BUILDWISE-WEBSITE FOLDER STRUCTURE TREE:

1. buildwise-website/prisma/schema.prisma

schema.prisma

generator client {

provider = "prisma-client-js"

}

datasource db {

provider = "mysql"

url = env("DATABASE\_URL")

}

model AdminCompanyApproval {

idAdminCompanyApproval Int @id @default(autoincrement())

company\_id Int

status AdminCompanyApproval\_status

Company Company @relation(fields: [company\_id], references: [idCompany], onDelete: NoAction, onUpdate: NoAction, map: "approval\_company\_id")

@@index([company\_id], map: "approval\_company\_id")

}

model AdminReviewApproval {

idAdminReviewApproval Int @id @default(autoincrement())

review\_id Int

status AdminReviewApproval\_status

Review Review @relation(fields: [review\_id], references: [idReview], onDelete: NoAction, onUpdate: NoAction, map: "approval\_review\_id")

@@index([review\_id], map: "approval\_review\_id")

}

model AdminUserApproval {

idAdminUserApproval Int @id @default(autoincrement())

user\_id Int

status AdminUserApproval\_status

User User @relation(fields: [user\_id], references: [idUser], onDelete: NoAction, onUpdate: NoAction, map: "approval\_user\_id")

@@index([user\_id], map: "user\_id\_idx")

}

model Certification {

idCertification Int @id @default(autoincrement())

company\_id Int

certification\_name String @db.VarChar(255)

Company Company @relation(fields: [company\_id], references: [idCompany], onDelete: NoAction, onUpdate: NoAction, map: "certification\_company\_id")

@@index([company\_id], map: "certification\_company\_id")

}

model Company {

idCompany Int @id @default(autoincrement())

user\_id Int

name String @db.VarChar(255)

logo Bytes? @db.MediumBlob

about String? @db.Text

company\_type String? @db.VarChar(100)

year\_established Int? @db.Year

adress String? @db.Text

phone String? @db.VarChar(20)

email String? @db.VarChar(100)

website\_url String? @db.VarChar(255)

social\_media Json?

geographical\_availability String? @db.Text

specialization\_id Int

AdminCompanyApproval AdminCompanyApproval[]

Certification Certification[]

Specialization Specialization @relation(fields: [specialization\_id], references: [idSpecialization], onDelete: NoAction, onUpdate: NoAction, map: "specialization\_id")

User User @relation(fields: [user\_id], references: [idUser], onDelete: NoAction, onUpdate: NoAction, map: "user\_id")

Project Project[]

Review Review[]

Service Service[]

@@index([user\_id], map: "user\_id\_idx")

@@index([specialization\_id], map: "specialization\_id\_idx")

}

model Project {

idProject Int @id @default(autoincrement())

company\_id Int

name String @db.VarChar(255)

description String? @db.Text

project\_type String? @db.VarChar(100)

location String? @db.VarChar(255)

start\_date DateTime? @db.Date

end\_date DateTime? @db.Date

value Decimal? @db.Decimal(15, 2)

images Bytes? @db.MediumBlob

video Bytes?

social\_media Json?

Company Company @relation(fields: [company\_id], references: [idCompany], onDelete: NoAction, onUpdate: NoAction, map: "project\_company\_id")

@@index([company\_id], map: "project\_company\_id")

}

model Review {

idReview Int @id @default(autoincrement())

client\_id Int

company\_id Int

rating Int

review\_text String? @db.Text

approved Int? @default(0) @db.TinyInt

AdminReviewApproval AdminReviewApproval[]

User User @relation(fields: [client\_id], references: [idUser], onDelete: NoAction, onUpdate: NoAction, map: "client\_id")

Company Company @relation(fields: [company\_id], references: [idCompany], onDelete: NoAction, onUpdate: NoAction, map: "review\_company\_id")

@@index([client\_id], map: "client\_id\_idx")

@@index([company\_id], map: "review\_company\_id")

}

model Role {

idRole Int @id @default(autoincrement())

role\_name String @unique(map: "role\_name\_UNIQUE") @db.VarChar(50)

User User[]

}

model Service {

idService Int @id @default(autoincrement())

company\_id Int

service\_name String @db.VarChar(255)

Company Company @relation(fields: [company\_id], references: [idCompany], onDelete: NoAction, onUpdate: NoAction, map: "service\_company\_id")

@@index([company\_id], map: "service\_company\_id")

}

model Specialization {

idSpecialization Int @id @default(autoincrement())

name String @unique(map: "name\_UNIQUE") @db.VarChar(100)

Company Company[]

}

model User {

idUser Int @id @default(autoincrement())

username String @unique(map: "username\_UNIQUE") @db.VarChar(50)

password\_hash String @db.VarChar(255)

email String @unique(map: "email\_UNIQUE") @db.VarChar(100)

role\_id Int

approved Int? @default(0) @db.TinyInt

AdminUserApproval AdminUserApproval[]

Company Company[]

Review Review[]

Role Role @relation(fields: [role\_id], references: [idRole], onDelete: NoAction, onUpdate: NoAction, map: "role\_id")

@@index([role\_id], map: "role\_id\_idx")

}

enum AdminCompanyApproval\_status {

pending

approved

rejected

}

enum AdminReviewApproval\_status {

pending

approved

rejected

}

enum AdminUserApproval\_status {

pending

approved

rejected

}

1. buildwise-website/src/app/api/auth/login/route.ts

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

import bcrypt from "bcryptjs";

import { NextResponse } from "next/server";

const prisma = new PrismaClient();

export async function POST(req: Request) {

try {

const { email, password } = await req.json();

// Find user by email

const user = await prisma.user.findUnique({

where: { email },

});

if (!user) {

return NextResponse.json(

{ message: "User not found" },

{ status: 404 }

);

}

// Check password

const isValid = await bcrypt.compare(password, user.password\_hash);

if (!isValid) {

return NextResponse.json(

{ message: "Invalid credentials" },

{ status: 401 }

);

}

// Return success response (you'd normally set a session or JWT)

return NextResponse.json(

{ message: "Login successful", user: { id: user.idUser, role: user.role\_id } },

{ status: 200 }

);

} catch (error) {

return NextResponse.json(

{ message: "Internal server error", error },

{ status: 500 }

);

}

}

1. buildwise-website/src/app/api/auth/signup/route.ts

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

import bcrypt from "bcryptjs";

import { NextResponse } from "next/server";

const prisma = new PrismaClient();

export async function POST(req: Request) {

try {

const { email, password, confirmPassword, username } = await req.json();

// Validate passwords match

if (password !== confirmPassword) {

return NextResponse.json(

{ message: "Passwords do not match" },

{ status: 400 }

);

}

// Check if user already exists

const existingUser = await prisma.user.findUnique({ where: { email } });

if (existingUser) {

return NextResponse.json(

{ message: "User already exists" },

{ status: 409 }

);

}

// Hash password

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

// Create new user

const newUser = await prisma.user.create({

data: {

email,

password\_hash: hashedPassword,

role\_id: 3, // Default to "Client"

username,

},

});

return NextResponse.json(

{ message: "User created successfully", user: newUser },

{ status: 201 }

);

} catch (error) {

return NextResponse.json(

{ message: "Internal server error", error },

{ status: 500 }

);

}

}

1. buildwise-website/src/app/api/companies/add/route.ts

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

import { NextResponse } from "next/server";

const prisma = new PrismaClient();

export async function POST(req: Request) {

try {

const {

user\_id,

name,

about,

email,

phone,

website\_url,

adress,

company\_type,

year\_established,

geographical\_availability,

social\_media,

specialization\_id, // New required field

} = await req.json();

// Create a new company profile

const newCompany = await prisma.company.create({

data: {

user\_id,

name,

about,

email,

phone,

website\_url,

adress,

company\_type,

year\_established,

geographical\_availability,

social\_media,

specialization\_id, // Linking specialization

},

});

// Add an approval request entry

await prisma.adminCompanyApproval.create({

data: {

company\_id: newCompany.idCompany,

status: "pending",

},

});

return NextResponse.json(

{ message: "Company profile submitted for approval", company: newCompany },

{ status: 201 }

);

} catch (error) {

console.error(error);

return NextResponse.json({ message: "Internal Server Error", error }, { status: 500 });

}

}

1. buildwise-website/src/app/api/companies/search/route.ts

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

import { NextResponse } from "next/server";

const prisma = new PrismaClient();

export async function GET(req: Request) {

try {

const url = new URL(req.url);

const query = url.searchParams.get("query") || "";

const companies = await prisma.company.findMany({

where: {

name: { contains: query },

// Only fetch companies that are approved

AdminCompanyApproval: {

some: { status: "approved" },

},

},

include: {

AdminCompanyApproval: true,

},

});

return NextResponse.json(companies, { status: 200 });

} catch (error) {

console.error(error);

return NextResponse.json({ message: "Internal Server Error", error }, { status: 500 });

}

}

1. buildwise-website/src/app/api/specialization/create/route.ts

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

import { NextResponse } from "next/server";

const prisma = new PrismaClient();

export async function POST(req: Request) {

try {

const { name } = await req.json();

// Check if the specialization already exists

const existingSpecialization = await prisma.specialization.findUnique({

where: { name },

});

if (existingSpecialization) {

return NextResponse.json(

{ message: "Specialization already exists", specialization: existingSpecialization },

{ status: 409 }

);

}

// Create a new specialization

const newSpecialization = await prisma.specialization.create({

data: { name },

});

return NextResponse.json(

{ message: "Specialization created successfully", specialization: newSpecialization },

{ status: 201 }

);

} catch (error) {

console.error(error);

return NextResponse.json({ message: "Internal Server Error", error }, { status: 500 });

}

}

1. buildwise-website/src/app/api/specialization/list/route.ts

//Fetch All Specializations

//Create an endpoint to retrieve all existing specializations so users can choose from them.

//GET Method

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

import { NextResponse } from "next/server";

const prisma = new PrismaClient();

export async function GET() {

try {

const specializations = await prisma.specialization.findMany();

return NextResponse.json(specializations, { status: 200 });

} catch (error) {

console.error(error);

return NextResponse.json({ message: "Internal Server Error", error }, { status: 500 });

}

}

1. buildwise-website/src/app/globals.css

@tailwind base;

@tailwind components;

@tailwind utilities;

:root {

--background: #ffffff;

--foreground: #171717;

}

@media (prefers-color-scheme: dark) {

:root {

--background: #0a0a0a;

--foreground: #ededed;

}

}

body {

color: var(--foreground);

background: var(--background);

font-family: Arial, Helvetica, sans-serif;

}

1. buildwise-website/src/app/layout.tsx

import type { Metadata } from "next";

import { Geist, Geist\_Mono } from "next/font/google";

import "./globals.css";

const geistSans = Geist({

variable: "--font-geist-sans",

subsets: ["latin"],

});

const geistMono = Geist\_Mono({

variable: "--font-geist-mono",

subsets: ["latin"],

});

export const metadata: Metadata = {

title: "Create Next App",

description: "Generated by create next app",

};

export default function RootLayout({

children,

}: Readonly<{

children: React.ReactNode;

}>) {

return (

<html lang="en">

<body

className={`${geistSans.variable} ${geistMono.variable} antialiased`}

>

{children}

</body>

</html>

);

}

1. buildwise-website/src/app/page.tsx

"use client";

import React, { useState } from "react";

export default function Home() {

const [isLogin, setIsLogin] = useState(true);

const [formData, setFormData] = useState({

email: "",

password: "",

confirmPassword: "",

username: "",

});

const handleInputChange = (e: React.ChangeEvent<HTMLInputElement>) => {

setFormData({ ...formData, [e.target.id]: e.target.value });

};

const handleSubmit = async (e: React.FormEvent) => {

e.preventDefault();

const endpoint = isLogin ? "/api/auth/login" : "/api/auth/signup";

const payload = isLogin

? { email: formData.email, password: formData.password }

: {

email: formData.email,

password: formData.password,

confirmPassword: formData.confirmPassword,

username: formData.username,

};

try {

const response = await fetch(endpoint, {

method: "POST",

headers: { "Content-Type": "application/json" },

body: JSON.stringify(payload),

});

const result = await response.json();

if (response.ok) {

alert(result.message);

console.log(result);

} else {

alert(result.message);

}

} catch (error) {

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

}

};

return (

<main className="flex flex-col items-center justify-center min-h-screen bg-gray-100 p-4">

<h1 className="text-4xl font-bold mb-6 text-center text-gray-800">

Welcome to BuildWise

</h1>

<p className="mb-6 text-center text-gray-600">

A platform where companies can showcase their portfolios and clients can

leave reviews!

</p>

<div className="w-full max-w-md bg-white p-6 rounded-lg shadow-md">

<h2 className="text-2xl font-semibold mb-4">

{isLogin ? "Login" : "Create Account"}

</h2>

<form onSubmit={handleSubmit}>

<div className="mb-4">

<label htmlFor="email" className="block text-gray-700">

Email

</label>

<input

id="email"

type="email"

value={formData.email}

onChange={handleInputChange}

className="w-full px-4 py-2 border rounded-lg focus:ring-2 focus:ring-blue-400"

required

/>

</div>

<div className="mb-4">

<label htmlFor="password" className="block text-gray-700">

Password

</label>

<input

id="password"

type="password"

value={formData.password}

onChange={handleInputChange}

className="w-full px-4 py-2 border rounded-lg focus:ring-2 focus:ring-blue-400"

required

/>

</div>

{!isLogin && (

<>

<div className="mb-4">

<label htmlFor="confirmPassword" className="block text-gray-700">

Confirm Password

</label>

<input

id="confirmPassword"

type="password"

value={formData.confirmPassword}

onChange={handleInputChange}

className="w-full px-4 py-2 border rounded-lg focus:ring-2 focus:ring-blue-400"

required

/>

</div>

<div className="mb-4">

<label htmlFor="username" className="block text-gray-700">

Username

</label>

<input

id="username"

type="text"

value={formData.username}

onChange={handleInputChange}

className="w-full px-4 py-2 border rounded-lg focus:ring-2 focus:ring-blue-400"

required

/>

</div>

</>

)}

<button

type="submit"

className="w-full bg-blue-500 text-white py-2 rounded-lg hover:bg-blue-600"

>

{isLogin ? "Login" : "Sign Up"}

</button>

</form>

<div className="mt-4 text-center">

<button

onClick={() => setIsLogin(!isLogin)}

className="text-blue-500 hover:underline"

>

{isLogin

? "Don't have an account? Create one"

: "Already have an account? Login"}

</button>

</div>

</div>

</main>

);

}

1. buildwise-website/src/components/SpecializationSelector.tsx

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

interface Specialization {

idSpecialization: number;

name: string;

}

interface Props {

onSpecializationChange: (id: number) => void;

}

const SpecializationSelector: React.FC<Props> = ({ onSpecializationChange }) => {

const [specializations, setSpecializations] = useState<Specialization[]>([]);

const [newSpecialization, setNewSpecialization] = useState("");

useEffect(() => {

async function fetchSpecializations() {

const response = await fetch("/api/specializations/list");

const data: Specialization[] = await response.json();

setSpecializations(data);

}

fetchSpecializations();

}, []);

const handleCreateSpecialization = async () => {

const response = await fetch("/api/specializations/create", {

method: "POST",

headers: { "Content-Type": "application/json" },

body: JSON.stringify({ name: newSpecialization }),

});

const data = await response.json();

if (response.ok) {

const newSpec: Specialization = data.specialization;

setSpecializations([...specializations, newSpec]);

setNewSpecialization("");

} else {

alert(data.message || "An error occurred.");

}

};

return (

<div>

<label htmlFor="specialization">Select or Create Specialization:</label>

<select

id="specialization"

onChange={(e) => onSpecializationChange(Number(e.target.value))}

>

<option value="">Select a specialization</option>

{specializations.map((spec) => (

<option key={spec.idSpecialization} value={spec.idSpecialization}>

{spec.name}

</option>

))}

</select>

<div>

<input

type="text"

placeholder="New Specialization"

value={newSpecialization}

onChange={(e) => setNewSpecialization(e.target.value)}

/>

<button type="button" onClick={handleCreateSpecialization}>

Add Specialization

</button>

</div>

</div>

);

};

export default SpecializationSelector;

1. buildwise-website/package.json

{

"name": "buildwise-website",

"version": "0.1.0",

"private": true,

"prisma": {

"seed": "node prisma/seed.js"

},

"scripts": {

"dev": "next dev",

"build": "next build",

"start": "next start",

"lint": "next lint"

},

"dependencies": {

"@prisma/client": "^6.1.0",

"bcryptjs": "^2.4.3",

"boostrap": "^2.0.0",

"bootstrap": "^5.3.3",

"mysql2": "^3.11.5",

"next": "15.1.0",

"prisma": "^6.1.0",

"react": "^19.0.0",

"react-dom": "^19.0.0"

},

"devDependencies": {

"@eslint/eslintrc": "^3",

"@types/bcryptjs": "^2.4.6",

"@types/node": "^20.17.10",

"@types/react": "^19",

"@types/react-dom": "^19",

"eslint": "^9",

"eslint-config-next": "15.1.0",

"postcss": "^8",

"tailwindcss": "^3.4.1",

"ts-node": "^10.9.2",

"typescript": "^5.7.2"

}