTitle>Delete Cover Letter?</AlertDialogTitle>

<AlertDialogDescription>

This action cannot be undone. This will permanently

delete your cover letter for {letter.jobTitle} at{" "}

{letter.companyName}.

</AlertDialogDescription>

</AlertDialogHeader>

<AlertDialogFooter>

<AlertDialogCancel>Cancel</AlertDialogCancel>

<AlertDialogAction

onClick={() => handleDelete(letter.id)}

className="bg-destructive text-destructive-foreground hover:bg-destructive/90"

>

Delete

</AlertDialogAction>

</AlertDialogFooter>

</AlertDialogContent>

</AlertDialog>

</div>

</div>

</CardHeader>

<CardContent>

<div className="text-muted-foreground text-sm line-clamp-3">

{letter.jobDescription}

</div>

</CardContent>

</Card>

))}

</div>

);

}

3. App/main/ai-cover-letter/components/preview.js:

"use client";

import React from "react";

import MDEditor from "@uiw/react-md-editor";

const CoverLetterPreview = ({ content }) => {

  return (

    <div className="py-4">

      <MDEditor value={content} preview="preview" height={700} />

    </div>

  );

};

export default CoverLetterPreview;

1. app/(main)/[id]/page.jsx:

import Link from "next/link";

import { ArrowLeft } from "lucide-react";

import { Button } from "@/components/ui/button";

import { getCoverLetter } from "@/actions/cover-letter";

import CoverLetterPreview from "../\_components/cover-letter-preview";

export default async function EditCoverLetterPage({ params }) {

const { id } = await params;

const coverLetter = await getCoverLetter(id);

return (

<div className="container mx-auto py-6">

<div className="flex flex-col space-y-2">

<Link href="/ai-cover-letter">

<Button variant="link" className="gap-2 pl-0">

<ArrowLeft className="h-4 w-4" />

Back to Cover Letters

</Button>

</Link>

<h1 className="text-6xl font-bold gradient-title mb-6">

{coverLetter?.jobTitle} at {coverLetter?.companyName}

</h1>

</div>

<CoverLetterPreview content={coverLetter?.content} />

</div>

);

}

1. app\(main)\ai-cover-letter\new\page.jsx:

import Link from "next/link";

import { ArrowLeft } from "lucide-react";

import { Button } from "@/components/ui/button";

import CoverLetterGenerator from "../\_components/cover-letter-generator";

export default function NewCoverLetterPage() {

return (

<div className="container mx-auto py-6">

<div className="flex flex-col space-y-2">

<Link href="/ai-cover-letter">

<Button variant="link" className="gap-2 pl-0">

<ArrowLeft className="h-4 w-4" />

Back to Cover Letters

</Button>

</Link>

<div className="pb-6">

<h1 className="text-6xl font-bold gradient-title">

Create Cover Letter

</h1>

<p className="text-muted-foreground">

Generate a tailored cover letter for your job application

</p>

</div>

</div>

<CoverLetterGenerator />

</div>

);

}

1. \app\(main)\ai-cover-letter\page.jsx:

import { getCoverLetters } from "@/actions/cover-letter";

import Link from "next/link";

import { Plus } from "lucide-react";

import { Button } from "@/components/ui/button";

import CoverLetterList from "./\_components/cover-letter-list";

export default async function CoverLetterPage() {

const coverLetters = await getCoverLetters();

return (

<div>

<div className="flex flex-col md:flex-row gap-2 items-center justify-between mb-5">

<h1 className="text-6xl font-bold gradient-title">My Cover Letters</h1>

<Link href="/ai-cover-letter/new">

<Button>

<Plus className="h-4 w-4 mr-2" />

Create New

</Button>

</Link>

</div>

<CoverLetterList coverLetters={coverLetters} />

</div>

);

}

1. app\(main)\courses\page.jsx

"use client";

import { useEffect, useState } from "react";

import { useUser } from "@clerk/nextjs";

import {

Card,

CardHeader,

CardTitle,

CardContent,

CardFooter,

} from "@/components/ui/card";

import { Button } from "@/components/ui/button";

export default function CoursesPage() {

const { user } = useUser();

const [courses, setCourses] = useState([]);

const [loading, setLoading] = useState(true);

useEffect(() => {

if (user) {

fetch(`https://mlm-vrqj.onrender.com/api/get-courses?userId=${user.id}`, {

method: "GET",

headers: {

"Content-Type": "application/json",

},

})

.then((response) => response.json())

.then((data) => {

setCourses(data.recommendations || []);

setLoading(false);

})

.catch((error) => {

console.error("Error fetching courses:", error);

setLoading(false);

});

}

}, [user]);

return (

<div className="container mx-auto px-4 py-8">

<h1 className="text-3xl font-bold mb-6">Recommended Courses</h1>

{loading ? (

<p className="text-lg">Loading courses...</p>

) : courses.length > 0 ? (

<div className="grid grid-cols-1 sm:grid-cols-2 lg:grid-cols-3 gap-6">

{courses.map((course, index) => (

<Card key={index} className="shadow">

<CardHeader>

<CardTitle className="text-lg font-semibold truncate">

{course.title}

</CardTitle>

</CardHeader>

<CardContent>

<p className="text-sm text-muted-foreground truncate">

{course.url}

</p>

</CardContent>

<CardFooter>

<Button variant="link" asChild>

<a href={course.url} target="\_blank" rel="noopener noreferrer">

Visit Course

</a>

</Button>

</CardFooter>

</Card>

))}

</div>

) : (

<p className="text-lg">No courses found.</p>

)}

</div>

);

}

1. app\(main)\dashboard\\_components\dashboard-view.jsx

"use client";

import React from "react";

import {

BarChart,

Bar,

XAxis,

YAxis,

CartesianGrid,

Tooltip,

ResponsiveContainer,

} from "recharts";

import {

BriefcaseIcon,

LineChart,

TrendingUp,

TrendingDown,

Brain,

} from "lucide-react";

import { format, formatDistanceToNow } from "date-fns";

import {

Card,

CardContent,

CardDescription,

CardHeader,

CardTitle,

} from "@/components/ui/card";

import { Badge } from "@/components/ui/badge";

import { Progress } from "@/components/ui/progress";

const DashboardView = ({ insights }) => {

// Transform salary data for the chart

const salaryData = insights.salaryRanges.map((range) => ({

name: range.role,

min: range.min / 1000,

max: range.max / 1000,

median: range.median / 1000,

}));

const getDemandLevelColor = (level) => {

switch (level.toLowerCase()) {

case "high":

return "bg-green-500";

case "medium":

return "bg-yellow-500";

case "low":

return "bg-red-500";

default:

return "bg-gray-500";

}

};

const getMarketOutlookInfo = (outlook) => {

switch (outlook.toLowerCase()) {

case "positive":

return { icon: TrendingUp, color: "text-green-500" };

case "neutral":

return { icon: LineChart, color: "text-yellow-500" };

case "negative":

return { icon: TrendingDown, color: "text-red-500" };

default:

return { icon: LineChart, color: "text-gray-500" };

}

};

const OutlookIcon = getMarketOutlookInfo(insights.marketOutlook).icon;

const outlookColor = getMarketOutlookInfo(insights.marketOutlook).color;

// Format dates using date-fns

const lastUpdatedDate = format(new Date(insights.lastUpdated), "dd/MM/yyyy");

const nextUpdateDistance = formatDistanceToNow(

new Date(insights.nextUpdate),

{ addSuffix: true }

);

return (

<div className="space-y-6">

<div className="flex justify-between items-center">

<Badge variant="outline">Last updated: {lastUpdatedDate}</Badge>

</div>

{/\* Market Overview Cards \*/}

<div className="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-4 gap-4">

<Card>

<CardHeader className="flex flex-row items-center justify-between space-y-0 pb-2">

<CardTitle className="text-sm font-medium">

Market Outlook

</CardTitle>

<OutlookIcon className={`h-4 w-4 ${outlookColor}`} />

</CardHeader>

<CardContent>

<div className="text-2xl font-bold">{insights.marketOutlook}</div>

<p className="text-xs text-muted-foreground">

Next update {nextUpdateDistance}

</p>

</CardContent>

</Card>

<Card>

<CardHeader className="flex flex-row items-center justify-between space-y-0 pb-2">

<CardTitle className="text-sm font-medium">

Industry Growth

</CardTitle>

<TrendingUp className="h-4 w-4 text-muted-foreground" />

</CardHeader>

<CardContent>

<div className="text-2xl font-bold">

{insights.growthRate.toFixed(1)}%

</div>

<Progress value={insights.growthRate} className="mt-2" />

</CardContent>

</Card>

<Card>

<CardHeader className="flex flex-row items-center justify-between space-y-0 pb-2">

<CardTitle className="text-sm font-medium">Demand Level</CardTitle>

<BriefcaseIcon className="h-4 w-4 text-muted-foreground" />

</CardHeader>

<CardContent>

<div className="text-2xl font-bold">{insights.demandLevel}</div>

<div

className={`h-2 w-full rounded-full mt-2 ${getDemandLevelColor(

insights.demandLevel

)}`}

/>

</CardContent>

</Card>

<Card>

<CardHeader className="flex flex-row items-center justify-between space-y-0 pb-2">

<CardTitle className="text-sm font-medium">Top Skills</CardTitle>

<Brain className="h-4 w-4 text-muted-foreground" />

</CardHeader>

<CardContent>

<div className="flex flex-wrap gap-1">

{insights.topSkills.map((skill) => (

<Badge key={skill} variant="secondary">

{skill}

</Badge>

))}

</div>

</CardContent>

</Card>

</div>

{/\* Salary Ranges Chart \*/}

<Card className="col-span-4">

<CardHeader>

<CardTitle>Salary Ranges by Role</CardTitle>

<CardDescription>

Displaying minimum, median, and maximum salaries (in thousands)

</CardDescription>

</CardHeader>

<CardContent>

<div className="h-[400px]">

<ResponsiveContainer width="100%" height="100%">

<BarChart data={salaryData}>

<CartesianGrid strokeDasharray="3 3" />

<XAxis dataKey="name" />

<YAxis />

<Tooltip

content={({ active, payload, label }) => {

if (active && payload && payload.length) {

return (

<div className="bg-background border rounded-lg p-2 shadow-md">

<p className="font-medium">{label}</p>

{payload.map((item) => (

<p key={item.name} className="text-sm">

{item.name}: ${item.value}K

</p>

))}

</div>

);

}

return null;

}}

/>

<Bar dataKey="min" fill="#94a3b8" name="Min Salary (K)" />

<Bar dataKey="median" fill="#64748b" name="Median Salary (K)" />

<Bar dataKey="max" fill="#475569" name="Max Salary (K)" />

</BarChart>

</ResponsiveContainer>

</div>

</CardContent>

</Card>

{/\* Industry Trends \*/}

<div className="grid grid-cols-1 md:grid-cols-2 gap-4">

<Card>

<CardHeader>

<CardTitle>Key Industry Trends</CardTitle>

<CardDescription>

Current trends shaping the industry

</CardDescription>

</CardHeader>

<CardContent>

<ul className="space-y-4">

{insights.keyTrends.map((trend, index) => (

<li key={index} className="flex items-start space-x-2">

<div className="h-2 w-2 mt-2 rounded-full bg-primary" />

<span>{trend}</span>

</li>

))}

</ul>

</CardContent>

</Card>

<Card>

<CardHeader>

<CardTitle>Recommended Skills</CardTitle>

<CardDescription>Skills to consider developing</CardDescription>

</CardHeader>

<CardContent>

<div className="flex flex-wrap gap-2">

{insights.recommendedSkills.map((skill) => (

<Badge key={skill} variant="outline">

{skill}

</Badge>

))}

</div>

</CardContent>

</Card>

</div>

</div>

);

};

export default DashboardView;

1. app\(main)\dashboard\layout.js:

import React, { Suspense } from "react";

import { BarLoader } from "react-spinners";

const Layout = ({children}) => {

return (

<div className="px-5">

<div className="flex justify-between items-center mb-4">

<h1 className="text-6xl font-bold gradient-title">

Industry Insights

</h1>

</div>

<Suspense

fallback={<BarLoader className="mt-4" width={"100%"} color="gray"/>}

>

{children}

</Suspense>

</div>

)

}

export default Layout;

1. app\(main)\dashboard\page.jsx

import React from "react";

import { getUserOnboardingStatus } from "@/actions/user";

import { redirect } from "next/navigation";

import { getIndustryInsights } from "@/actions/dashboard";

import DashboardView from "./\_components/dashboard-view";

const IndustryInsightsPage = async() => {

const {isOnboarded}= await getUserOnboardingStatus();

const insights = await getIndustryInsights();

if (!isOnboarded) {

redirect("/onboarding");

}

return(

<div className="container mx-auto">

<DashboardView

insights={insights}

/>

</div>

);

}

export default IndustryInsightsPage

1. app\(main)\interview\\_components\performace-chart.jsx:

"use client";

import {

Line,

XAxis,

YAxis,

CartesianGrid,

Tooltip,

ResponsiveContainer,

} from "recharts";

import {

Card,

CardContent,

CardDescription,

CardHeader,

CardTitle,

} from "@/components/ui/card";

import { useEffect, useState } from "react";

import { format } from "date-fns";

import { LineChart } from "recharts";

export default function PerformanceChart({ assessments }) {

const [chartData, setChartData] = useState([]);

useEffect(() => {

if (assessments) {

const formattedData = assessments.map((assessment) => ({

date: format(new Date(assessment.createdAt), "MMM dd"),

score: assessment.quizScore,

}));

setChartData(formattedData);

}

}, [assessments]);

return (

<Card>

<CardHeader>

<CardTitle className="gradient-title text-3xl md:text-4xl">

Performance Trend

</CardTitle>

<CardDescription>Your quiz scores over time</CardDescription>

</CardHeader>

<CardContent>

<div className="h-[300px]">

<ResponsiveContainer width="100%" height="100%">

<LineChart data={chartData}>

<CartesianGrid strokeDasharray="3 3" />

<XAxis dataKey="date" />

<YAxis domain={[0, 100]} />

<Tooltip

content={({ active, payload }) => {

if (active && payload?.length) {

return (

<div className="bg-background border rounded-lg p-2 shadow-md">

<p className="text-sm font-medium">

Score: {payload[0].value}%

</p>

<p className="text-xs text-muted-foreground">

{payload[0].payload.date}

</p>

</div>

);

}

return null;

}}

/>

<Line

type="monotone"

dataKey="score"

stroke="#FFFFFF"

strokeWidth={2}

/>

</LineChart>

</ResponsiveContainer>

</div>

</CardContent>

</Card>

);

}

1. app\(main)\interview\\_components\quiz-list.jsx

"use client";

import { useState } from "react";

import { format } from "date-fns";

import { useRouter } from "next/navigation";

import { Button } from "@/components/ui/button";

import {

Card,

CardContent,

CardDescription,

CardHeader,

CardTitle,

} from "@/components/ui/card";

import {

Dialog,

DialogContent,

DialogHeader,

DialogTitle,

} from "@/components/ui/dialog";

import QuizResult from "./quiz-result";

export default function QuizList({ assessments }) {

const router = useRouter();

const [selectedQuiz, setSelectedQuiz] = useState(null);

return (

<>

<Card>

<CardHeader>

<div className="flex items-center justify-between">

<div>

<CardTitle className="gradient-title text-3xl md:text-4xl">

Recent Quizzes

</CardTitle>

<CardDescription>

Review your past quiz performance

</CardDescription>

</div>

<Button onClick={() => router.push("/interview/mock")}>

Start New Quiz

</Button>

</div>

</CardHeader>

<CardContent>

<div className="space-y-4">

{assessments?.map((assessment, i) => (

<Card

key={assessment.id}

className="cursor-pointer hover:bg-muted/50 transition-colors"

onClick={() => setSelectedQuiz(assessment)}

>

<CardHeader>

<CardTitle className="gradient-title text-2xl">

Quiz {i + 1}

</CardTitle>

<CardDescription className="flex justify-between w-full">

<div>Score: {assessment.quizScore.toFixed(1)}%</div>

<div>

{format(

new Date(assessment.createdAt),

"MMMM dd, yyyy HH:mm"

)}

</div>

</CardDescription>

</CardHeader>

{assessment.improvementTip && (

<CardContent>

<p className="text-sm text-muted-foreground">

{assessment.improvementTip}

</p>

</CardContent>

)}

</Card>

))}

</div>

</CardContent>

</Card>

<Dialog open={!!selectedQuiz} onOpenChange={() => setSelectedQuiz(null)}>

<DialogContent className="max-w-3xl max-h-[90vh] overflow-y-auto">

<DialogHeader>

<DialogTitle></DialogTitle>

</DialogHeader>

<QuizResult

result={selectedQuiz}

hideStartNew

onStartNew={() => router.push("/interview/mock")}

/>

</DialogContent>

</Dialog>

</>

);

}

1. app\(main)\interview\\_components\quiz-result.jsx:

"use client";

import { Trophy, CheckCircle2, XCircle } from "lucide-react";

import { Button } from "@/components/ui/button";

import { CardContent, CardFooter } from "@/components/ui/card";

import { Progress } from "@/components/ui/progress";

export default function QuizResult({

result,

hideStartNew = false,

onStartNew,

}) {

if (!result) return null;

return (

<div className="mx-auto">

<h1 className="flex items-center gap-2 text-3xl gradient-title">

<Trophy className="h-6 w-6 text-yellow-500" />

Quiz Results

</h1>

<CardContent className="space-y-6">

{/\* Score Overview \*/}

<div className="text-center space-y-2">

<h3 className="text-2xl font-bold">{result.quizScore.toFixed(1)}%</h3>

<Progress value={result.quizScore} className="w-full" />

</div>

{/\* Improvement Tip \*/}

{result.improvementTip && (

<div className="bg-muted p-4 rounded-lg">

<p className="font-medium">Improvement Tip:</p>

<p className="text-muted-foreground">{result.improvementTip}</p>

</div>

)}

{/\* Questions Review \*/}

<div className="space-y-4">

<h3 className="font-medium">Question Review</h3>

{result.questions.map((q, index) => (

<div key={index} className="border rounded-lg p-4 space-y-2">

<div className="flex items-start justify-between gap-2">

<p className="font-medium">{q.question}</p>

{q.isCorrect ? (

<CheckCircle2 className="h-5 w-5 text-green-500 flex-shrink-0" />

) : (

<XCircle className="h-5 w-5 text-red-500 flex-shrink-0" />

)}

</div>

<div className="text-sm text-muted-foreground">

<p>Your answer: {q.userAnswer}</p>

{!q.isCorrect && <p>Correct answer: {q.answer}</p>}

</div>

<div className="text-sm bg-muted p-2 rounded">

<p className="font-medium">Explanation:</p>

<p>{q.explanation}</p>

</div>

</div>

))}

</div>

</CardContent>

{!hideStartNew && (

<CardFooter>

<Button onClick={onStartNew} className="w-full">

Start New Quiz

</Button>

</CardFooter>

)}

</div>

);

}

1. app\(main)\interview\\_components\quiz.jsx:

"use client";

import { useState, useEffect } from "react";

import { toast } from "sonner";

import { Button } from "@/components/ui/button";

import {

Card,

CardContent,

CardFooter,

CardHeader,

CardTitle,

} from "@/components/ui/card";

import { RadioGroup, RadioGroupItem } from "@/components/ui/radio-group";

import { Label } from "@/components/ui/label";

import { generateQuiz, saveQuizResult } from "@/actions/interview";

import QuizResult from "./quiz-result";

import useFetch from "@/hooks/use-fetch";

import { BarLoader } from "react-spinners";

export default function Quiz() {

const [currentQuestion, setCurrentQuestion] = useState(0);

const [answers, setAnswers] = useState([]);

const [showExplanation, setShowExplanation] = useState(false);

const {

loading: generatingQuiz,

fn: generateQuizFn,

data: quizData,

} = useFetch(generateQuiz);

const {

loading: savingResult,

fn: saveQuizResultFn,

data: resultData,

setData: setResultData,

} = useFetch(saveQuizResult);

useEffect(() => {

if (quizData) {

setAnswers(new Array(quizData.length).fill(null));

}

}, [quizData]);

const handleAnswer = (answer) => {

const newAnswers = [...answers];

newAnswers[currentQuestion] = answer;

setAnswers(newAnswers);

};

const handleNext = () => {

if (currentQuestion < quizData.length - 1) {

setCurrentQuestion(currentQuestion + 1);

setShowExplanation(false);

} else {

finishQuiz();

}

};

const calculateScore = () => {

let correct = 0;

answers.forEach((answer, index) => {

if (answer === quizData[index].correctAnswer) {

correct++;

}

});

return (correct / quizData.length) \* 100;

};

const finishQuiz = async () => {

const score = calculateScore();

try {

await saveQuizResultFn(quizData, answers, score);

toast.success("Quiz completed!");

} catch (error) {

toast.error(error.message || "Failed to save quiz results");

}

};

const startNewQuiz = () => {

setCurrentQuestion(0);

setAnswers([]);

setShowExplanation(false);

generateQuizFn();

setResultData(null);

};

if (generatingQuiz) {

return <BarLoader className="mt-4" width={"100%"} color="gray" />;

}

// Show results if quiz is completed

if (resultData) {

return (

<div className="mx-2">

<QuizResult result={resultData} onStartNew={startNewQuiz} />

</div>

);

}

if (!quizData) {

return (

<Card className="mx-2">

<CardHeader>

<CardTitle>Ready to test your knowledge?</CardTitle>

</CardHeader>

<CardContent>

<p className="text-muted-foreground">

This quiz contains 10 questions specific to your industry and

skills. Take your time and choose the best answer for each question.

</p>

</CardContent>

<CardFooter>

<Button onClick={generateQuizFn} className="w-full">

Start Quiz

</Button>

</CardFooter>

</Card>

);

}

const question = quizData[currentQuestion];

return (

<Card className="mx-2">

<CardHeader>

<CardTitle>

Question {currentQuestion + 1} of {quizData.length}

</CardTitle>

</CardHeader>

<CardContent className="space-y-4">

<p className="text-lg font-medium">{question.question}</p>

<RadioGroup

onValueChange={handleAnswer}

value={answers[currentQuestion]}

className="space-y-2"

>

{question.options.map((option, index) => (

<div key={index} className="flex items-center space-x-2">

<RadioGroupItem value={option} id={`option-${index}`} />

<Label htmlFor={`option-${index}`}>{option}</Label>

</div>

))}

</RadioGroup>

{showExplanation && (

<div className="mt-4 p-4 bg-muted rounded-lg">

<p className="font-medium">Explanation:</p>

<p className="text-muted-foreground">{question.explanation}</p>

</div>

)}

</CardContent>

<CardFooter className="flex justify-between">

{!showExplanation && (

<Button

onClick={() => setShowExplanation(true)}

variant="outline"

disabled={!answers[currentQuestion]}

>

Show Explanation

</Button>

)}

<Button

onClick={handleNext}

disabled={!answers[currentQuestion] || savingResult}

className="ml-auto"

>

{savingResult && (

<BarLoader className="mt-4" width={"100%"} color="gray" />

)}

{currentQuestion < quizData.length - 1

? "Next Question"

: "Finish Quiz"}

</Button>

</CardFooter>

</Card>

);

}

1. app\(main)\interview\\_components\stats-cards.jsx:

import { Brain, Target, Trophy } from "lucide-react";

import { Card, CardContent, CardHeader, CardTitle } from "@/components/ui/card";

export default function StatsCards({ assessments }) {

const getAverageScore = () => {

if (!assessments?.length) return 0;

const total = assessments.reduce(

(sum, assessment) => sum + assessment.quizScore,

0

);

return (total / assessments.length).toFixed(1);

};

const getLatestAssessment = () => {

if (!assessments?.length) return null;

return assessments[0];

};

const getTotalQuestions = () => {

if (!assessments?.length) return 0;

return assessments.reduce(

(sum, assessment) => sum + assessment.questions.length,

0

);

};

return (

<div className="grid gap-4 md:grid-cols-3">

<Card>

<CardHeader className="flex flex-row items-center justify-between space-y-0 pb-2">

<CardTitle className="text-sm font-medium">Average Score</CardTitle>

<Trophy className="h-4 w-4 text-muted-foreground" />

</CardHeader>

<CardContent>

<div className="text-2xl font-bold">{getAverageScore()}%</div>

<p className="text-xs text-muted-foreground">

Across all assessments

</p>

</CardContent>

</Card>

<Card>

<CardHeader className="flex flex-row items-center justify-between space-y-0 pb-2">

<CardTitle className="text-sm font-medium">

Questions Practiced

</CardTitle>

<Brain className="h-4 w-4 text-muted-foreground" />

</CardHeader>

<CardContent>

<div className="text-2xl font-bold">{getTotalQuestions()}</div>

<p className="text-xs text-muted-foreground">Total questions</p>

</CardContent>

</Card>

<Card>

<CardHeader className="flex flex-row items-center justify-between space-y-0 pb-2">

<CardTitle className="text-sm font-medium">Latest Score</CardTitle>

<Target className="h-4 w-4 text-muted-foreground" />

</CardHeader>

<CardContent>

<div className="text-2xl font-bold">

{getLatestAssessment()?.quizScore.toFixed(1) || 0}%

</div>

<p className="text-xs text-muted-foreground">Most recent quiz</p>

</CardContent>

</Card>

</div>

);

}

1. app\(main)\interview\mock\page.jsx:

import Link from "next/link";

import { ArrowLeft } from "lucide-react";

import { Button } from "@/components/ui/button";

import Quiz from "../\_components/quiz";

export default function MockInterviewPage() {

return (

<div className="container mx-auto space-y-4 py-6">

<div className="flex flex-col space-y-2 mx-2">

<Link href="/interview">

<Button variant="link" className="gap-2 pl-0">

<ArrowLeft className="h-4 w-4" />

Back to Interview Preparation

</Button>

</Link>

<div>

<h1 className="text-6xl font-bold gradient-title">Mock Interview</h1>

<p className="text-muted-foreground">

Test your knowledge with industry-specific questions

</p>

</div>

</div>

<Quiz />

</div>

);

}

1. app\(main)\interview\layout.js:

import { Suspense } from "react";

import { BarLoader } from "react-spinners";

export default function Layout({ children }) {

return (

<div className="px-5">

<Suspense

fallback={<BarLoader className="mt-4" width={"100%"} color="gray" />}

>

{children}

</Suspense>

</div>

);

}

1. app\(main)\interview\page.jsx:

import { getAssessments } from "@/actions/interview";

import StatsCards from "./\_components/stats-cards";

import PerformanceChart from "./\_components/performace-chart";

import QuizList from "./\_components/quiz-list";

export default async function InterviewPrepPage() {

const assessments = await getAssessments();

return (

<div>

<div className="flex items-center justify-between mb-5">

<h1 className="text-6xl font-bold gradient-title">

Interview Preparation

</h1>

</div>

<div className="space-y-6">

<StatsCards assessments={assessments} />

<PerformanceChart assessments={assessments} />

<QuizList assessments={assessments} />

</div>

</div>

);

}

1. app\(main)\onboarding\\_components\onboarding-form.jsx:

"use client";

import { Button } from "@/components/ui/button";

import { Textarea } from "@/components/ui/textarea";

import { Input } from "@/components/ui/input";

import { useForm } from "react-hook-form";

import { zodResolver } from '@hookform/resolvers/zod';

import React, { useState } from "react";

import { onboardingSchema } from "@/app/lib/schema";

import { useRouter } from 'next/navigation';

import { Card,CardHeader,CardContent,CardDescription,CardFooter,CardTitle } from "@/components/ui/card";

import { Select,SelectTrigger,SelectValue,SelectItem,SelectContent } from "@/components/ui/select";

import { Label } from "@/components/ui/label";

import useFetch from "@/hooks/use-fetch";

import { UpdateUser } from "@/actions/user";

import { toast } from "sonner";

import { Loader2 } from "lucide-react";

import { useEffect } from "react";

const OnboardingForm = ({industries}) => {

const[selectedIndustry,setSelectedIndustry]=useState(null);

const router = useRouter();

const {loading:updateLoading,

fn: updateUserFn,

data: updateResult,

} = useFetch(UpdateUser);

const{

register,

handleSubmit,

formState: { errors },

setValue,

watch,

}=useForm({

resolver: zodResolver(onboardingSchema),

});

const onSubmit = async (values) => {

try{

const formattedIndustry = `${values.industry}-${values.subIndustry

.toLowerCase()

.replace(/ /g,"-")}`;

await updateUserFn({

...values,

industry: formattedIndustry,

});

}catch(error){

console.error("onboarding error:",error)

}

};

useEffect(()=>{

if(updateResult?.success && !updateLoading){

toast.success("Profile completed succesfully");

router.push("/dashboard");

router.refresh();

}

},[updateResult,updateLoading]);

const watchIndustry = watch("industry");

return(

<div className="flex items-center justify-center bg-background">

<Card className="w-full max-w-lg mt-10 mx-2">

<CardHeader>

<CardTitle className="gradient-title text-4xl">Complete your profile</CardTitle>

<CardDescription>Select your industry to get personalized career Insights

and recommendations

</CardDescription>

</CardHeader>

<CardContent>

<form className="space-y-6 " onSubmit={handleSubmit(onSubmit)}>

<div className="space-y-2">

<Label htmlFor="industry">

Industry

</Label>

<Select

onValueChange = {(value)=>{

setValue("industry",value);

setSelectedIndustry(

industries.find((ind)=>ind.id === value)

);

setValue("subIndustry" , "");

}}

>

<SelectTrigger id="industry">

<SelectValue placeholder="Select an industry" />

</SelectTrigger>

<SelectContent>

{industries.map((ind)=>{

return <SelectItem value={ind.id} key={ind.id}>{ind.name}</SelectItem>

})}

</SelectContent>

</Select>

{

errors.industry && (

<p className="text-sm text-red-500">

{errors.industry.message}

</p>

)

}

</div>

{watchIndustry && (

<div className="space-y-2">

<Label htmlFor="subindustry">

Specialization

</Label>

<Select

onValueChange = {(value)=>{

setValue("subIndustry" , value);

}}

>

<SelectTrigger id="subindustry">

<SelectValue placeholder="Select specialization" />

</SelectTrigger>

<SelectContent>

{selectedIndustry?.subIndustries.map((ind)=>{

return <SelectItem value={ind} key={ind}>{ind}</SelectItem>

})}

</SelectContent>

</Select>

{

errors.subIndustry && (

<p className="text-sm text-red-500">

{errors.subIndustry.message}

</p>

)

}

</div>

)

}

<div className="space-y-2">

<Label htmlFor="experience">

Years of Experience

</Label>

<Input

id="experience"

type = "number"

min = "0"

max = "50"

placeholder = "Enter Years of Experience"

{...register("experience")}

/>

{

errors.experience && (

<p className="text-sm text-red-500">

{errors.experience.message}

</p>

)

}

</div>

<div className="space-y-2">

<Label htmlFor="skills">

Skills

</Label>

<Input

id="skills"

placeholder="eg Python , Java, C++"

{...register("skills")}

/>

<p className="text-sm text-muted-foreground">

Separate multiple skills with commas

</p>

{

errors.skills && (

<p className="text-sm text-red-500">

{errors.skills.message}

</p>

)

}

</div>

<div className="space-y-2">

<Label htmlFor="bio">

Professional Bio

</Label>

<Textarea

id="bio"

placeholder="Tell about your background"

className="h-32"

{...register("bio")}

/>

{

errors.bio && (

<p className="text-sm text-red-500">

{errors.bio.message}

</p>

)

}

</div>

<Button type="submit" className="w-full" disabled={updateLoading}>

{updateLoading?

(

<>

<Loader2 className="mr-2 h-4 w-4 animate-spin"/>

Saving...

</>

) :(

"Complete Profile"

)

}

</Button>

</form>

</CardContent>

</Card>

</div>

);

}

export default OnboardingForm;

1. app\(main)\onboarding\page.jsx:

import { getUserOnboardingStatus } from "@/actions/user";

import { industries } from "@/data/industries";

import { redirect } from "next/dist/server/api-utils";

import React from "react";

import OnboardingForm from "./\_components/onboarding-form";

const OnboardingPage = async () => {

const {isOnboarded}= await getUserOnboardingStatus();

if (isOnboarded) {

redirect("/dashboard");

}

return(

<main>

<OnboardingForm industries={industries}/>

</main>

);

}

export default OnboardingPage

1. app\(main)\resume\page.jsx:

import { getResume } from "@/actions/resume";

import ResumeBuilder from "./\_components/resume-builder";

export default async function ResumePage() {

const resume = await getResume();

return (

<div className="container mx-auto py-6">

<ResumeBuilder initialContent={resume?.content} />

</div>

);

}

1. app\(main)\layout.js:

import React from "react";

const MainLayout = async ({ children }) => {

return <div className="container mx-auto mt-24 mb-20">{children}</div>;

};

export default MainLayout;

**app/api folder:**

1. app\api\inngest\route.js:

import { serve } from "inngest/next";

import { inngest } from "@/lib/inngest/client";

import { generateIndustryInsights } from "@/lib/inngest/functions";

export const { GET, POST, PUT } = serve({

client: inngest,

functions: [generateIndustryInsights],

});

**App/lib folder:**

1. app\lib\helper.js:

// Helper function to convert entries to markdown

export function entriesToMarkdown(entries, type) {

if (!entries?.length) return "";

return (

`## ${type}\n\n` +

entries

.map((entry) => {

const dateRange = entry.current

? `${entry.startDate} - Present`

: `${entry.startDate} - ${entry.endDate}`;

return `### ${entry.title} @ ${entry.organization}\n${dateRange}\n\n${entry.description}`;

})

.join("\n\n")

);

}

1. app\lib\schema.js:

import { z } from "zod";

export const onboardingSchema = z.object({

industry: z

.string({ required\_error: 'Please select an industry' })

.nonempty({ message: 'Please select an industry' }),

subIndustry: z

.string({ required\_error: 'Please select a specialization' })

.nonempty({ message: 'Please select a specialization' }),

bio: z.string().max(500).optional(),

experience: z

.string({ required\_error: 'Please enter years of experience' })

.nonempty({ message: 'Please enter years of experience' })

.transform((val) => parseInt(val, 10))

.pipe(

z

.number()

.min(0, "Experience must be at least 0 years")

.max(50, "Experience cannot exceed 50 years")

),

skills: z

.string({ required\_error: 'Please provide your skills' })

.nonempty({ message: 'Please provide your skills' })

.transform((val) =>

val.split(',').map((skill) => skill.trim()).filter(Boolean)

),

});

export const contactSchema = z.object({

email: z.string().email("Invalid email address"),

mobile: z.string().optional(),

linkedin: z.string().optional(),

twitter: z.string().optional(),

});

export const entrySchema = z

.object({

title: z.string().min(1, "Title is required"),

organization: z.string().min(1, "Organization is required"),

startDate: z.string().min(1, "Start date is required"),

endDate: z.string().optional(),

description: z.string().min(1, "Description is required"),

current: z.boolean().default(false),

})

.refine(

(data) => {

if (!data.current && !data.endDate) {

return false;

}

return true;

},

{

message: "End date is required unless this is your current position",

path: ["endDate"],

}

);

export const resumeSchema = z.object({

contactInfo: contactSchema,

summary: z.string().min(1, "Professional summary is required"),

skills: z.string().min(1, "Skills are required"),

experience: z.array(entrySchema),

education: z.array(entrySchema),

projects: z.array(entrySchema),

});

export const coverLetterSchema = z.object({

companyName: z.string().min(1, "Company name is required"),

jobTitle: z.string().min(1, "Job title is required"),

jobDescription: z.string().min(1, "Job description is required"),

});

App/global.css:

@import "tailwindcss";

@plugin "tailwindcss-animate";

@custom-variant dark (&:is(.dark \*));

@theme inline {

  --color-background: var(--background);

  --color-foreground: var(--foreground);

  --font-sans: var(--font-geist-sans);

  --font-mono: var(--font-geist-mono);

  --color-sidebar-ring: var(--sidebar-ring);

  --color-sidebar-border: var(--sidebar-border);

  --color-sidebar-accent-foreground: var(--sidebar-accent-foreground);

  --color-sidebar-accent: var(--sidebar-accent);

  --color-sidebar-primary-foreground: var(--sidebar-primary-foreground);

  --color-sidebar-primary: var(--sidebar-primary);

  --color-sidebar-foreground: var(--sidebar-foreground);

  --color-sidebar: var(--sidebar);

  --color-chart-5: var(--chart-5);

  --color-chart-4: var(--chart-4);

  --color-chart-3: var(--chart-3);

  --color-chart-2: var(--chart-2);

  --color-chart-1: var(--chart-1);

  --color-ring: var(--ring);

  --color-input: var(--input);

  --color-border: var(--border);

  --color-destructive-foreground: var(--destructive-foreground);

  --color-destructive: var(--destructive);

  --color-accent-foreground: var(--accent-foreground);

  --color-accent: var(--accent);

  --color-muted-foreground: var(--muted-foreground);

  --color-muted: var(--muted);

  --color-secondary-foreground: var(--secondary-foreground);

  --color-secondary: var(--secondary);

  --color-primary-foreground: var(--primary-foreground);

  --color-primary: var(--primary);

  --color-popover-foreground: var(--popover-foreground);

  --color-popover: var(--popover);

  --color-card-foreground: var(--card-foreground);

  --color-card: var(--card);

  --radius-sm: calc(var(--radius) - 4px);

  --radius-md: calc(var(--radius) - 2px);

  --radius-lg: var(--radius);

  --radius-xl: calc(var(--radius) + 4px);

}

:root {

  --background: oklch(1 0 0);

  --foreground: oklch(0.145 0 0);

  --card: oklch(1 0 0);

  --card-foreground: oklch(0.145 0 0);

  --popover: oklch(1 0 0);

  --popover-foreground: oklch(0.145 0 0);

  --primary: oklch(0.205 0 0);

  --primary-foreground: oklch(0.985 0 0);

  --secondary: oklch(0.97 0 0);

  --secondary-foreground: oklch(0.205 0 0);

  --muted: oklch(0.97 0 0);

  --muted-foreground: oklch(0.556 0 0);

  --accent: oklch(0.97 0 0);

  --accent-foreground: oklch(0.205 0 0);

  --destructive: oklch(0.577 0.245 27.325);

  --destructive-foreground: oklch(0.577 0.245 27.325);

  --border: oklch(0.922 0 0);

  --input: oklch(0.922 0 0);

  --ring: oklch(0.708 0 0);

  --chart-1: oklch(0.646 0.222 41.116);

  --chart-2: oklch(0.6 0.118 184.704);

  --chart-3: oklch(0.398 0.07 227.392);

  --chart-4: oklch(0.828 0.189 84.429);

  --chart-5: oklch(0.769 0.188 70.08);

  --radius: 0.625rem;

  --sidebar: oklch(0.985 0 0);

  --sidebar-foreground: oklch(0.145 0 0);

  --sidebar-primary: oklch(0.205 0 0);

  --sidebar-primary-foreground: oklch(0.985 0 0);

  --sidebar-accent: oklch(0.97 0 0);

  --sidebar-accent-foreground: oklch(0.205 0 0);

  --sidebar-border: oklch(0.922 0 0);

  --sidebar-ring: oklch(0.708 0 0);

}

.dark {

  --background: oklch(0.145 0 0);

  --foreground: oklch(0.985 0 0);

  --card: oklch(0.145 0 0);

  --card-foreground: oklch(0.985 0 0);

  --popover: oklch(0.145 0 0);

  --popover-foreground: oklch(0.985 0 0);

  --primary: oklch(0.985 0 0);

  --primary-foreground: oklch(0.205 0 0);

  --secondary: oklch(0.269 0 0);

  --secondary-foreground: oklch(0.985 0 0);

  --muted: oklch(0.269 0 0);

  --muted-foreground: oklch(0.708 0 0);

  --accent: oklch(0.269 0 0);

  --accent-foreground: oklch(0.985 0 0);

  --destructive: oklch(0.396 0.141 25.723);

  --destructive-foreground: oklch(0.637 0.237 25.331);

  --border: oklch(0.269 0 0);

  --input: oklch(0.269 0 0);

  --ring: oklch(0.439 0 0);

  --chart-1: oklch(0.488 0.243 264.376);

  --chart-2: oklch(0.696 0.17 162.48);

  --chart-3: oklch(0.769 0.188 70.08);

  --chart-4: oklch(0.627 0.265 303.9);

  --chart-5: oklch(0.645 0.246 16.439);

  --sidebar: oklch(0.205 0 0);

  --sidebar-foreground: oklch(0.985 0 0);

  --sidebar-primary: oklch(0.488 0.243 264.376);

  --sidebar-primary-foreground: oklch(0.985 0 0);

  --sidebar-accent: oklch(0.269 0 0);

  --sidebar-accent-foreground: oklch(0.985 0 0);

  --sidebar-border: oklch(0.269 0 0);

  --sidebar-ring: oklch(0.439 0 0);

}

@layer base {

  \* {

    @apply border-border outline-ring/50;

  }

  body {

    @apply bg-background text-foreground;

  }

}

.grid-background {

  position: fixed;

  top: 0;

  left: 0;

  width: 100%;

  height: 100%;

  background: linear-gradient(

    to right,

    rgba(255,255,255,0.1) 1px,

    transparent 1px

  ),

  linear-gradient(to bottom , rgba(255,255,255,0.1) 1px, transparent 1px);

  background-size: 50px 50px;

  pointer-events: none;

  z-index: -1;

}

.grid-background::before{

  content: " " ;

  position: absolute;

  left: 0;

  top: 0;

  width: 100%;

  height: 100%;

  background: radial-gradient(circle,transparent,rgba(0,0,0,0.9));

}

.hero-image-wrapper{

  perspective: 1000px;

}

.hero-image{

  transform: rotateX(15deg) scale(1);

  transition: transform 0.5s ease-out;

  will-change: transform;

}

.hero-image.scrolled{

  transform: rotateX(0deg) scale(1) translateY(40px);

}

app\layout.js:

import { Inter } from "next/font/google";

import "./globals.css";

import { ThemeProvider } from "@/components/theme-provider";

import Header from "@/components/header";

import { ClerkProvider } from "@clerk/nextjs";

import { dark } from '@clerk/themes'

import { Toaster } from "sonner";

const inter = Inter({ subsets: ["latin"] });

export const metadata = {

  title: "Create Next App",

  description: "Generated by create next app",

};

export default function RootLayout({ children }) {

  return (

    <ClerkProvider appearance={{

      baseTheme: dark

    }}>

      <html lang="en" suppressHydrationWarning>

        <body className={`${inter.className}`}>

          <ThemeProvider

            attribute="class"

            defaultTheme="dark"

            enableSystem

            disableTransitionOnChange

          >

            {/\*header\*/}

            <Header />

            <main className="min-h-screen">

              {children}

            </main>

            <Toaster richColors />

            {/\*footer\*/}

            <footer className="bg-muted/50 py-3">

              <div className="container mx-auto px-4 text-center text-gray-200">

                <p>Made with ❤️</p>

              </div>

            </footer>

          </ThemeProvider>

        </body>

      </html>

    </ClerkProvider>

  );

}

app\not-found.jsx:

import Link from "next/link";

import { Button } from "@/components/ui/button";

export default function NotFound() {

  return (

    <div className="flex flex-col items-center justify-center min-h-screen px-4 text-center">

      <h1 className="text-6xl font-bold gradient-title mb-4">404</h1>

      <h2 className="text-2xl font-semibold mb-4">Page Not Found</h2>

      <p className="text-gray-600 mb-8">

        Oops! The page you&apos;re looking for doesn&apos;t exist or has been moved.

      </p>

      <Link href="/">

        <Button>Return Home</Button>

      </Link>

    </div>

  );

}

App/page.jsx:

import HeroSection from "@/components/hero";

import { features } from "@/data/features";

import { Card ,CardHeader ,CardTitle , CardDescription , CardContent, CardFooter } from "@/components/ui/card";

import { faqs } from "@/data/faqs";

import { Accordion ,AccordionItem ,AccordionTrigger,AccordionContent } from "@/components/ui/accordion";

import Link from "next/link";

import { Button } from "@/components/ui/button";

import { Arrow } from "@radix-ui/react-dropdown-menu";

import { ArrowRight } from "lucide-react";

export default function Home() {

  return (

    <div>

     <div></div>

     <HeroSection />

      <section className="w-full py-12 md:py-24 lg:py-32 bg-background">

        <div className="container mx-auto px-4 md:px-6">

          <h2 className="text-3x1 font-bold tracking-tighter text-center mb-12">

            Various Features

          </h2>

          <div className="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-4 gap-6 max-w-6x1 mx-auto"

          >{features.map((feature,index)=>{

            return(

              <Card key={index} className="border-2 hover:border-primary transition-colors duration-300">

                <CardContent className="pt-6 text-center flex flex-col items-center">

                  <div className="flex flex-col items-center justify-center">

                    {feature.icon}

                    <h3 className="text-xl font-bold mb-2">

                      {feature.title}

                      </h3>

                    <p className="text-muted-forceground">{feature.description}</p>

                  </div>

                </CardContent>

              </Card>

            );

          })}</div>

        </div>

      </section>

      <section className="w-full py-12 md:py-24 lg:py-32 bg-muted/100">

        <div className="container mx-auto px-4 md:px-6">

          <div className="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-4 gap-6 max-w-6xl mx-auto">

           <div className="flex flex-col items-center justify-center space-y-2">

            <h3 className="text-4xl font-bold">50+</h3>

            <p className="text-muted-foreground">Industries Covered</p>

           </div>

           <div className="flex flex-col items-center justify-center space-y-2">

            <h3 className="text-4xl font-bold">200+</h3>

            <p className="text-muted-foreground">Interview Questions</p>

           </div>

           <div className="flex flex-col items-center justify-center space-y-2">

            <h3 className="text-4xl font-bold">24/7</h3>

            <p className="text-muted-foreground">AI Support</p>

           </div>

           <div className="flex flex-col items-center justify-center space-y-2">

            <h3 className="text-4xl font-bold">100+</h3>

            <p className="text-muted-foreground">Courses</p>

           </div>

          </div>

        </div>

      </section>

      <section className="w-full py-12 md:py-24 lg:py-32 bg-background">

        <div className="container mx-auto px-4 md:px-6">

        <div className="text-center max-w-3xl mx-auto mb-12">

         <h2 className="text-3xl font-bold mb-4">Frequently Asked Questions

         </h2>

         <p className="text-muted-foreground">Find answers to most common questions about our platform.

         </p>

         <div className="max-w-6xl mx-auto">

         <Accordion type="single" collapsible>

          {faqs.map((faq,index)=>{

              return(

              <AccordionItem key={index} value={`item-${index}`}>

              <AccordionTrigger>{faq.question}</AccordionTrigger>

              <AccordionContent>

               {faq.answer}

              </AccordionContent>

            </AccordionItem> );

          })}

           </Accordion>

         </div>

        </div>

        </div>

      </section>

      <section className="w-full bg-white">

        <div className="mx-auto py-24 gradient rounded-lg">

        <div className="flex flex-col items-center justify-center space-y-4 text-center max-w-3xl mx-auto">

         <h2 className="text-3x1 font-bold tracking-tighter text-primary-foreground sm:text-4xl md:text-5xl text-black">Ready to Accelerate Your Career?

         </h2>

         <p className="mx-auto max-w-[600px] text-primary-foreground/80 md:text-xl">Join us and grow smartly with AI

         </p>

       <Link href="/dashboard" passHref>

       <Button

       size="lg"

       variant="secondary"

       className="h-11 mt-5 animate-bounce"

       >

        Start Your Journey Today <ArrowRight className="ml-2 h-4 w-4"/>

       </Button>

       </Link>

        </div>

        </div>

      </section>

    </div>

  );

}

**Components folder:**

**Components/ui:**

1. components\ui\accordion.jsx:

"use client"

import \* as React from "react"

import \* as AccordionPrimitive from "@radix-ui/react-accordion"

import { ChevronDownIcon } from "lucide-react"

import { cn } from "@/lib/utils"

function Accordion({

...props

}) {

return <AccordionPrimitive.Root data-slot="accordion" {...props} />;

}

function AccordionItem({

className,

...props

}) {

return (

<AccordionPrimitive.Item

data-slot="accordion-item"

className={cn("border-b last:border-b-0", className)}

{...props} />

);

}

function AccordionTrigger({

className,

children,

...props

}) {

return (

<AccordionPrimitive.Header className="flex">

<AccordionPrimitive.Trigger

data-slot="accordion-trigger"

className={cn(

"focus-visible:border-ring focus-visible:ring-ring/50 flex flex-1 items-start justify-between gap-4 rounded-md py-4 text-left text-sm font-medium transition-all outline-none hover:underline focus-visible:ring-[3px] disabled:pointer-events-none disabled:opacity-50 [&[data-state=open]>svg]:rotate-180",

className

)}

{...props}>

{children}

<ChevronDownIcon

className="text-muted-foreground pointer-events-none size-4 shrink-0 translate-y-0.5 transition-transform duration-200" />

</AccordionPrimitive.Trigger>

</AccordionPrimitive.Header>

);

}

function AccordionContent({

className,

children,

...props

}) {

return (

<AccordionPrimitive.Content

data-slot="accordion-content"

className="data-[state=closed]:animate-accordion-up data-[state=open]:animate-accordion-down overflow-hidden text-sm"

{...props}>

<div className={cn("pt-0 pb-4", className)}>{children}</div>

</AccordionPrimitive.Content>

);

}

export { Accordion, AccordionItem, AccordionTrigger, AccordionContent }

1. components\ui\alert-dialog.jsx:

"use client"

import \* as React from "react"

import \* as AlertDialogPrimitive from "@radix-ui/react-alert-dialog"

import { cn } from "@/lib/utils"

import { buttonVariants } from "@/components/ui/button"

function AlertDialog({

...props

}) {

return <AlertDialogPrimitive.Root data-slot="alert-dialog" {...props} />;

}

function AlertDialogTrigger({

...props

}) {

return (<AlertDialogPrimitive.Trigger data-slot="alert-dialog-trigger" {...props} />);

}

function AlertDialogPortal({

...props

}) {

return (<AlertDialogPrimitive.Portal data-slot="alert-dialog-portal" {...props} />);

}

function AlertDialogOverlay({

className,

...props

}) {

return (

<AlertDialogPrimitive.Overlay

data-slot="alert-dialog-overlay"

className={cn(

"data-[state=open]:animate-in data-[state=closed]:animate-out data-[state=closed]:fade-out-0 data-[state=open]:fade-in-0 fixed inset-0 z-50 bg-black/80",

className

)}

{...props} />

);

}

function AlertDialogContent({

className,

...props

}) {

return (

<AlertDialogPortal>

<AlertDialogOverlay />

<AlertDialogPrimitive.Content

data-slot="alert-dialog-content"

className={cn(

"bg-background data-[state=open]:animate-in data-[state=closed]:animate-out data-[state=closed]:fade-out-0 data-[state=open]:fade-in-0 data-[state=closed]:zoom-out-95 data-[state=open]:zoom-in-95 fixed top-[50%] left-[50%] z-50 grid w-full max-w-[calc(100%-2rem)] translate-x-[-50%] translate-y-[-50%] gap-4 rounded-lg border p-6 shadow-lg duration-200 sm:max-w-lg",

className

)}

{...props} />

</AlertDialogPortal>

);

}

function AlertDialogHeader({

className,

...props

}) {

return (

<div

data-slot="alert-dialog-header"

className={cn("flex flex-col gap-2 text-center sm:text-left", className)}

{...props} />

);

}

function AlertDialogFooter({

className,

...props

}) {

return (

<div

data-slot="alert-dialog-footer"

className={cn("flex flex-col-reverse gap-2 sm:flex-row sm:justify-end", className)}

{...props} />

);

}

function AlertDialogTitle({

className,

...props

}) {

return (

<AlertDialogPrimitive.Title

data-slot="alert-dialog-title"

className={cn("text-lg font-semibold", className)}

{...props} />

);

}

function AlertDialogDescription({

className,

...props

}) {

return (

<AlertDialogPrimitive.Description

data-slot="alert-dialog-description"

className={cn("text-muted-foreground text-sm", className)}

{...props} />

);

}

function AlertDialogAction({

className,

...props

}) {

return (<AlertDialogPrimitive.Action className={cn(buttonVariants(), className)} {...props} />);

}

function AlertDialogCancel({

className,

...props

}) {

return (

<AlertDialogPrimitive.Cancel

className={cn(buttonVariants({ variant: "outline" }), className)}

{...props} />

);

}

export {

AlertDialog,

AlertDialogPortal,

AlertDialogOverlay,

AlertDialogTrigger,

AlertDialogContent,

AlertDialogHeader,

AlertDialogFooter,

AlertDialogTitle,

AlertDialogDescription,

AlertDialogAction,

AlertDialogCancel,

}

1. components\ui\badge.jsx:

import \* as React from "react"

import { Slot } from "@radix-ui/react-slot"

import { cva } from "class-variance-authority";

import { cn } from "@/lib/utils"

const badgeVariants = cva(

"inline-flex items-center justify-center rounded-md border px-2 py-0.5 text-xs font-medium w-fit whitespace-nowrap shrink-0 [&>svg]:size-3 gap-1 [&>svg]:pointer-events-none focus-visible:border-ring focus-visible:ring-ring/50 focus-visible:ring-[3px] aria-invalid:ring-destructive/20 dark:aria-invalid:ring-destructive/40 aria-invalid:border-destructive transition-[color,box-shadow] overflow-hidden",

{

variants: {

variant: {

default:

"border-transparent bg-primary text-primary-foreground [a&]:hover:bg-primary/90",

secondary:

"border-transparent bg-secondary text-secondary-foreground [a&]:hover:bg-secondary/90",

destructive:

"border-transparent bg-destructive text-white [a&]:hover:bg-destructive/90 focus-visible:ring-destructive/20 dark:focus-visible:ring-destructive/40",

outline:

"text-foreground [a&]:hover:bg-accent [a&]:hover:text-accent-foreground",

},

},

defaultVariants: {

variant: "default",

},

}

)

function Badge({

className,

variant,

asChild = false,

...props

}) {

const Comp = asChild ? Slot : "span"

return (

<Comp

data-slot="badge"

className={cn(badgeVariants({ variant }), className)}

{...props} />

);

}

export { Badge, badgeVariants }

1. Ccomponents\ui\button.jsx:

import \* as React from "react"

import { Slot } from "@radix-ui/react-slot"

import { cva } from "class-variance-authority";

import { cn } from "@/lib/utils"

const buttonVariants = cva(

"inline-flex items-center justify-center gap-2 whitespace-nowrap rounded-md text-sm font-medium transition-[color,box-shadow] disabled:pointer-events-none disabled:opacity-50 [&\_svg]:pointer-events-none [&\_svg:not([class\*='size-'])]:size-4 shrink-0 [&\_svg]:shrink-0 outline-none focus-visible:border-ring focus-visible:ring-ring/50 focus-visible:ring-[3px] aria-invalid:ring-destructive/20 dark:aria-invalid:ring-destructive/40 aria-invalid:border-destructive",

{

variants: {

variant: {

default:

"bg-primary text-primary-foreground shadow-xs hover:bg-primary/90",

destructive:

"bg-destructive text-white shadow-xs hover:bg-destructive/90 focus-visible:ring-destructive/20 dark:focus-visible:ring-destructive/40",

outline:

"border border-input bg-background shadow-xs hover:bg-accent hover:text-accent-foreground",

secondary:

"bg-secondary text-secondary-foreground shadow-xs hover:bg-secondary/80",

ghost: "hover:bg-accent hover:text-accent-foreground",

link: "text-primary underline-offset-4 hover:underline",

},

size: {

default: "h-9 px-4 py-2 has-[>svg]:px-3",

sm: "h-8 rounded-md gap-1.5 px-3 has-[>svg]:px-2.5",

lg: "h-10 rounded-md px-6 has-[>svg]:px-4",

icon: "size-9",

},

},

defaultVariants: {

variant: "default",

size: "default",

},

}

)

function Button({

className,

variant,

size,

asChild = false,

...props

}) {

const Comp = asChild ? Slot : "button"

return (

<Comp

data-slot="button"

className={cn(buttonVariants({ variant, size, className }))}

{...props} />

);

}

export { Button, buttonVariants }

1. Components/ui/card/jsx:

import \* as React from "react"

import { cn } from "@/lib/utils"

function Card({

className,

...props

}) {

return (

<div

data-slot="card"

className={cn(

"bg-card text-card-foreground flex flex-col gap-6 rounded-xl border py-6 shadow-sm",

className

)}

{...props} />

);

}

function CardHeader({

className,

...props

}) {

return (

<div

data-slot="card-header"

className={cn("flex flex-col gap-1.5 px-6", className)}

{...props} />

);

}

function CardTitle({

className,

...props

}) {

return (

<div

data-slot="card-title"

className={cn("leading-none font-semibold", className)}

{...props} />

);

}

function CardDescription({

className,

...props

}) {

return (

<div

data-slot="card-description"

className={cn("text-muted-foreground text-sm", className)}

{...props} />

);

}

function CardContent({

className,

...props

}) {

return (<div data-slot="card-content" className={cn("px-6", className)} {...props} />);

}

function CardFooter({

className,

...props

}) {

return (

<div

data-slot="card-footer"

className={cn("flex items-center px-6", className)}

{...props} />

);

}

export { Card, CardHeader, CardFooter, CardTitle, CardDescription, CardContent }

others are components/ ui/ button.jsx, card.jsx, dialog.jsx, dropdown.jsx, dropdown-menu.jsx, input .jsx, label.jsx, progress.jsx, radio-group,jsx, select.jsx, sonner.jsx, tabs.jsx, textarea.jsx

components/header.jsx then we have components/ hero.jsx and compoennets/ theme-provider.jsx

data folder has:

faqs.jsx, features.jsx, testimonials.jsx

\hooks\use-fetch.js:

import { useState } from "react";

import { toast } from "sonner";

import { useEffect } from "react";

const useFetch = (cb) => {

  const [data, setData] = useState(null);

  const [loading, setLoading] = useState(false);

  const [error, setError] = useState(null);

  // Accept arguments to be passed to the callback

  const fn = async (...args) => {

    setLoading(true);

    setError(null);

    try {

      const response = await cb(...args);

      setData(response);

      setError(null);

    } catch (error) {

      setError(error);

      toast.error(error.message);

    } finally {

      setLoading(false);

    }

  };

  return { data, loading, error, fn, setData };

};

export default useFetch;

C:\Users\APOORVA\Desktop\innov\careerflow-master\lib\checkUser.js:

import { currentUser } from "@clerk/nextjs/server";

import { db} from "./prisma";

export const checkUser = async()=>{

    const user = await currentUser();

    if(!user){

        return null;

    }

    try{

        const loggedInUser = await db.user.findUnique({

            where:{

                clerkUserId: user.id,

            },

        });

        if(loggedInUser){

            return loggedInUser;

        }

        const name = `${user.firstName} ${user.lastName}`;

        const newUser = await db.user.create({

            data:{

                 clerkUserId: user.id,

                 name,

                 imageUrl: user.imageUrl,

                 email: user.emailAddresses[0].emailAddress,

            },

        });

        return newUser;

    }catch (error){

        console.error(error.message);

    }

}

Lib/inngest/prisma.js:

import { PrismaClient } from "@prisma/client";

export const db = new PrismaClient();

if(process.env.NODE\_ENV!=="production"){

    globalThis.prisma=db;

}

Lib/utils.js:

import { clsx } from "clsx";

import { twMerge } from "tailwind-merge"

export function cn(...inputs) {

  return twMerge(clsx(inputs));

}

Then we have the node folder with the default node modules

Then the prisma folder:

prisma\migrations\20250307145246\_create\_models\migration.sql:

-- CreateEnum

CREATE TYPE "DemandLevel" AS ENUM ('HIGH', 'MEDIUM', 'LOW');

-- CreateEnum

CREATE TYPE "MarketOutlook" AS ENUM ('POSITIVE', 'NEUTRAL', 'NEGATIVE');

-- CreateTable

CREATE TABLE "User" (

    "id" TEXT NOT NULL,

    "clerkUserId" TEXT NOT NULL,

    "email" TEXT NOT NULL,

    "name" TEXT,

    "imageUrl" TEXT,

    "industry" TEXT,

    "createdAt" TIMESTAMP(3) NOT NULL DEFAULT CURRENT\_TIMESTAMP,

    "updatedAt" TIMESTAMP(3) NOT NULL,

    "bio" TEXT,

    "experience" INTEGER,

    "skills" TEXT[],

    CONSTRAINT "User\_pkey" PRIMARY KEY ("id")

);

-- CreateTable

CREATE TABLE "Assessment" (

    "id" TEXT NOT NULL,

    "userId" TEXT NOT NULL,

    "quizScore" DOUBLE PRECISION NOT NULL,

    "questions" JSONB[],

    "category" TEXT NOT NULL,

    "improvementTip" TEXT,

    "createdAt" TIMESTAMP(3) NOT NULL DEFAULT CURRENT\_TIMESTAMP,

    "updatedAt" TIMESTAMP(3) NOT NULL,

    CONSTRAINT "Assessment\_pkey" PRIMARY KEY ("id")

);

-- CreateTable

CREATE TABLE "Resume" (

    "id" TEXT NOT NULL,

    "userId" TEXT NOT NULL,

    "content" TEXT NOT NULL,

    "atsScore" DOUBLE PRECISION,

    "feedback" TEXT,

    "createdAt" TIMESTAMP(3) NOT NULL DEFAULT CURRENT\_TIMESTAMP,

    "updatedAt" TIMESTAMP(3) NOT NULL,

    CONSTRAINT "Resume\_pkey" PRIMARY KEY ("id")

);

-- CreateTable

CREATE TABLE "CoverLetter" (

    "id" TEXT NOT NULL,

    "userId" TEXT NOT NULL,

    "content" TEXT NOT NULL,

    "jobDescription" TEXT,

    "companyName" TEXT NOT NULL,

    "jobTitle" TEXT NOT NULL,

    "createdAt" TIMESTAMP(3) NOT NULL DEFAULT CURRENT\_TIMESTAMP,

    "updatedAt" TIMESTAMP(3) NOT NULL,

    CONSTRAINT "CoverLetter\_pkey" PRIMARY KEY ("id")

);

-- CreateTable

CREATE TABLE "IndustryInsight" (

    "id" TEXT NOT NULL,

    "industry" TEXT NOT NULL,

    "salaryRanges" JSONB[],

    "growthRate" DOUBLE PRECISION NOT NULL,

    "demandLevel" "DemandLevel" NOT NULL,

    "topSkills" TEXT[],

    "marketOutlook" "MarketOutlook" NOT NULL,

    "keyTrends" TEXT[],

    "recommendedSkills" TEXT[],

    "lastUpdated" TIMESTAMP(3) NOT NULL DEFAULT CURRENT\_TIMESTAMP,

    "nextUpdate" TIMESTAMP(3) NOT NULL,

    CONSTRAINT "IndustryInsight\_pkey" PRIMARY KEY ("id")

);

-- CreateIndex

CREATE UNIQUE INDEX "User\_clerkUserId\_key" ON "User"("clerkUserId");

-- CreateIndex

CREATE UNIQUE INDEX "User\_email\_key" ON "User"("email");

-- CreateIndex

CREATE INDEX "Assessment\_userId\_idx" ON "Assessment"("userId");

-- CreateIndex

CREATE UNIQUE INDEX "Resume\_userId\_key" ON "Resume"("userId");

-- CreateIndex

CREATE INDEX "CoverLetter\_userId\_idx" ON "CoverLetter"("userId");

-- CreateIndex

CREATE UNIQUE INDEX "IndustryInsight\_industry\_key" ON "IndustryInsight"("industry");

-- CreateIndex

CREATE INDEX "IndustryInsight\_industry\_idx" ON "IndustryInsight"("industry");

-- AddForeignKey

ALTER TABLE "User" ADD CONSTRAINT "User\_industry\_fkey" FOREIGN KEY ("industry") REFERENCES "IndustryInsight"("industry") ON DELETE SET NULL ON UPDATE CASCADE;

-- AddForeignKey

ALTER TABLE "Assessment" ADD CONSTRAINT "Assessment\_userId\_fkey" FOREIGN KEY ("userId") REFERENCES "User"("id") ON DELETE RESTRICT ON UPDATE CASCADE;

-- AddForeignKey

ALTER TABLE "Resume" ADD CONSTRAINT "Resume\_userId\_fkey" FOREIGN KEY ("userId") REFERENCES "User"("id") ON DELETE RESTRICT ON UPDATE CASCADE;

-- AddForeignKey

ALTER TABLE "CoverLetter" ADD CONSTRAINT "CoverLetter\_userId\_fkey" FOREIGN KEY ("userId") REFERENCES "User"("id") ON DELETE RESTRICT ON UPDATE CASCADE;

prisma\migrations\20250308184952\_\migration.sql:

-- AlterTable

ALTER TABLE "CoverLetter" ADD COLUMN     "status" TEXT DEFAULT 'draft';

Migrations\_lock.toml:

# Please do not edit this file manually

# It should be added in your version-control system (e.g., Git)

Provider = "postgresql"

Prisma/schema.prisma:

// This is your Prisma schema file,

// learn more about it in the docs: https://pris.ly/d/prisma-schema

// Looking for ways to speed up your queries, or scale easily with your serverless or edge functions?

// Try Prisma Accelerate: https://pris.ly/cli/accelerate-init

generator client {

provider = "prisma-client-js"

}

datasource db {

provider = "postgresql"

url = env("DATABASE\_URL")

}

model User {

id String @id @default(uuid())

clerkUserId String @unique

email String @unique

name String?

imageUrl String?

industry String?

industryInsight IndustryInsight? @relation(fields: [industry], references: [industry])

createdAt DateTime @default(now())

updatedAt DateTime @updatedAt

bio String?

experience Int?

skills String[]

assessments Assessment[]

resume Resume?

coverLetter CoverLetter[]

}

model Assessment {

id String @id @default(cuid())

userId String

user User @relation(fields: [userId], references: [id])

quizScore Float

questions Json[]

category String

improvementTip String?

createdAt DateTime @default(now())

updatedAt DateTime @updatedAt

@@index([userId])

}

model Resume {

id String @id @default(cuid())

userId String @unique

user User @relation(fields: [userId], references: [id])

content String @db.Text

atsScore Float?

feedback String?

createdAt DateTime @default(now())

updatedAt DateTime @updatedAt

}

model CoverLetter {

id String @id @default(cuid())

userId String

user User @relation(fields: [userId], references: [id])

content String // Markdown content

jobDescription String?

companyName String // Name of the company applying to

jobTitle String // Position applying for

status String @default("completed") // draft, completed

createdAt DateTime @default(now())

updatedAt DateTime @updatedAt

@@index([userId])

}

model IndustryInsight {

id String @id @default(cuid())

industry String @unique

users User[]

salaryRanges Json[]

growthRate Float

demandLevel DemandLevel

topSkills String[]

marketOutlook MarketOutlook

keyTrends String[]

recommendedSkills String[]

lastUpdated DateTime @default(now())

nextUpdate DateTime

@@index([industry])

}

enum DemandLevel{

HIGH

MEDIUM

LOW

}

enum MarketOutlook{

POSITIVE

NEUTRAL

NEGATIVE

}

**GROK CHATLINK:**

https://grok.com/chat/bf38917d-4691-488a-8e65-39e134c624c4