Portfolio Website - Project Plan

Portfolio Website - Comprehensive Project Plan

Repository: https://github.com/MoncyDev/Portfolio-Website **Primary Language:** TypeScript **Project Type:** Web Frontend (React + Three.js) **Complexity:** Complex **Last Updated:** June 2, 2025

Table of Contents

Project Summary & Goals

Overview

This is an open-source portfolio website built with modern web technologies, featuring interactive 3D elements, smooth animations, and responsive design. The project showcases advanced frontend development skills using React, Three.js, and GSAP.

Primary Goals

- **Showcase Portfolio:** Display projects, skills, and professional experience
- **Interactive Experience:** Provide engaging 3D visuals and animations
- **Performance:** Maintain fast loading times despite complex graphics
- **Accessibility:** Ensure usability across devices and accessibility standards
- **Modern Stack:** Demonstrate proficiency with cutting-edge web technologies

Target Audience

- Potential employers and clients
- Fellow developers seeking inspiration
- Students learning modern web development
- Anyone interested in interactive web experiences

Key Features & Use Cases

Core Features

Interactive 3D Graphics

- **Three.js Integration:** Complex 3D scenes and models
- **React Three Fiber:** Declarative 3D rendering in React
- **Post-processing Effects:** Advanced visual effects and shaders
- **Physics Simulation:** Realistic object interactions with Cannon.js

Advanced Animations

- **GSAP Integration:** Professional-grade animations
- **Smooth Transitions:** Seamless page and component transitions
- **Scroll-triggered Animations:** Interactive storytelling
- **Performance Optimized:** Hardware-accelerated animations

Responsive Design

- **Mobile-first Approach:** Optimized for all screen sizes
- **Touch Interactions:** Mobile-friendly 3D controls
- **Progressive Enhancement:** Graceful degradation for older devices

Modern Development Stack

- **TypeScript:** Type-safe development
- **Vite:** Fast build tool and development server
- **ESLint:** Code quality and consistency
- **Modern CSS:** Advanced styling techniques

Use Cases

1. **Portfolio Showcase**

Display projects with interactive previews

Highlight technical skills and achievements

Provide contact information and social links

1. **Educational Resource**

Demonstrate modern web development techniques

Show integration of multiple complex libraries

Provide code examples for learning

1. **Client Presentations**

Impress potential clients with interactive demos

Showcase technical capabilities

Provide professional online presence

Technology Stack

Frontend Framework

- **React 18+** - Component-based UI library
- **TypeScript** - Type-safe JavaScript development
- **Vite** - Next-generation frontend build tool

3D Graphics & Animation

- **Three.js** - 3D graphics library
- **@react-three/fiber** - React renderer for Three.js
- **@react-three/drei** - Useful helpers for React Three Fiber
- **@react-three/cannon** - Physics engine integration
- **@react-three/postprocessing** - Post-processing effects
- **@gsap/react** - Professional animation library

Development Tools

- **ESLint** - Code linting and quality
- **TypeScript Compiler** - Type checking and compilation
- **Vite Dev Server** - Fast development experience

File Breakdown

- **TypeScript:** 32 files (Primary development language)
- **CSS:** 13 files (Styling and animations)
- **JSON:** 5 files (Configuration and data)
- **JavaScript:** 3 files (Legacy or build scripts)
- **HTML:** 1 file (Entry point)
- **Markdown:** 1 file (Documentation)

Project Structure

‘ portfolio-website/ public/ # Static assets draco/ # Draco compression files for 3D models images/ # Image assets (14+ files) models/ # 3D model files (4+ files) src/ # Source code assets/ # Project assets components/ # React components (17+ files) context/ # React context providers data/ # Static data and configurations App.css # Main application styles App.tsx # Root React component index.css # Global styles main.tsx # Application entry point vite-env.d.ts # Vite type definitions eslint.config.js # ESLint configuration index.html # HTML entry point package.json # Dependencies and scripts package-lock.json # Dependency lock file tsconfig.*.json # TypeScript configurations vite.config.ts # Vite build configuration README.md # Project documentation ‘

Directory Descriptions

`/public/`

- **Static Assets:** Files served directly by the web server
- **draco/:** Compression library for optimizing 3D models
- **images/:** Portfolio images, backgrounds, and visual assets
- **models/:** 3D model files (likely .glb, .gltf formats)

`/src/`

- **Main Source Code:** All TypeScript/React development files
- **components/:** Reusable React components and UI elements
- **context/:** State management and data providers
- **data/:** Configuration files and static data

Major Components & Modules

Core Application Components

1. App.tsx - Root Component

- **Purpose:** Main application wrapper and routing
- **Responsibilities:**
- Initialize global state and context
- Set up routing and navigation
- Load global styles and configurations
- Handle error boundaries

2. Components Directory (17+ files) Based on typical portfolio structure, likely includes:

- **Header/Navigation**
- Responsive navigation menu
- Logo and branding
- Mobile hamburger menu

- **Hero Section**
- 3D animated introduction
- Interactive background elements
- Call-to-action buttons

- **Portfolio Gallery**
- Project showcase with 3D previews
- Filtering and categorization
- Modal overlays for detailed views

- **About Section**
- Personal information and bio
- Skills visualization
- Interactive elements

- **Contact Form**
- Form validation and submission
- Social media links
- Contact information

- **3D Scene Components**
- Three.js canvas setup
- 3D model loaders
- Animation controllers
- Physics interactions

3. Context Providers

- **Theme Context:** Dark/light mode switching
- **Animation Context:** Global animation state
- **Portfolio Context:** Project data management
- **User Interaction Context:** Mouse/touch tracking

4. Data Management

- **Project Data:** Portfolio project information
- **Skills Data:** Technical skills and proficiencies
- **Configuration:** Site settings and constants

3D Graphics Architecture

Three.js Integration

- **Scene Setup:** Camera, lighting, and renderer configuration
- **Model Loading:** Efficient 3D asset loading with Draco compression
- **Animation System:** GSAP-powered smooth animations
- **Physics Engine:** Realistic object interactions
- **Post-processing:** Visual effects and shaders

Performance Optimizations

- **Level of Detail (LOD):** Adaptive model complexity
- **Frustum Culling:** Render only visible objects
- **Texture Optimization:** Compressed and optimized images
- **Lazy Loading:** Load assets on demand

Setup Instructions

Prerequisites

Required Software ‘bash

Node.js (version 16 or higher)

node --version # Should be 16.0.0 or higher npm --version # Should be 8.0.0 or higher

Git for version control

git --version ‘

System Requirements

- **Operating System:** Windows 10+, macOS 10.15+, or Linux
- **RAM:** Minimum 8GB (16GB recommended for development)
- **Graphics:** WebGL-compatible graphics card
- **Browser:** Chrome 90+, Firefox 88+, Safari 14+, or Edge 90+

Step-by-Step Installation

1. Clone the Repository ‘bash

Clone the repository

git clone https://github.com/MoncyDev/Portfolio-Website.git

Navigate to project directory

cd Portfolio-Website

Verify project structure

ls -la ‘

2. Install Dependencies ‘bash

Install all npm dependencies

npm install

Verify installation

npm list --depth=0 ‘

3. Verify Installation ‘bash

Check if all dependencies are installed correctly

npm audit

Fix any vulnerabilities (if needed)

npm audit fix ‘

4. Environment Setup ‘bash

Create environment file (if needed)

cp .env.example .env

Edit environment variables

nano .env # or use your preferred editor ‘

Troubleshooting Installation

Common Issues Node Version Conflicts: ‘bash

Use Node Version Manager (nvm)

nvm install 18 nvm use 18 ‘ **Permission Errors:** ‘bash

Fix npm permissions (macOS/Linux)

sudo chown -R $(whoami) ~/.npm

Or use npx instead of global installs

npx create-react-app --version ‘ **Dependency Conflicts:** ‘bash

Clear npm cache

npm cache clean --force

Delete node_modules and reinstall

rm -rf node_modules package-lock.json npm install ‘

Configuration Required

Environment Variables

Create a `.env` file in the root directory:

‘bash

.env file

Development configuration

VITE_APP_TITLE=”Portfolio Website” VITE_APP_DESCRIPTION=”Interactive portfolio showcasing modern web development”

API endpoints (if applicable)

VITE_API_BASE_URL=”https://api.example.com”

Analytics (optional)

VITE_GOOGLE_ANALYTICS_ID=”GA_MEASUREMENT_ID”

Contact form (if using external service)

VITE_CONTACT_FORM_ENDPOINT=”https://formspree.io/f/your-form-id” VITE_EMAILJS_SERVICE_ID=”your_service_id” VITE_EMAILJS_TEMPLATE_ID=”your_templa VITE_EMAILJS_PUBLIC_KEY=”your_public_key”

Social media links

VITE_GITHUB_URL=”https://github.com/MoncyDev” VITE_LINKEDIN_URL=”https://linkedin.com/i VITE_TWITTER_URL=”https://twitter.com/yourhandle” ‘

TypeScript Configuration

The project includes multiple TypeScript configurations:

`tsconfig.json` - Base Configuration ‘json { ”compilerOptions”: { ”target”: ”ES2020”, ”lib”: [”ES2020”, ”DOM”, ”DOM.Iterable”], ”module”: ”ESNext”, ”skipLibCheck”: true, ”moduleResolution”: ”bundler”, ”allowImportingTsExtensions”: true, ”resolveJsonModule”: true, ”isolatedModules”: true, ”noEmit”: true, ”jsx”: ”react-jsx”, ”strict”: true, ”noUnusedLocals”: true, ”noUnusedParameters”: true, ”noFallthroughCasesInSwitch”: true } } ‘

Vite Configuration

`vite.config.ts` ‘typescript import { defineConfig } from 'vite' import react from '@vitejs/plugin-react'

export default defineConfig({ plugins: [react()], server: { port: 3000, open: true }, build: { outDir: 'dist', sourcemap: true }, optimizeDeps: { include: ['three', '@react-three/fiber', '@react-three/drei'] } }) ‘

ESLint Configuration

`eslint.config.js` Ensures code quality and consistency: ‘javascript export default { extends: ['eslint:recommended', '@typescript-eslint/recommended', 'plugin:react/recommended', 'plugin:react-hooks/recommended'], rules: { // Custom rules for the project } } ‘

Execution Plan

Development Mode

1. Start Development Server ‘bash

Start the development server

npm run dev

Alternative: Start with specific port

npm run dev -- --port 3000

Start with host binding (for network access)

npm run dev -- --host ‘

2. Verify Development Environment ‘bash

Open browser to development URL

open http://localhost:3000

Check console for any errors

Verify 3D elements are loading correctly

Test responsive design on different screen sizes

‘

3. Development Workflow ‘bash

Run linting

npm run lint

Fix linting issues automatically

npm run lint:fix

Type checking

npm run type-check

Run tests (if available)

npm test ‘

Production Build

1. Build for Production ‘bash

Create production build

npm run build

Verify build output

ls -la dist/

Check build size

du -sh dist/ ‘

2. Preview Production Build ‘bash

Preview production build locally

npm run preview

Test production build

open http://localhost:4173 ‘

3. Build Optimization ‘bash

Analyze bundle size

npm run build -- --analyze

Check for unused dependencies

npx depcheck

Optimize images (if needed)

npx imagemin-cli public/images/* --out-dir=public/images/optimized ‘

Testing Strategy

Manual Testing Checklist

- [ ] **Page Load:** All pages load without errors
- [ ] **3D Graphics:** Three.js scenes render correctly
- [ ] **Animations:** GSAP animations play smoothly
- [ ] **Responsive:** Works on mobile, tablet, and desktop
- [ ] **Performance:** Page loads in under 3 seconds
- [ ] **Accessibility:** Keyboard navigation works
- [ ] **Cross-browser:** Works in Chrome, Firefox, Safari, Edge

Performance Testing ‘bash

Lighthouse audit

npx lighthouse http://localhost:3000 --output html --output-path ./lighthouse-report.html

Bundle analyzer

npm install --save-dev webpack-bundle-analyzer npm run build && npx webpack-bundle-analyzer dist/static/js/*.js ‘

Development Workflow

Daily Development Process

1. Start Development Session ‘bash

Pull latest changes

git pull origin main

Install any new dependencies

npm install

Start development server

npm run dev ‘

2. Code Quality Checks ‘bash

Before committing changes

npm run lint npm run type-check npm test # if tests exist ‘

3. Git Workflow ‘bash

Create feature branch

git checkout -b feature/new-component

Make changes and commit

git add . git commit -m ”feat: add new portfolio component”

Push changes

git push origin feature/new-component ‘

Code Organization Best Practices

Component Structure ‘typescript // components/PortfolioItem/index.tsx import React from 'react' import { Canvas } from '@react-three/fiber' import './PortfolioItem.css'

interface PortfolioItemProps { title: string description: string modelPath: string }

export const PortfolioItem: React.FC = ({ title, description, modelPath }) => { return (

{/ *3D content* /}

{title}

{description}

) } ‘

File Naming Conventions

- **Components:** PascalCase (e.g., `PortfolioItem.tsx`)
- **Hooks:** camelCase with 'use' prefix (e.g., `useAnimation.ts`)
- **Utilities:** camelCase (e.g., `mathUtils.ts`)
- **Constants:** UPPER_SNAKE_CASE (e.g., `API_ENDPOINTS.ts`)

Deployment Checklist

Pre-deployment Preparation

1. Code Quality Verification

- [ ] All TypeScript errors resolved
- [ ] ESLint warnings addressed
- [ ] Code formatted consistently
- [ ] No console.log statements in production code
- [ ] All TODO comments resolved or documented

2. Performance Optimization

- [ ] Images optimized and compressed
- [ ] 3D models optimized with Draco compression
- [ ] Bundle size analyzed and optimized
- [ ] Lazy loading implemented for heavy components
- [ ] Service worker configured (if applicable)

3. Security Checklist

- [ ] No sensitive data in environment variables
- [ ] API keys properly secured
- [ ] HTTPS configured
- [ ] Content Security Policy (CSP) headers set
- [ ] Dependencies audited for vulnerabilities

Deployment Platforms

Vercel (Recommended) ‘bash

Install Vercel CLI

npm install -g vercel

Deploy to Vercel

vercel

Set environment variables

vercel env add VITE_API_BASE_URL ‘

Netlify ‘bash

Install Netlify CLI

npm install -g netlify-cli

Build and deploy

npm run build netlify deploy --prod --dir=dist ‘

GitHub Pages ‘bash

Install gh-pages

npm install --save-dev gh-pages

Add to package.json scripts

”deploy”: ”gh-pages -d dist”

Deploy

npm run build npm run deploy ‘

Post-deployment Verification

1. Functional Testing

- [ ] All pages load correctly
- [ ] 3D graphics render properly

- [ ] Animations work smoothly
- [ ] Contact form submits successfully
- [ ] All links work correctly

2. Performance Testing

- [ ] Page load time under 3 seconds
- [ ] Lighthouse score above 90
- [ ] Mobile performance optimized
- [ ] CDN properly configured

3. SEO and Analytics

- [ ] Meta tags properly set
- [ ] Open Graph tags configured
- [ ] Google Analytics tracking
- [ ] Search console verification

Troubleshooting & Tips

Common Development Issues

1. Three.js Related Problems **Issue:** 3D models not loading ‘bash

Check model file paths

ls -la public/models/

Verify model format compatibility

Ensure models are in .glb or .gltf format

‘ **Solution:** ‘typescript // Use proper model loading with error handling import { useGLTF } from '@react-three/drei'

function Model({ url }: { url: string }) { const { scene } = useGLTF(url) return }

// Preload models useGLTF.preload('/models/portfolio-item.glb') ‘

2. Performance Issues **Issue:** Slow rendering or frame drops ‘typescript // Optimize with React.memo const OptimizedComponent = React.memo(({ data }) => { return })

// Use useMemo for expensive calculations const expensiveValue = useMemo(() => { return heavyCalculation(props.data) }, [props.data]) ‘

Solution: ‘typescript // Implement level of detail (LOD) import { Lod } from '@react-three/drei' {/ *Close* /} {/ *Medium* /} {/ *Far* /} ‘

3. TypeScript Errors **Issue:** Three.js type definitions ‘bash

Install additional type definitions

npm install --save-dev @types/three ‘ **Solution:** ‘typescript // Create custom type definitions // types/three-extensions.d.ts declare module 'three/examples/jsm/loaders/GLTFLoader' { export class GLTFLoader { load(url: string, onLoad: (gltf: any) => void): void } } ‘

4. Build Issues **Issue:** Vite build failures ‘bash

Clear cache and rebuild

rm -rf node_modules/.vite npm run build ‘ **Solution:** ‘typescript // vite.config.ts - Add proper optimizations export default defineConfig({ optimizeDeps: { include: ['three', '@react-three/fiber', '@react-three/drei', 'gsap'] }, build: { rollupOptions: { output: { manualChunks: { vendor: ['react', 'react-dom'], three: ['three', '@react-three/fiber', '@react-three/drei'] } } } } }) ‘

Performance Optimization Tips

1. 3D Model Optimization ‘bash

Use Draco compression for models

Models should be under 1MB each

Use .glb format for better compression

Optimize textures

Use WebP format for images

Keep texture resolution reasonable (1024x1024 max)

‘

2. Code Splitting ‘typescript // Lazy load heavy components const Portfolio3D = lazy(() => import('./components/Portfolio3D'))

// Use Suspense for loading states }> ‘

3. Animation Performance ‘typescript // Use GSAP's performance optimizations gsap.set(element, { force3D: true }) // Hardware acceleration gsap.ticker.fps(60) // Limit frame rate if needed

// Batch DOM updates gsap.set([element1, element2, element3], { opacity: 0, y: 50 }) ‘

Browser Compatibility

Supported Browsers

- **Chrome:** 90+ (Recommended)
- **Firefox:** 88+
- **Safari:** 14+
- **Edge:** 90+

Fallbacks for Older Browsers ‘typescript // Check WebGL support function checkWebGLSupport() { try { const canvas = document.createElement('canvas') return !!(window.WebGLRenderingContext && canvas.getContext('webgl')) } catch (e) { return false } }

// Provide fallback content if (!checkWebGLSupport()) { // Show static images instead of 3D content } ‘

Debugging Tools

Development Tools ‘bash

React Developer Tools

Install browser extension for React debugging

Three.js Inspector

Add to development environment

npm install --save-dev three-devtools ‘

Performance Monitoring ‘typescript // Add performance monitoring const observer = new PerformanceObserver((list) => { for (const entry of list.getEntries()) { console.log('Performance:', entry) } }) observer.observe({ entryTypes: ['measure', 'navigation'] }) ‘

Performance Optimization

Bundle Size Optimization

1. Analyze Bundle Size ‘bash

Install bundle analyzer

npm install --save-dev rollup-plugin-visualizer

Add to vite.config.ts

import { visualizer } from 'rollup-plugin-visualizer'

export default defineConfig({ plugins: [react(), visualizer({ filename: 'dist/stats.html', open: true })] }) ‘

2. Code Splitting Strategies ‘typescript // Route-based splitting const Home = lazy(() => import('./pages/Home')) const Portfolio = lazy(() => import('./pages/Portfolio')) const Contact = lazy(() => import('./pages/Contact'))

// Component-based splitting const Heavy3DComponent = lazy(() => import('./components/Heavy3DComponent')) ‘

3. Tree Shaking ‘typescript // Import only what you need import { Vector3, Mesh } from 'three' // Instead of: import * as THREE from 'three'

// Use named imports for utilities import { debounce } from 'lodash-es' // Instead of: import _ from 'lodash' ‘

Runtime Performance

1. Memory Management ‘typescript // Dispose of Three.js resources useEffect(() => { return () => { // Cleanup geometries geometry.dispose() // Cleanup materials material.dispose() // Cleanup textures texture.dispose() // Remove from scene scene.remove(mesh) } }, []) ‘

2. Efficient Rendering ‘typescript // Use object pooling for frequently created objects const objectPool = new Array(100).fill(null).map(() => new Vector3())

// Reuse objects instead of creating new ones const getPooledVector = () => { return objectPool.pop() || new Vector3() } ‘

Contributing Guidelines

Development Standards

1. Code Style

- Follow TypeScript strict mode
- Use ESLint configuration provided
- Maintain consistent naming conventions
- Write self-documenting code with clear variable names

2. Component Guidelines ‘typescript // Component template import React, { useEffect, useRef } from 'react' import './ComponentName.css'

interface ComponentNameProps { // Define all props with types title: string isVisible?: boolean }

export const ComponentName: React.FC = ({ title, isVisible = true }) => { // Hooks at the top const ref = useRef(null) // Effects useEffect(() => { // Effect logic }, []) // Event handlers const handleClick = () => { // Handler logic } // Render return (

{/ *JSX content* /}

) } ‘

3. Git Workflow ‘bash

Feature development

git checkout -b feature/component-name git commit -m ”feat: add new component” git push origin feature/component-name

Bug fixes

git checkout -b fix/issue-description git commit -m ”fix: resolve rendering issue”

Documentation

git commit -m ”docs: update setup instructions” ‘

Pull Request Process

1. **Create Feature Branch:** Branch from main
2. **Implement Changes:** Follow coding standards
3. **Test Thoroughly:** Ensure no regressions
4. **Update Documentation:** Keep docs current
5. **Submit PR:** Include detailed description
6. **Code Review:** Address feedback promptly
7. **Merge:** Squash commits if needed

Project Plan Generated: June 2, 2025 **Last Updated:** June 2, 2025 **Version:** 1.0 *This comprehensive project plan provides everything needed to understand, set up, develop, and deploy the Portfolio Website project. For questions or contributions, please refer to the repository's issue tracker.*