# **WebDesign AI Generator - Project Documentation**

## **Project Overview**

WebDesign AI Generator is a web application that allows users to upload existing website designs, analyze them using AI (Gemini), and generate multiple new design variations based on the original design using Claude AI models. The application provides an intuitive interface for users to interact with the designs, expand on them with additional prompts, and save favorites.

## **Features**

1. **Design Upload**: Upload website design images
2. **AI Analysis**: Analyze website designs using Gemini or other AI models
3. **Design Generation**: Generate new designs based on analysis and user prompts
4. **AI Selection**: Choose different AI models for both analysis and generation phases
5. **Custom API Keys**: Use personal API keys for various AI services
6. **Design Interaction**: Inspect designs in detail, expand on specific designs, and save favorites
7. **Responsive UI**: Works across desktop and mobile devices

## **Tech Stack**

### **Frontend**

* **Framework**: React.js
* **Language**: TypeScript
* **Styling**: Tailwind CSS
* **State Management**: React Context API + localStorage

### **Backend**

* **Runtime**: Node.js
* **Framework**: Express.js
* **API Integration**: Anthropic Claude API, Google Gemini API

## **Project Structure**

src/

├── components/

│ ├── common/

│ │ ├── Button.tsx

│ │ ├── Dropdown.tsx

│ │ ├── Modal.tsx

│ │ ├── LoadingSpinner.tsx

│ │ └── ErrorBoundary.tsx

│ ├── layout/

│ │ ├── Header.tsx

│ │ ├── Footer.tsx

│ │ ├── Sidebar.tsx

│ │ └── Container.tsx

│ ├── upload/

│ │ ├── ImageUpload.tsx

│ │ ├── ImagePreview.tsx

│ │ └── UploadStatus.tsx

│ ├── prompt/

│ │ ├── PromptInput.tsx

│ │ ├── AISelector.tsx

│ │ └── PromptHistory.tsx

│ ├── designs/

│ │ ├── DesignGallery.tsx

│ │ ├── DesignCard.tsx

│ │ ├── DesignDetail.tsx

│ │ ├── DesignExpander.tsx

│ │ └── EmptyState.tsx

│ └── settings/

│ ├── SettingsPanel.tsx

│ ├── APIKeyInput.tsx

│ └── ModelSelection.tsx

├── hooks/

│ ├── useImageUpload.ts

│ ├── useAIServices.ts

│ ├── useSettings.ts

│ ├── useFavorites.ts

│ ├── useDesignHistory.ts

│ ├── useLocalStorage.ts

│ └── useMediaQuery.ts

├── services/

│ ├── ai/

│ │ ├── geminiService.ts

│ │ ├── claudeService.ts

│ │ ├── openaiService.ts

│ │ └── types.ts

│ ├── storage/

│ │ ├── favoriteService.ts

│ │ └── settingsService.ts

│ └── api.ts

├── utils/

│ ├── imageProcessing.ts

│ ├── promptEngineering.ts

│ ├── designParser.ts

│ ├── htmlRenderer.ts

│ ├── errorHandling.ts

│ └── validators.ts

├── contexts/

│ ├── SettingsContext.tsx

│ ├── DesignContext.tsx

│ ├── UploadContext.tsx

│ └── FavoritesContext.tsx

├── pages/

│ ├── Home.tsx

│ ├── Settings.tsx

│ ├── Favorites.tsx

│ ├── DesignDetail.tsx

│ └── ExpandDesign.tsx

├── types/

│ ├── design.ts

│ ├── api.ts

│ ├── settings.ts

│ └── index.ts

├── constants/

│ ├── aiModels.ts

│ ├── apiEndpoints.ts

│ ├── localStorage.ts

│ └── ui.ts

├── styles/

│ ├── tailwind.css

│ └── global.css

├── App.tsx

└── index.tsx

## **Detailed Component Specifications**

### **Core Components**

#### **ImageUpload.tsx**

* Drag-and-drop interface for website design uploads
* Image preview functionality
* File type validation (allow only image formats)
* Upload progress indicator
* Error handling for invalid files

#### **PromptInput.tsx**

* Text area for entering custom prompts
* Character counter
* Placeholder text with examples
* History of previous prompts (optional)
* Clear button

#### **AISelector.tsx**

* Dropdown for selecting AI models
* Separate selectors for analysis and generation phases
* Model description tooltips
* Indication of which models require API keys

#### **DesignGallery.tsx**

* Grid layout of generated designs
* Responsive grid (3 columns on desktop, 2 on tablet, 1 on mobile)
* Loading states for designs being generated
* Empty state when no designs are available
* Filter options (if implemented)

#### **DesignCard.tsx**

* Thumbnail preview of the design
* "Expand on Design" button
* "Save" button
* Click handler for detailed view
* Loading state

#### **DesignDetail.tsx**

* Full-size view of the design
* Rendered HTML/CSS preview
* Code view toggle
* Download options
* Back button
* Expand functionality

#### **SettingsPanel.tsx**

* API key management
* Default AI model selection
* Theme preferences (if implemented)
* Local storage or account settings

## **Context System**

### **SettingsContext**

Manages user preferences and API keys.

interface SettingsContextType {

apiKeys: {

claude: string;

gemini: string;

openai?: string;

};

defaultModels: {

analysis: string;

generation: string;

};

updateApiKey: (service: string, key: string) => void;

updateDefaultModel: (phase: string, model: string) => void;

clearSettings: () => void;

}

### **DesignContext**

Manages the current design session.

interface DesignContextType {

originalImage: File | null;

analysisResult: AnalysisResult | null;

generatedDesigns: Design[];

currentPrompt: string;

isAnalyzing: boolean;

isGenerating: boolean;

selectedModels: {

analysis: string;

generation: string;

};

setOriginalImage: (file: File | null) => void;

setCurrentPrompt: (prompt: string) => void;

analyzeImage: () => Promise<void>;

generateDesigns: () => Promise<void>;

expandDesign: (designId: string, additionalPrompt: string) => Promise<void>;

resetSession: () => void;

selectModel: (phase: 'analysis' | 'generation', modelId: string) => void;

}

### **FavoritesContext**

Manages saved favorite designs.

interface FavoritesContextType {

favorites: Design[];

addFavorite: (design: Design) => void;

removeFavorite: (designId: string) => void;

isFavorite: (designId: string) => boolean;

}

## **Services**

### **AI Services**

#### **geminiService.ts**

Handles interactions with Google's Gemini AI.

interface GeminiService {

analyzeDesign: (imageData: string, apiKey?: string) => Promise<AnalysisResult>;

supportedModels: Model[];

}

#### **claudeService.ts**

Handles interactions with Anthropic's Claude AI.

interface ClaudeService {

generateDesigns: (

analysisResult: AnalysisResult,

userPrompt: string,

count: number,

apiKey?: string

) => Promise<Design[]>;

expandDesign: (

design: Design,

additionalPrompt: string,

apiKey?: string

) => Promise<Design>;

supportedModels: Model[];

}

### **Storage Services**

#### **favoriteService.ts**

Manages saving and retrieving favorite designs.

interface FavoriteService {

getFavorites: () => Design[];

addFavorite: (design: Design) => void;

removeFavorite: (designId: string) => void;

}

#### **settingsService.ts**

Manages user settings persistence.

interface SettingsService {

getSettings: () => UserSettings;

updateSettings: (settings: Partial<UserSettings>) => void;

clearSettings: () => void;

}

## **Data Models**

### **Design**

Represents a generated website design.

interface Design {

id: string;

htmlContent: string;

cssContent: string;

thumbnail?: string;

createdAt: number;

parentDesignId?: string;

prompt: string;

analysisModel: string;

generationModel: string;

}

### **AnalysisResult**

Represents the output from the design analysis phase.

interface AnalysisResult {

id: string;

originalImageUrl: string;

elements: {

layout: string;

colorPalette: string[];

typography: {

headings: string;

body: string;

other?: string;

};

components: string[];

style: string;

};

rawAnalysis: string;

model: string;

createdAt: number;

}

### **UserSettings**

Represents user configuration options.

interface UserSettings {

apiKeys: {

claude: string;

gemini: string;

openai?: string;

};

defaultModels: {

analysis: string;

generation: string;

};

theme?: 'light' | 'dark' | 'system';

}

## **Implementation Guides**

### **1. Initial Setup**

1. Create a new React project with TypeScript:

npx create-react-app webdesign-ai-generator --template typescript

1. Install dependencies:

npm install tailwindcss postcss autoprefixer react-router-dom axios uuid react-dropzone react-modal

1. Configure Tailwind CSS:

npx tailwindcss init -p

1. Set up folder structure as defined in the project structure section.

### **2. Core Functionality Implementation**

#### **Image Upload Flow**

1. Implement the ImageUpload component with react-dropzone.
2. Create the useImageUpload hook to handle file validation and processing.
3. Add the image preview functionality.
4. Implement the UploadContext to manage the current upload state.

// components/upload/ImageUpload.tsx

import React, { useCallback } from 'react';

import { useDropzone } from 'react-dropzone';

import { useDesignContext } from '../../contexts/DesignContext';

import { Button } from '../common/Button';

import { ImagePreview } from './ImagePreview';

export const ImageUpload: React.FC = () => {

const { setOriginalImage, originalImage, resetSession } = useDesignContext();

const onDrop = useCallback((acceptedFiles: File[]) => {

if (acceptedFiles && acceptedFiles.length > 0) {

setOriginalImage(acceptedFiles[0]);

}

}, [setOriginalImage]);

const { getRootProps, getInputProps, isDragActive } = useDropzone({

onDrop,

accept: {

'image/\*': ['.png', '.jpg', '.jpeg', '.gif', '.webp']

},

maxFiles: 1

});

return (

<div className="w-full max-w-md mx-auto">

{!originalImage ? (

<div

{...getRootProps()}

className={`border-2 border-dashed rounded-lg p-6 text-center cursor-pointer transition-colors ${

isDragActive ? 'border-blue-500 bg-blue-50' : 'border-gray-300 hover:border-blue-400'

}`}

>

<input {...getInputProps()} />

<p className="text-gray-700">Click to upload image or drag and drop</p>

<p className="text-sm text-gray-500 mt-1">Supported formats: PNG, JPG, JPEG, GIF, WEBP</p>

</div>

) : (

<div className="mt-4">

<ImagePreview image={originalImage} />

<div className="mt-2 flex justify-end">

<Button onClick={resetSession} variant="secondary" className="mr-2">

Clear

</Button>

</div>

</div>

)}

</div>

);

};

#### **Analysis and Generation Flow**

1. Implement the AI service interfaces for Gemini and Claude.
2. Create the PromptInput and AISelector components.
3. Implement the design generation workflow in the DesignContext.

// services/ai/geminiService.ts

import axios from 'axios';

import { AnalysisResult } from '../../types/design';

import { apiEndpoints } from '../../constants/apiEndpoints';

export const analyzeDesign = async (

imageData: string,

model: string = 'gemini-pro-vision',

apiKey?: string

): Promise<AnalysisResult> => {

const headers: Record<string, string> = {

'Content-Type': 'application/json'

};

if (apiKey) {

headers['x-api-key'] = apiKey;

}

try {

const response = await axios.post(

apiEndpoints.gemini,

{

model,

contents: [

{

parts: [

{ text: "Analyze this website design. Identify layout structure, color palette, typography, and key design elements. Return the analysis in JSON format." },

{ inline\_data: { mime\_type: "image/jpeg", data: imageData } }

]

}

]

},

{ headers }

);

// Process response into AnalysisResult format

const rawAnalysis = response.data.candidates[0].content.parts[0].text;

// Try to parse JSON response

let elements;

try {

elements = JSON.parse(rawAnalysis);

} catch (e) {

// If JSON parsing fails, extract information using regex or other means

elements = extractElementsFromText(rawAnalysis);

}

return {

id: Date.now().toString(),

originalImageUrl: imageData,

elements,

rawAnalysis,

model,

createdAt: Date.now()

};

} catch (error) {

console.error('Error analyzing design:', error);

throw new Error('Failed to analyze design');

}

};

const extractElementsFromText = (text: string) => {

// Fallback extraction logic

return {

layout: extractLayoutInfo(text),

colorPalette: extractColors(text),

typography: {

headings: extractHeadingInfo(text),

body: extractBodyTextInfo(text)

},

components: extractComponentInfo(text),

style: extractStyleInfo(text)

};

};

// Helper extraction functions

const extractLayoutInfo = (text: string) => {

// Implementation details

return '';

};

const extractColors = (text: string) => {

// Implementation details

return [];

};

// Other extraction functions...

export const supportedModels = [

{ id: 'gemini-2.0-flash', name: 'Gemini 2.0 Flash', description: 'Fast analysis with high accuracy' },

{ id: 'gemini-2.0-pro', name: 'Gemini 2.0 Pro', description: 'Best for detailed analysis' }

];

// services/ai/claudeService.ts

import axios from 'axios';

import { v4 as uuidv4 } from 'uuid';

import { Design, AnalysisResult } from '../../types/design';

import { apiEndpoints } from '../../constants/apiEndpoints';

export const generateDesigns = async (

analysisResult: AnalysisResult,

userPrompt: string,

count: number = 6,

model: string = 'claude-3-7-sonnet-20250219',

apiKey?: string

): Promise<Design[]> => {

const headers: Record<string, string> = {

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

'anthropic-version': '2023-06-01'

};

if (apiKey) {

headers['x-api-key'] = apiKey;

}

try {

const response = await axios.post(

apiEndpoints.claude,

{

model,

max\_tokens: 4000,

messages: [

{

role: "user",

content: `Based on this website design analysis: ${JSON.stringify(analysisResult.elements)}

${userPrompt ? `And considering this additional request: ${userPrompt}` : ''}

Generate ${count} variations of HTML/CSS code for new website designs that maintain the core aesthetic but offer creative alternatives.

For each design, return a JSON object with the following structure:

{

"htmlContent": "Complete HTML code",

"cssContent": "Complete CSS code",

"description": "Brief description of the design"

}

Return all designs in a JSON array.`

}

]

},

{ headers }

);

const responseText = response.data.content[0].text;

// Extract JSON from response

const jsonMatch = responseText.match(/```json([\s\S]\*?)```/) ||

responseText.match(/\[([\s\S]\*?)\]/) ||

responseText.match(/\{([\s\S]\*?)\}/);

let designsData = [];

if (jsonMatch) {

try {

const jsonString = jsonMatch[0].replace(/```json|```/g, '');

designsData = JSON.parse(jsonString);

} catch (e) {

console.error('Error parsing JSON from Claude response:', e);

// Fallback extraction

designsData = extractDesignsFromText(responseText);

}

} else {

designsData = extractDesignsFromText(responseText);

}

// Map to Design objects

return designsData.map((data: any) => ({

id: uuidv4(),

htmlContent: data.htmlContent,

cssContent: data.cssContent,

createdAt: Date.now(),

prompt: userPrompt,

analysisModel: analysisResult.model,

generationModel: model,

description: data.description || ''

}));

} catch (error) {

console.error('Error generating designs:', error);

throw new Error('Failed to generate designs');

}

};

export const expandDesign = async (

design: Design,

additionalPrompt: string,

model: string = 'claude-3-7-sonnet-20250219',

apiKey?: string

): Promise<Design> => {

// Similar implementation to generateDesigns but focused on expanding a single design

// ...

return {

...design,

id: uuidv4(),

parentDesignId: design.id,

// Updated properties based on expansion

prompt: `${design.prompt} ${additionalPrompt}`,

createdAt: Date.now()

};

};

const extractDesignsFromText = (text: string) => {

// Fallback extraction logic for when JSON parsing fails

// Implementation details

return [];

};

export const supportedModels = [

{ id: 'claude-3-7-sonnet-20250219', name: 'Claude 3.7 Sonnet', description: 'Best for creative design generation' },

{ id: 'claude-3-5-sonnet-20240620', name: 'Claude 3.5 Sonnet', description: 'Good balance of creativity and speed' }

];

#### **Design Gallery and Interaction**

1. Implement the DesignGallery and DesignCard components.
2. Create the design detail view.
3. Implement the expand functionality.

// components/designs/DesignGallery.tsx

import React from 'react';

import { useDesignContext } from '../../contexts/DesignContext';

import { DesignCard } from './DesignCard';

import { LoadingSpinner } from '../common/LoadingSpinner';

import { EmptyState } from './EmptyState';

export const DesignGallery: React.FC = () => {

const { generatedDesigns, isGenerating } = useDesignContext();

if (isGenerating) {

return (

<div className="flex justify-center items-center h-64">

<LoadingSpinner />

<p className="ml-2 text-gray-600">Generating designs...</p>

</div>

);

}

if (generatedDesigns.length === 0) {

return <EmptyState message="No designs generated yet" />;

}

return (

<div className="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-3 gap-4">

{generatedDesigns.map(design => (

<DesignCard key={design.id} design={design} />

))}

</div>

);

};

// components/designs/DesignCard.tsx

import React from 'react';

import { useNavigate } from 'react-router-dom';

import { useFavoritesContext } from '../../contexts/FavoritesContext';

import { Design } from '../../types/design';

import { Button } from '../common/Button';

interface DesignCardProps {

design: Design;

}

export const DesignCard: React.FC<DesignCardProps> = ({ design }) => {

const navigate = useNavigate();

const { addFavorite, removeFavorite, isFavorite } = useFavoritesContext();

const favorite = isFavorite(design.id);

const handleViewDetail = () => {

navigate(`/design/${design.id}`);

};

const handleExpand = () => {

navigate(`/expand/${design.id}`);

};

const handleToggleFavorite = () => {

if (favorite) {

removeFavorite(design.id);

} else {

addFavorite(design);

}

};

return (

<div className="border rounded-lg overflow-hidden shadow-sm hover:shadow-md transition-shadow">

<div

className="h-48 bg-gray-100 cursor-pointer"

onClick={handleViewDetail}

>

{/\* Render the design preview \*/}

<iframe

title={`Design ${design.id}`}

srcDoc={`

<html>

<head>

<style>${design.cssContent}</style>

</head>

<body>${design.htmlContent}</body>

</html>

`}

className="w-full h-full border-0"

/>

</div>

<div className="p-3 flex justify-between">

<Button onClick={handleExpand} variant="primary" size="sm">

Expand on Design

</Button>

<Button

onClick={handleToggleFavorite}

variant="secondary"

size="sm"

>

{favorite ? 'Saved' : 'Save'}

</Button>

</div>

</div>

);

};

### **3. Settings and API Key Management**

1. Implement the settings page with API key inputs.
2. Create the settings context and service.

// contexts/SettingsContext.tsx

import React, { createContext, useContext, useState, useEffect } from 'react';

import { settingsService } from '../services/storage/settingsService';

import { UserSettings } from '../types/settings';

interface SettingsContextType {

settings: UserSettings;

updateApiKey: (service: string, key: string) => void;

updateDefaultModel: (phase: string, model: string) => void;

clearSettings: () => void;

}

const defaultSettings: UserSettings = {

apiKeys: {

claude: '',

gemini: '',

openai: ''

},

defaultModels: {

analysis: 'gemini-2.0-flash',

generation: 'claude-3-7-sonnet-20250219'

}

};

const SettingsContext = createContext<SettingsContextType>({

settings: defaultSettings,

updateApiKey: () => {},

updateDefaultModel: () => {},

clearSettings: () => {}

});

export const SettingsProvider: React.FC<{ children: React.ReactNode }> = ({ children }) => {

const [settings, setSettings] = useState<UserSettings>(defaultSettings);

useEffect(() => {

const loadedSettings = settingsService.getSettings();

setSettings(loadedSettings);

}, []);

const updateApiKey = (service: string, key: string) => {

const updatedSettings = {

...settings,

apiKeys: {

...settings.apiKeys,

[service]: key

}

};

setSettings(updatedSettings);

settingsService.updateSettings(updatedSettings);

};

const updateDefaultModel = (phase: string, model: string) => {

const updatedSettings = {

...settings,

defaultModels: {

...settings.defaultModels,

[phase]: model

}

};

setSettings(updatedSettings);

settingsService.updateSettings(updatedSettings);

};

const clearSettings = () => {

setSettings(defaultSettings);

settingsService.clearSettings();

};

return (

<SettingsContext.Provider value={{ settings, updateApiKey, updateDefaultModel, clearSettings }}>

{children}

</SettingsContext.Provider>

);

};

export const useSettings = () => useContext(SettingsContext);

// services/storage/settingsService.ts

import { UserSettings } from '../../types/settings';

import { LOCAL\_STORAGE\_KEYS } from '../../constants/localStorage';

const defaultSettings: UserSettings = {

apiKeys: {

claude: '',

gemini: '',

openai: ''

},

defaultModels: {

analysis: 'gemini-pro-vision',

generation: 'claude-3-7-sonnet-20250219'

}

};

export const settingsService = {

getSettings: (): UserSettings => {

try {

const storedSettings = localStorage.getItem(LOCAL\_STORAGE\_KEYS.SETTINGS);

if (!storedSettings) return defaultSettings;

return JSON.parse(storedSettings);

} catch (error) {

console.error('Error retrieving settings:', error);

return defaultSettings;

}

},

updateSettings: (settings: Partial<UserSettings>): void => {

try {

const currentSettings = settingsService.getSettings();

const updatedSettings = {

...currentSettings,

...settings,

apiKeys: {

...currentSettings.apiKeys,

...(settings.apiKeys || {})

},

defaultModels: {

...currentSettings.defaultModels,

...(settings.defaultModels || {})

}

};

localStorage.setItem(

LOCAL\_STORAGE\_KEYS.SETTINGS,

JSON.stringify(updatedSettings)

);

} catch (error) {

console.error('Error updating settings:', error);

}

},

clearSettings: (): void => {

localStorage.removeItem(LOCAL\_STORAGE\_KEYS.SETTINGS);

}

};

### **4. Favorites Management**

1. Implement the favorites context and service.
2. Create the favorites page.

// contexts/FavoritesContext.tsx

import React, { createContext, useContext, useState, useEffect } from 'react';

import { favoriteService } from '../services/storage/favoriteService';

import { Design } from '../types/design';

interface FavoritesContextType {

favorites: Design[];

addFavorite: (design: Design) => void;

removeFavorite: (designId: string) => void;

isFavorite: (designId: string) => boolean;

}

const FavoritesContext = createContext<FavoritesContextType>({

favorites: [],

addFavorite: () => {},

removeFavorite: () => {},

isFavorite: () => false

});

export const FavoritesProvider: React.FC<{ children: React.ReactNode }> = ({ children }) => {

const [favorites, setFavorites] = useState<Design[]>([]);

useEffect(() => {

const loadedFavorites = favoriteService.getFavorites();

setFavorites(loadedFavorites);

}, []);

const addFavorite = (design: Design) => {

const updatedFavorites = [...favorites, design];

setFavorites(updatedFavorites);

favoriteService.addFavorite(design);

};

const removeFavorite = (designId: string) => {

const updatedFavorites = favorites.filter(design => design.id !== designId);

setFavorites(updatedFavorites);

favoriteService.removeFavorite(designId);

};

const isFavorite = (designId: string) => {

return favorites.some(design => design.id === designId);

};

return (

<FavoritesContext.Provider value={{ favorites, addFavorite, removeFavorite, isFavorite }}>

{children}

</FavoritesContext.Provider>

);

};

export const useFavorites = () => useContext(FavoritesContext);

// services/storage/favoriteService.ts

import { Design } from '../../types/design';

import { LOCAL\_STORAGE\_KEYS } from '../../constants/localStorage';

export const favoriteService = {

getFavorites: (): Design[] => {

try {

const storedFavorites = localStorage.getItem(LOCAL\_STORAGE\_KEYS.FAVORITES);

if (!storedFavorites) return [];

return JSON.parse(storedFavorites);

} catch (error) {

console.error('Error retrieving favorites:', error);

return [];

}

},

addFavorite: (design: Design): void => {

try {

const favorites = favoriteService.getFavorites();

const updatedFavorites = [...favorites, design];

localStorage.setItem(

LOCAL\_STORAGE\_KEYS.FAVORITES,

JSON.stringify(updatedFavorites)

);

} catch (error) {

console.error('Error removing favorite:', error);

}

}

};

### **5. Expand on Design Functionality**

1. Implement the design expansion workflow.
2. Create the expand design page.

// pages/ExpandDesign.tsx

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

import { useParams, useNavigate } from 'react-router-dom';

import { useDesignContext } from '../contexts/DesignContext';

import { Button } from '../components/common/Button';

import { PromptInput } from '../components/prompt/PromptInput';

import { LoadingSpinner } from '../components/common/LoadingSpinner';

export const ExpandDesign: React.FC = () => {

const { designId } = useParams<{ designId: string }>();

const navigate = useNavigate();

const { generatedDesigns, expandDesign, isGenerating } = useDesignContext();

const [design, setDesign] = useState(null);

const [additionalPrompt, setAdditionalPrompt] = useState('');

useEffect(() => {

if (designId && generatedDesigns.length > 0) {

const foundDesign = generatedDesigns.find(d => d.id === designId);

if (foundDesign) {

setDesign(foundDesign);

} else {

navigate('/');

}

}

}, [designId, generatedDesigns, navigate]);

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

e.preventDefault();

if (!design || !additionalPrompt.trim()) return;

try {

await expandDesign(design.id, additionalPrompt);

navigate('/');

} catch (error) {

console.error('Error expanding design:', error);

}

};

if (!design) {

return (

<div className="flex justify-center items-center h-64">

<LoadingSpinner />

<p className="ml-2 text-gray-600">Loading design...</p>

</div>

);

}

return (

<div className="max-w-4xl mx-auto p-4">

<h1 className="text-2xl font-semibold mb-4">Expand on Design</h1>

<div className="mb-6 border rounded-lg overflow-hidden">

<iframe

title={`Design ${design.id}`}

srcDoc={`

<html>

<head>

<style>${design.cssContent}</style>

</head>

<body>${design.htmlContent}</body>

</html>

`}

className="w-full h-64 border-0"

/>

</div>

<div className="mb-4">

<h2 className="text-lg font-medium mb-2">Original Prompt</h2>

<p className="bg-gray-100 p-3 rounded">{design.prompt || 'No prompt provided'}</p>

</div>

<form onSubmit={handleSubmit}>

<h2 className="text-lg font-medium mb-2">Additional Directions</h2>

<PromptInput

value={additionalPrompt}

onChange={setAdditionalPrompt}

placeholder="Add more specific directions for how to expand this design..."

/>

<div className="mt-4 flex justify-end">

<Button

type="button"

variant="secondary"

onClick={() => navigate('/')}

className="mr-2"

>

Cancel

</Button>

<Button

type="submit"

variant="primary"

disabled={isGenerating || !additionalPrompt.trim()}

>

{isGenerating ? 'Generating...' : 'Expand Design'}

</Button>

</div>

</form>

</div>

);

};

### **6. Design Detail Page**

Create a detailed view for inspecting designs.

// pages/DesignDetail.tsx

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

import { useParams, useNavigate } from 'react-router-dom';

import { useDesignContext } from '../contexts/DesignContext';

import { useFavoritesContext } from '../contexts/FavoritesContext';

import { Button } from '../components/common/Button';

import { Tabs, Tab } from '../components/common/Tabs';

import { LoadingSpinner } from '../components/common/LoadingSpinner';

export const DesignDetail: React.FC = () => {

const { designId } = useParams<{ designId: string }>();

const navigate = useNavigate();

const { generatedDesigns } = useDesignContext();

const { addFavorite, removeFavorite, isFavorite } = useFavoritesContext();

const [design, setDesign] = useState(null);

const [activeTab, setActiveTab] = useState('preview');

useEffect(() => {

if (designId) {

// Check in generated designs

let foundDesign = generatedDesigns.find(d => d.id === designId);

// If not found, check in favorites

if (!foundDesign) {

const favorites = useFavoritesContext().favorites;

foundDesign = favorites.find(d => d.id === designId);

}

if (foundDesign) {

setDesign(foundDesign);

} else {

navigate('/');

}

}

}, [designId, generatedDesigns, navigate]);

const handleToggleFavorite = () => {

if (!design) return;

if (isFavorite(design.id)) {

removeFavorite(design.id);

} else {

addFavorite(design);

}

};

const handleDownload = () => {

if (!design) return;

// Create HTML file content

const htmlContent = `

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Design ${design.id}</title>

<style>

${design.cssContent}

</style>

</head>

<body>

${design.htmlContent}

</body>

</html>

`;

// Create download link

const blob = new Blob([htmlContent], { type: 'text/html' });

const url = URL.createObjectURL(blob);

const a = document.createElement('a');

a.href = url;

a.download = `design-${design.id}.html`;

document.body.appendChild(a);

a.click();

document.body.removeChild(a);

URL.revokeObjectURL(url);

};

if (!design) {

return (

<div className="flex justify-center items-center h-64">

<LoadingSpinner />

<p className="ml-2 text-gray-600">Loading design...</p>

</div>

);

}

return (

<div className="max-w-4xl mx-auto p-4">

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

<h1 className="text-2xl font-semibold">Design Detail</h1>

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

<Button onClick={() => navigate(`/expand/${design.id}`)} variant="primary">

Expand on Design

</Button>

<Button

onClick={handleToggleFavorite}

variant={isFavorite(design.id) ? "secondary" : "outline"}

>

{isFavorite(design.id) ? 'Saved' : 'Save'}

</Button>

<Button onClick={handleDownload} variant="outline">

Download

</Button>

</div>

</div>

<Tabs activeTab={activeTab} onChange={setActiveTab}>

<Tab id="preview" label="Preview">

<div className="border rounded-lg overflow-hidden h-96">

<iframe

title={`Design ${design.id}`}

srcDoc={`

<html>

<head>

<style>${design.cssContent}</style>

</head>

<body>${design.htmlContent}</body>

</html>

`}

className="w-full h-full border-0"

/>

</div>

</Tab>

<Tab id="html" label="HTML">

<pre className="bg-gray-100 p-4 rounded-lg overflow-auto h-96">

<code>{design.htmlContent}</code>

</pre>

</Tab>

<Tab id="css" label="CSS">

<pre className="bg-gray-100 p-4 rounded-lg overflow-auto h-96">

<code>{design.cssContent}</code>

</pre>

</Tab>

</Tabs>

<div className="mt-4">

<h2 className="text-lg font-medium mb-2">Design Details</h2>

<div className="bg-gray-100 p-4 rounded-lg">

<p><strong>Prompt:</strong> {design.prompt || 'No prompt provided'}</p>

<p><strong>Analysis Model:</strong> {design.analysisModel}</p>

<p><strong>Generation Model:</strong> {design.generationModel}</p>

<p><strong>Created:</strong> {new Date(design.createdAt).toLocaleString()}</p>

</div>

</div>

</div>

);

};

## **Performance Optimization Strategies**

### **1. Code Splitting**

Implement React.lazy and Suspense to load components only when needed:

// App.tsx

import React, { Suspense, lazy } from 'react';

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

import { Header } from './components/layout/Header';

import { LoadingSpinner } from './components/common/LoadingSpinner';

import { SettingsProvider } from './contexts/SettingsContext';

import { DesignProvider } from './contexts/DesignContext';

import { FavoritesProvider } from './contexts/FavoritesContext';

// Lazy-loaded pages

const Home = lazy(() => import('./pages/Home'));

const Settings = lazy(() => import('./pages/Settings'));

const Favorites = lazy(() => import('./pages/Favorites'));

const DesignDetail = lazy(() => import('./pages/DesignDetail'));

const ExpandDesign = lazy(() => import('./pages/ExpandDesign'));

const App: React.FC = () => {

return (

<BrowserRouter>

<SettingsProvider>

<DesignProvider>

<FavoritesProvider>

<div className="min-h-screen bg-gray-50">

<Header />

<main className="container mx-auto py-6 px-4">

<Suspense fallback={

<div className="flex justify-center items-center h-64">

<LoadingSpinner />

</div>

}>

<Routes>

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

<Route path="/settings" element={<Settings />} />

<Route path="/favorites" element={<Favorites />} />

<Route path="/design/:designId" element={<DesignDetail />} />

<Route path="/expand/:designId" element={<ExpandDesign />} />

</Routes>

</Suspense>

</main>

</div>

</FavoritesProvider>

</DesignProvider>

</SettingsProvider>

</BrowserRouter>

);

};

export default App;

### **2. Memoization**

Use React.memo for expensive components:

// components/designs/DesignCard.tsx

import React from 'react';

// ... imports

const DesignCardComponent: React.FC<DesignCardProps> = ({ design }) => {

// ... implementation

};

export const DesignCard = React.memo(DesignCardComponent);

### **3. Virtualization**

For long lists, implement virtualization to render only visible items:

// components/designs/DesignGallery.tsx

import React from 'react';

import { FixedSizeGrid } from 'react-window';

import { useDesignContext } from '../../contexts/DesignContext';

import { DesignCard } from './DesignCard';

import { useMediaQuery } from '../../hooks/useMediaQuery';

export const DesignGallery: React.FC = () => {

const { generatedDesigns } = useDesignContext();

const isMobile = useMediaQuery('(max-width: 640px)');

const isTablet = useMediaQuery('(max-width: 1024px) and (min-width: 641px)');

// Determine number of columns based on screen size

const columnCount = isMobile ? 1 : isTablet ? 2 : 3;

const rowCount = Math.ceil(generatedDesigns.length / columnCount);

// Calculate optimal cell size based on available width

const cellWidth = Math.floor((window.innerWidth - 32) / columnCount);

const cellHeight = 280; // Fixed height for each card

const Cell = ({ columnIndex, rowIndex, style }) => {

const index = rowIndex \* columnCount + columnIndex;

if (index >= generatedDesigns.length) return null;

return (

<div style={style}>

<DesignCard design={generatedDesigns[index]} />

</div>

);

};

return (

<FixedSizeGrid

columnCount={columnCount}

columnWidth={cellWidth}

height={Math.min(window.innerHeight \* 0.8, rowCount \* cellHeight)}

rowCount={rowCount}

rowHeight={cellHeight}

width={window.innerWidth - 32}

>

{Cell}

</FixedSizeGrid>

);

};

## **Error Handling Strategy**

### **1. Error Boundaries**

Create an error boundary component to catch and handle errors:

// components/common/ErrorBoundary.tsx

import React, { Component, ErrorInfo, ReactNode } from 'react';

interface Props {

children: ReactNode;

fallback?: ReactNode;

}

interface State {

hasError: boolean;

error: Error | null;

}

export class ErrorBoundary extends Component<Props, State> {

public state: State = {

hasError: false,

error: null

};

public static getDerivedStateFromError(error: Error): State {

return { hasError: true, error };

}

public componentDidCatch(error: Error, errorInfo: ErrorInfo) {

console.error("Uncaught error:", error, errorInfo);

}

public render() {

if (this.state.hasError) {

return this.props.fallback || (

<div className="text-center p-6 bg-red-50 rounded-lg">

<h2 className="text-xl font-semibold text-red-700 mb-2">Something went wrong</h2>

<p className="text-red-600 mb-4">

{this.state.error?.message || "An unexpected error occurred"}

</p>

<button

className="px-4 py-2 bg-red-600 text-white rounded hover:bg-red-700 transition-colors"

onClick={() => this.setState({ hasError: false, error: null })}

>

Try again

</button>

</div>

);

}

return this.props.children;

}

}

### **2. API Error Handling**

Create a reusable utility for handling API errors:

// utils/errorHandling.ts

export interface ApiError {

status?: number;

message: string;

code?: string;

}

export class ApiErrorHandler {

static handle(error: any): ApiError {

if (error.response) {

// The request was made and the server responded with an error status

return {

status: error.response.status,

message: error.response.data.message || 'Server error',

code: error.response.data.code

};

} else if (error.request) {

// The request was made but no response was received

return {

message: 'No response received from server. Please check your internet connection.'

};

} else {

// Something happened in setting up the request

return {

message: error.message || 'An unexpected error occurred'

};

}

}

static isRateLimitError(error: any): boolean {

return error.response && (error.response.status === 429 ||

(error.response.data && error.response.data.code === 'rate\_limit\_exceeded'));

}

static isAuthError(error: any): boolean {

return error.response && (error.response.status === 401 || error.response.status === 403);

}

}

## **Deployment Considerations**

### **1. Environment Configuration**

Create environment-specific configuration files:

// constants/config.ts

export const CONFIG = {

API\_BASE\_URL: process.env.REACT\_APP\_API\_BASE\_URL || 'http://localhost:3001',

GEMINI\_API\_ENDPOINT: process.env.REACT\_APP\_GEMINI\_API\_ENDPOINT || 'https://api.google.com/v1/models/gemini-pro-vision:generateContent',

CLAUDE\_API\_ENDPOINT: process.env.REACT\_APP\_CLAUDE\_API\_ENDPOINT || 'https://api.anthropic.com/v1/messages',

DEFAULT\_ANALYSIS\_MODEL: process.env.REACT\_APP\_DEFAULT\_ANALYSIS\_MODEL || 'gemini-2.0-flash',

DEFAULT\_GENERATION\_MODEL: process.env.REACT\_APP\_DEFAULT\_GENERATION\_MODEL || 'claude-3-7-sonnet-20250219',

MAX\_IMAGE\_SIZE\_MB: 5,

MAX\_DESIGNS\_PER\_SESSION: 20

};

### **2. Build Setup**

Configure the build process in package.json:

{

"scripts": {

"start": "react-scripts start",

"build": "react-scripts build",

"build:staging": "env-cmd -f .env.staging react-scripts build",

"build:production": "env-cmd -f .env.production react-scripts build",

"test": "react-scripts test",

"eject": "react-scripts eject",

"lint": "eslint --ext .ts,.tsx src",

"format": "prettier --write \"src/\*\*/\*.{ts,tsx,js,jsx,json,css}\""

}

}

## **Testing Strategy**

### **1. Component Testing**

Use React Testing Library for component tests:

// components/upload/ImageUpload.test.tsx

import React from 'react';

import { render, screen, fireEvent } from '@testing-library/react';

import { ImageUpload } from './ImageUpload';

import { DesignContext } from '../../contexts/DesignContext';

// Mock the design context

const mockSetOriginalImage = jest.fn();

const mockResetSession = jest.fn();

const mockContextValue = {

originalImage: null,

setOriginalImage: mockSetOriginalImage,

resetSession: mockResetSession,

// Add other required context properties

analysisResult: null,

generatedDesigns: [],

currentPrompt: '',

isAnalyzing: false,

isGenerating: false,

selectedModels: {

analysis: 'gemini-pro-vision',

generation: 'claude-3-7-sonnet-20250219'

},

setCurrentPrompt: jest.fn(),

analyzeImage: jest.fn(),

generateDesigns: jest.fn(),

expandDesign: jest.fn(),

selectModel: jest.fn()

};

describe('ImageUpload', () => {

beforeEach(() => {

jest.clearAllMocks();

});

test('renders upload area when no image is selected', () => {

render(

<DesignContext.Provider value={mockContextValue}>

<ImageUpload />

</DesignContext.Provider>

);

expect(screen.getByText(/Click to upload image or drag and drop/i)).toBeInTheDocument();

expect(screen.getByText(/Supported formats/i)).toBeInTheDocument();

});

test('renders image preview when image is selected', () => {

const file = new File(['dummy content'], 'example.png', { type: 'image/png' });

render(

<DesignContext.Provider value={{

...mockContextValue,

originalImage: file

}}>

<ImageUpload />

</DesignContext.Provider>

);

// Check for Clear button

expect(screen.getByText('Clear')).toBeInTheDocument();

// ImagePreview should be rendered

// Note: You might need to adjust this based on how your ImagePreview component is implemented

expect(screen.queryByText(/Click to upload image or drag and drop/i)).not.toBeInTheDocument();

});

test('calls resetSession when Clear button is clicked', () => {

const file = new File(['dummy content'], 'example.png', { type: 'image/png' });

render(

<DesignContext.Provider value={{

...mockContextValue,

originalImage: file

}}>

<ImageUpload />

</DesignContext.Provider>

);

fireEvent.click(screen.getByText('Clear'));

expect(mockResetSession).toHaveBeenCalledTimes(1);

});

});

### **2. Service Testing**

Test the AI service layer:

// services/ai/geminiService.test.ts

import { analyzeDesign } from './geminiService';

import axios from 'axios';

// Mock axios

jest.mock('axios');

const mockedAxios = axios as jest.Mocked<typeof axios>;

describe('geminiService', () => {

beforeEach(() => {

jest.clearAllMocks();

});

test('analyzeDesign calls Gemini API with correct parameters', async () => {

// Mock successful API response

mockedAxios.post.mockResolvedValueOnce({

data: {

candidates: [

{

content: {

parts: [

{

text: JSON.stringify({

layout: 'grid',

colorPalette: ['#fff', '#000'],

typography: {

headings: 'Sans-serif',

body: 'Sans-serif'

},

components: ['header', 'footer', 'content section'],

style: 'minimalist'

})

}

]

}

}

]

}

});

// Call the service

const imageData = 'base64EncodedImageData';

const result = await analyzeDesign(imageData, 'gemini-pro-vision', 'test-api-key');

// Assertions

expect(mockedAxios.post).toHaveBeenCalledTimes(1);

expect(mockedAxios.post).toHaveBeenCalledWith(

expect.any(String),

{

model: 'gemini-pro-vision',

contents: [

{

parts: [

{ text: expect.any(String) },

{ inline\_data: { mime\_type: 'image/jpeg', data: imageData } }

]

}

]

},

{ headers: expect.objectContaining({ 'x-api-key': 'test-api-key' }) }

);

// Check returned structure

expect(result).toHaveProperty('id');

expect(result).toHaveProperty('originalImageUrl', imageData);

expect(result).toHaveProperty('elements.layout', 'grid');

expect(result).toHaveProperty('elements.colorPalette');

expect(result).toHaveProperty('rawAnalysis');

expect(result).toHaveProperty('model', 'gemini-pro-vision');

expect(result).toHaveProperty('createdAt');

});

test('analyzeDesign handles error response', async () => {

// Mock failed API call

mockedAxios.post.mockRejectedValueOnce(new Error('API error'));

// Call the service and expect it to throw

await expect(analyzeDesign('imageData')).rejects.toThrow('Failed to analyze design');

});

});

## **Accessibility Considerations**

### **1. Form Controls**

Ensure form controls are properly labeled and accessible:

// components/prompt/PromptInput.tsx

import React from 'react';

interface PromptInputProps {

value: string;

onChange: (value: string) => void;

placeholder?: string;

maxLength?: number;

id?: string;

label?: string;

}

export const PromptInput: React.FC<PromptInputProps> = ({

value,

onChange,

placeholder = 'Enter additional instructions...',

maxLength = 500,

id = 'prompt-input',

label = 'Design Prompt'

}) => {

const handleChange = (e: React.ChangeEvent<HTMLTextAreaElement>) => {

onChange(e.target.value);

};

return (

<div className="w-full">

<label

htmlFor={id}

className="block text-sm font-medium text-gray-700 mb-1"

>

{label}

</label>

<textarea

id={id}

value={value}

onChange={handleChange}

placeholder={placeholder}

maxLength={maxLength}

rows={4}

className="w-full p-3 border border-gray-300 rounded-md focus:ring-blue-500 focus:border-blue-500"

aria-describedby={`${id}-hint`}

/>

<div

id={`${id}-hint`}

className="flex justify-end mt-1"

>

<span className="text-xs text-gray-500">

{value.length}/{maxLength} characters

</span>

</div>

</div>

);

};

### **2. Focus Management**

Implement proper focus management for modal dialogs:

// components/common/Modal.tsx

import React, { useEffect, useRef } from 'react';

import ReactDOM from 'react-dom';

interface ModalProps {

isOpen: boolean;

onClose: () => void;

title: string;

children: React.ReactNode;

}

export const Modal: React.FC<ModalProps> = ({ isOpen, onClose, title, children }) => {

const modalRef = useRef<HTMLDivElement>(null);

useEffect(() => {

const handleEscape = (e: KeyboardEvent) => {

if (e.key === 'Escape') {

onClose();

}

};

if (isOpen) {

document.addEventListener('keydown', handleEscape);

// Focus the modal when it opens

modalRef.current?.focus();

// Prevent scrolling of background content

document.body.style.overflow = 'hidden';

}

return () => {

document.removeEventListener('keydown', handleEscape);

document.body.style.overflow = 'unset';

};

}, [isOpen, onClose]);

if (!isOpen) return null;

return ReactDOM.createPortal(

<div className="fixed inset-0 z-50 overflow-y-auto">

<div className="flex items-center justify-center min-h-screen p-4">

{/\* Backdrop \*/}

<div

className="fixed inset-0 bg-black opacity-50 transition-opacity"

onClick={onClose}

aria-hidden="true"

/>

{/\* Modal \*/}

<div

ref={modalRef}

className="bg-white rounded-lg overflow-hidden shadow-xl transform transition-all w-full max-w-lg z-10"

role="dialog"

aria-modal="true"

aria-labelledby="modal-title"

tabIndex={-1}

>

<div className="px-4 py-3 border-b flex justify-between items-center">

<h2 id="modal-title" className="text-lg font-semibold">

{title}

</h2>

<button

onClick={onClose}

className="rounded-full p-1 hover:bg-gray-100"

aria-label="Close"

>

<svg className="w-5 h-5" fill="currentColor" viewBox="0 0 20 20">

<path

fillRule="evenodd"

d="M4.293 4.293a1 1 0 011.414 0L10 8.586l4.293-4.293a1 1 0 111.414 1.414L11.414 10l4.293 4.293a1 1 0 01-1.414 1.414L10 11.414l-4.293 4.293a1 1 0 01-1.414-1.414L8.586 10 4.293 5.707a1 1 0 010-1.414z"

clipRule="evenodd"

/>

</svg>

</button>

</div>

<div className="p-4">

{children}

</div>

</div>

</div>

</div>,

document.body

);

};

## **Additional Implementation Details**

### **User Interface Components**

#### **Header Component**

The Header component will provide navigation and access to settings:

// components/layout/Header.tsx

import React from 'react';

import { Link, useLocation } from 'react-router-dom';

export const Header: React.FC = () => {

const location = useLocation();

const isActive = (path: string) => {

return location.pathname === path ? 'text-blue-600 border-blue-600' : 'text-gray-500 border-transparent hover:text-gray-700 hover:border-gray-300';

};

return (

<header className="bg-white shadow-sm">

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

<div className="flex justify-between items-center h-16">

<div className="flex items-center">

<Link to="/" className="flex-shrink-0">

<h1 className="text-xl font-bold text-gray-800">WebDesign AI Generator</h1>

</Link>

</div>

<nav className="flex space-x-4">

<Link

to="/"

className={`inline-flex items-center px-1 pt-1 border-b-2 text-sm font-medium ${isActive('/')}`}

>

Home

</Link>

<Link

to="/favorites"

className={`inline-flex items-center px-1 pt-1 border-b-2 text-sm font-medium ${isActive('/favorites')}`}

>

Favorites

</Link>

<Link

to="/settings"

className={`inline-flex items-center px-1 pt-1 border-b-2 text-sm font-medium ${isActive('/settings')}`}

>

Settings

</Link>

</nav>

</div>

</div>

</header>

);

};

#### **Home Page**

The main page where users upload designs and generate variations:

// pages/Home.tsx

import React, { useState } from 'react';

import { ImageUpload } from '../components/upload/ImageUpload';

import { PromptInput } from '../components/prompt/PromptInput';

import { AISelector } from '../components/prompt/AISelector';

import { DesignGallery } from '../components/designs/DesignGallery';

import { Button } from '../components/common/Button';

import { useDesignContext } from '../contexts/DesignContext';

import { ErrorBoundary } from '../components/common/ErrorBoundary';

const Home: React.FC = () => {

const {

originalImage,

analysisResult,

currentPrompt,

setCurrentPrompt,

isAnalyzing,

isGenerating,

analyzeImage,

generateDesigns,

selectModel,

selectedModels

} = useDesignContext();

const handleAnalyze = async () => {

if (!originalImage) return;

await analyzeImage();

};

const handleGenerate = async () => {

if (!analysisResult) return;

await generateDesigns();

};

return (

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

<section className="mb-8">

<h2 className="text-xl font-semibold mb-4">Upload Website Design</h2>

<ErrorBoundary>

<ImageUpload />

</ErrorBoundary>

</section>

{originalImage && (

<section className="mb-8">

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

<h2 className="text-xl font-semibold">Analyze Design</h2>

<AISelector

phase="analysis"

selectedModel={selectedModels.analysis}

onSelect={(model) => selectModel('analysis', model)}

/>

</div>

<Button

onClick={handleAnalyze}

disabled={isAnalyzing || !originalImage}

className="w-full"

>

{isAnalyzing ? 'Analyzing...' : 'Analyze Design'}

</Button>

</section>

)}

{analysisResult && (

<section className="mb-8">

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

<h2 className="text-xl font-semibold">Generate Variations</h2>

<AISelector

phase="generation"

selectedModel={selectedModels.generation}

onSelect={(model) => selectModel('generation', model)}

/>

</div>

<div className="mb-4">

<PromptInput

value={currentPrompt}

onChange={setCurrentPrompt}

placeholder="Describe how you want to modify the design (e.g., 'Make it more minimal' or 'Use warmer colors')..."

label="Design Instructions (Optional)"

/>

</div>

<Button

onClick={handleGenerate}

disabled={isGenerating}

className="w-full"

>

{isGenerating ? 'Generating...' : 'Generate Design Variations'}

</Button>

</section>

)}

<section>

<h2 className="text-xl font-semibold mb-4">Generated Designs</h2>

<ErrorBoundary>

<DesignGallery />

</ErrorBoundary>

</section>

</div>

);

};

export default Home;

### **Handling API Responses**

When working with AI model responses, you'll need robust parsing logic:

// utils/designParser.ts

import { Design, AnalysisResult } from '../types/design';

import { v4 as uuidv4 } from 'uuid';

export const extractJsonFromResponse = (text: string): any => {

// Try to find JSON in markdown code blocks

const jsonMarkdownRegex = /```(?:json)?\s\*([\s\S]\*?)\s\*```/;

const markdownMatch = text.match(jsonMarkdownRegex);

if (markdownMatch && markdownMatch[1]) {

try {

return JSON.parse(markdownMatch[1]);

} catch (e) {

console.error('Failed to parse JSON from markdown block', e);

}

}

// Try to find JSON array pattern

const jsonArrayRegex = /\[\s\*\{[\s\S]\*\}\s\*\]/;

const arrayMatch = text.match(jsonArrayRegex);

if (arrayMatch) {

try {

return JSON.parse(arrayMatch[0]);

} catch (e) {

console.error('Failed to parse JSON array', e);

}

}

// Try to find a single JSON object

const jsonObjectRegex = /\{\s\*"[\w]+"\s\*:[\s\S]\*\}/;

const objectMatch = text.match(jsonObjectRegex);

if (objectMatch) {

try {

return JSON.parse(objectMatch[0]);

} catch (e) {

console.error('Failed to parse JSON object', e);

}

}

// If all parsing attempts fail, return null

return null;

};

export const parseDesignsFromResponse = (text: string, analysisResult: AnalysisResult, prompt: string, model: string): Design[] => {

const jsonData = extractJsonFromResponse(text);

if (Array.isArray(jsonData)) {

return jsonData.map((item) => mapToDesign(item, analysisResult, prompt, model));

}

// If the response is a single object with designs array

if (jsonData && jsonData.designs && Array.isArray(jsonData.designs)) {

return jsonData.designs.map((item) => mapToDesign(item, analysisResult, prompt, model));

}

// Fallback: Try to extract design information using regex patterns

return extractDesignsUsingRegex(text, analysisResult, prompt, model);

};

const mapToDesign = (data: any, analysisResult: AnalysisResult, prompt: string, model: string): Design => {

return {

id: uuidv4(),

htmlContent: data.htmlContent || data.html || '',

cssContent: data.cssContent || data.css || '',

thumbnail: data.thumbnail || '',

createdAt: Date.now(),

prompt,

analysisModel: analysisResult.model,

generationModel: model,

description: data.description || ''

};

};

const extractDesignsUsingRegex = (text: string, analysisResult: AnalysisResult, prompt: string, model: string): Design[] => {

const designs: Design[] = [];

// Extract HTML blocks

const htmlBlocks = text.match(/<html[\s\S]\*?<\/html>/g) || [];

// For each HTML block, try to extract CSS

htmlBlocks.forEach((htmlBlock) => {

const cssMatch = htmlBlock.match(/<style>([\s\S]\*?)<\/style>/);

const cssContent = cssMatch ? cssMatch[1] : '';

// Extract body content

const bodyMatch = htmlBlock.match(/<body>([\s\S]\*?)<\/body>/);

const htmlContent = bodyMatch ? bodyMatch[1] : htmlBlock;

designs.push({

id: uuidv4(),

htmlContent,

cssContent,

createdAt: Date.now(),

prompt,

analysisModel: analysisResult.model,

generationModel: model

});

});

// If no HTML blocks found, look for separate HTML and CSS sections

if (designs.length === 0) {

// Extract code blocks

const codeBlocks = text.match(/```(?:html|css)?\s\*([\s\S]\*?)\s\*```/g) || [];

let htmlContent = '';

let cssContent = '';

codeBlocks.forEach((block) => {

if (block.includes('```html') || (!block.includes('```css') && block.includes('<div'))) {

htmlContent = block.replace(/```(?:html)?\s\*|\s\*```/g, '');

} else if (block.includes('```css') || (!block.includes('```html') && block.includes('{'))) {

cssContent = block.replace(/```(?:css)?\s\*|\s\*```/g, '');

}

});

if (htmlContent) {

designs.push({

id: uuidv4(),

htmlContent,

cssContent,

createdAt: Date.now(),

prompt,

analysisModel: analysisResult.model,

generationModel: model

});

}

}

return designs;

};

### **Local Storage Management**

Create a hook for easier localStorage management:

// hooks/useLocalStorage.ts

import { useState, useEffect } from 'react';

export function useLocalStorage<T>(key: string, initialValue: T): [T, (value: T | ((val: T) => T)) => void] {

// Get stored value from localStorage or use initialValue

const readValue = (): T => {

if (typeof window === 'undefined') {

return initialValue;

}

try {

const item = window.localStorage.getItem(key);

return item ? JSON.parse(item) : initialValue;

} catch (error) {

console.warn(`Error reading localStorage key "${key}":`, error);

return initialValue;

}

};

// State to store our value

const [storedValue, setStoredValue] = useState<T>(readValue);

// Return a wrapped version of useState's setter function that persists the new value to localStorage

const setValue = (value: T | ((val: T) => T)) => {

try {

// Allow value to be a function so we have same API as useState

const valueToStore = value instanceof Function ? value(storedValue) : value;

// Save state

setStoredValue(valueToStore);

// Save to localStorage

if (typeof window !== 'undefined') {

window.localStorage.setItem(key, JSON.stringify(valueToStore));

}

} catch (error) {

console.warn(`Error setting localStorage key "${key}":`, error);

}

};

// Listen for changes to this localStorage key in other tabs/windows

useEffect(() => {

const handleStorageChange = (e: StorageEvent) => {

if (e.key === key && e.newValue !== null) {

setStoredValue(JSON.parse(e.newValue));

}

};

window.addEventListener('storage', handleStorageChange);

return () => {

window.removeEventListener('storage', handleStorageChange);

};

}, [key]);

return [storedValue, setValue];

}

### **API Integration Utils**

Create utility functions for API access:

// utils/api.ts

import axios, { AxiosRequestConfig, AxiosResponse } from 'axios';

import { ApiErrorHandler } from './errorHandling';

import { settingsService } from '../services/storage/settingsService';

const API\_TIMEOUT = 60000; // 60 seconds

export const createApiClient = (baseURL: string) => {

const client = axios.create({

baseURL,

timeout: API\_TIMEOUT,

headers: {

'Content-Type': 'application/json'

}

});

client.interceptors.request.use((config) => {

// Check if this is a request to an AI service that needs an API key

if (config.url?.includes('anthropic')) {

const settings = settingsService.getSettings();

if (settings.apiKeys.claude) {

config.headers['x-api-key'] = settings.apiKeys.claude;

config.headers['anthropic-version'] = '2023-06-01';

}

} else if (config.url?.includes('google')) {

const settings = settingsService.getSettings();

if (settings.apiKeys.gemini) {

config.headers['x-api-key'] = settings.apiKeys.gemini;

}

}

return config;

});

client.interceptors.response.use(

(response) => response,

(error) => {

const apiError = ApiErrorHandler.handle(error);

// Handle rate limiting

if (ApiErrorHandler.isRateLimitError(error)) {

// Could implement retry logic here

console.warn('Rate limit exceeded', apiError);

}

// Handle auth errors

if (ApiErrorHandler.isAuthError(error)) {

console.error('Authentication error', apiError);

// Could trigger a notification to update API key

}

return Promise.reject(apiError);

}

);

return client;

};

// Helper function for making API requests with proper error handling

export const apiRequest = async <T>(

config: AxiosRequestConfig

): Promise<T> => {

try {

const client = createApiClient('');

const response: AxiosResponse<T> = await client(config);

return response.data;

} catch (error) {

console.error('API request failed:', error);

throw error;

}

};

### **Media Queries Hook**

Create a hook for responsive design:

// hooks/useMediaQuery.ts

import { useState, useEffect } from 'react';

export const useMediaQuery = (query: string): boolean => {

const [matches, setMatches] = useState(false);

useEffect(() => {

const media = window.matchMedia(query);

// Initial check

if (media.matches !== matches) {

setMatches(media.matches);

}

// Listen for changes

const listener = () => setMatches(media.matches);

media.addEventListener('change', listener);

return () => media.removeEventListener('change', listener);

}, [query, matches]);

return matches;

};

## **Production Optimization**

### **Code Splitting**

Implement code splitting to optimize initial load time:

// App.tsx with code splitting

import React, { Suspense, lazy } from 'react';

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

import { Header } from './components/layout/Header';

import { LoadingSpinner } from './components/common/LoadingSpinner';

import { ErrorBoundary } from './components/common/ErrorBoundary';

// Context providers

import { SettingsProvider } from './contexts/SettingsContext';

import { DesignProvider } from './contexts/DesignContext';

import { FavoritesProvider } from './contexts/FavoritesContext';

// Lazy-loaded pages

const Home = lazy(() => import('./pages/Home'));

const Settings = lazy(() => import('./pages/Settings'));

const Favorites = lazy(() => import('./pages/Favorites'));

const DesignDetail = lazy(() => import('./pages/DesignDetail'));

const ExpandDesign = lazy(() => import('./pages/ExpandDesign'));

const App: React.FC = () => {

return (

<BrowserRouter>

<ErrorBoundary>

<SettingsProvider>

<DesignProvider>

<FavoritesProvider>

<div className="min-h-screen bg-gray-50">

<Header />

<main className="container mx-auto py-6 px-4">

<Suspense fallback={

<div className="flex justify-center items-center h-64">

<LoadingSpinner />

<p className="ml-2">Loading...</p>

</div>

}>

<Routes>

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

<Route path="/settings" element={<Settings />} />

<Route path="/favorites" element={<Favorites />} />

<Route path="/design/:designId" element={<DesignDetail />} />

<Route path="/expand/:designId" element={<ExpandDesign />} />

</Routes>

</Suspense>

</main>

</div>

</FavoritesProvider>

</DesignProvider>

</SettingsProvider>

</ErrorBoundary>

</BrowserRouter>

);

};

export default App;

### **Performance Monitoring**

Add performance monitoring with the Performance API:

// utils/performance.ts

interface PerformanceMetrics {

timeToFirstByte: number;

domContentLoaded: number;

firstPaint: number;

firstContentfulPaint: number;

timeToInteractive: number;

}

export const capturePerformanceMetrics = (): PerformanceMetrics | null => {

if (typeof window === 'undefined' || !window.performance) {

return null;

}

const navigationTiming = performance.getEntriesByType('navigation')[0] as PerformanceNavigationTiming;

const paintTiming = performance.getEntriesByType('paint');

if (!navigationTiming) {

return null;

}

const firstPaint = paintTiming.find(entry => entry.name === 'first-paint')?.startTime || 0;

const firstContentfulPaint = paintTiming.find(entry => entry.name === 'first-contentful-paint')?.startTime || 0;

return {

timeToFirstByte: navigationTiming.responseStart - navigationTiming.requestStart,

domContentLoaded: navigationTiming.domContentLoadedEventEnd - navigationTiming.fetchStart,

firstPaint,

firstContentfulPaint,

timeToInteractive: navigationTiming.domInteractive - navigationTiming.fetchStart

};

};

// Usage example

export const logPerformanceMetrics = () => {

window.addEventListener('load', () => {

// Wait for all resource loading to complete

setTimeout(() => {

const metrics = capturePerformanceMetrics();

if (metrics) {

console.log('Performance Metrics:', metrics);

// Could send to analytics service

}

}, 0);

});

};

## **Conclusion**

This project documentation provides a comprehensive guide for building the WebDesign AI Generator application. The architecture and implementation details are designed to create a clean, maintainable, and performant web application that leverages AI capabilities to generate website designs.

Key components of the implementation include:

* Modular React component architecture
* TypeScript for type safety
* Context API for state management
* AI service integration (Gemini 2.0 Flash and Claude 3.7 Sonnet)
* Responsive UI with Tailwind CSS
* Performance optimizations
* Accessibility considerations
* Error handling strategy
* Testing approach

Follow this documentation to implement each feature systematically, focusing on code quality and user experience. The modular approach allows for easy expansion and maintenance as the application evolves.

Start by setting up the project structure, then implement the core components and services, and finally add the advanced features and optimizations as needed. This incremental approach will help you build a robust application while maintaining code quality throughout the development process.