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

import {

User, Crown, Briefcase, Star, Zap, Lock, Mail, Eye, EyeOff, CheckCircle,

Settings, Palette, BarChart3, Code, Lightbulb, Library, DollarSign,

Plus, Edit3, Save, Copy, RefreshCw, Send, MessageSquare, Target,

TrendingUp, Users, Calendar, Award, Shield, Search, Filter,

ChevronRight, ChevronDown, X, Check, AlertTriangle, Info

} from 'lucide-react';

// Contexts for state management

const AuthContext = createContext();

const NotificationContext = createContext();

const AppStateContext = createContext();

// Auth Provider

const AuthProvider = ({ children }) => {

const [isAuthenticated, setIsAuthenticated] = useState(false);

const [user, setUser] = useState(null);

const login = (email, licenseType = 'basic') => {

const userData = {

email,

licenseType,

name: email.split('@')[0],

joinDate: new Date().toISOString(),

usage: {

promptsCreated: Math.floor(Math.random() \* 100),

templatesUsed: Math.floor(Math.random() \* 50),

apiCalls: Math.floor(Math.random() \* 1000)

}

};

setUser(userData);

setIsAuthenticated(true);

};

const logout = () => {

setUser(null);

setIsAuthenticated(false);

};

return (

<AuthContext.Provider value={{ isAuthenticated, user, login, logout }}>

{children}

</AuthContext.Provider>

);

};

// Notification Provider

const NotificationProvider = ({ children }) => {

const [notifications, setNotifications] = useState([]);

const addNotification = (message, type = 'info') => {

const id = Date.now();

setNotifications(prev => [...prev, { id, message, type }]);

setTimeout(() => {

setNotifications(prev => prev.filter(n => n.id !== id));

}, 5000);

};

return (

<NotificationContext.Provider value={{ notifications, addNotification }}>

{children}

</NotificationContext.Provider>

);

};

// App State Provider

const AppStateProvider = ({ children }) => {

const [currentSection, setCurrentSection] = useState('prompt-builder');

const [prompts, setPrompts] = useState([]);

const [templates, setTemplates] = useState([

{

id: 1,

title: "Blog Post Writer",

description: "SEO-optimized engaging blog posts with viral potential",

category: "Content",

prompt: "Write a comprehensive blog post about [TOPIC]. Include:\n\n1. Attention-grabbing headline\n2. SEO-optimized introduction\n3. Well-structured main content with subheadings\n4. Actionable takeaways\n5. Compelling conclusion with call-to-action\n\nTone: [TONE]\nTarget audience: [AUDIENCE]\nKeywords to include: [KEYWORDS]",

isPremium: false

},

{

id: 2,

title: "Viral Social Media Campaign",

description: "Multi-platform viral content strategy with engagement hooks",

category: "Marketing",

prompt: "Create a viral social media campaign for [PRODUCT/SERVICE]:\n\n1. Hook: Start with attention-grabbing opening\n2. Value: Clearly state the benefit\n3. Proof: Include social proof or testimonials\n4. Urgency: Create time-sensitive motivation\n5. CTA: Strong call-to-action\n\nPlatforms: [PLATFORMS]\nTarget demographic: [DEMOGRAPHIC]\nCampaign goal: [GOAL]",

isPremium: true

},

{

id: 3,

title: "Professional Email Template",

description: "High-converting business emails for any situation",

category: "Business",

prompt: "Write a professional email for [PURPOSE]:\n\nSubject Line: [Create compelling subject]\n\nEmail Structure:\n1. Personal greeting\n2. Context/reason for email\n3. Main request/proposal\n4. Clear next steps\n5. Professional closing\n\nTone: [TONE]\nRecipient: [RECIPIENT\_TYPE]\nDesired outcome: [OUTCOME]",

isPremium: false

},

{

id: 4,

title: "Code Documentation Generator",

description: "Clear, comprehensive code documentation and comments",

category: "Development",

prompt: "Create comprehensive documentation for this code:\n\n[CODE\_SNIPPET]\n\nInclude:\n1. Purpose and functionality overview\n2. Parameter descriptions\n3. Return value explanation\n4. Usage examples\n5. Error handling notes\n6. Performance considerations\n\nLanguage: [PROGRAMMING\_LANGUAGE]\nAudience: [DEVELOPER\_LEVEL]",

isPremium: true

},

{

id: 5,

title: "Creative Story Generator",

description: "Engaging narratives with compelling characters and plots",

category: "Creative",

prompt: "Write a creative story with these elements:\n\nSetting: [SETTING]\nMain character: [CHARACTER]\nConflict: [CONFLICT]\nGenre: [GENRE]\n\nStructure:\n1. Compelling opening hook\n2. Character development\n3. Rising action with obstacles\n4. Climactic moment\n5. Satisfying resolution\n\nLength: [LENGTH]\nTarget audience: [AUDIENCE]",

isPremium: false

}

]);

const addPrompt = (prompt) => {

const newPrompt = {

id: Date.now(),

...prompt,

createdAt: new Date().toISOString()

};

setPrompts(prev => [newPrompt, ...prev]);

};

return (

<AppStateContext.Provider value={{

currentSection,

setCurrentSection,

prompts,

setPrompts,

addPrompt,

templates,

setTemplates

}}>

{children}

</AppStateContext.Provider>

);

};

// Notification Component

const NotificationDisplay = () => {

const { notifications } = useContext(NotificationContext);

return (

<div className="fixed top-4 right-4 z-50 space-y-2">

{notifications.map(notification => (

<div

key={notification.id}

className={`p-4 rounded-lg shadow-lg max-w-sm ${

notification.type === 'success' ? 'bg-green-500 text-white' :

notification.type === 'error' ? 'bg-red-500 text-white' :

notification.type === 'warning' ? 'bg-yellow-500 text-white' :

'bg-blue-500 text-white'

}`}

>

{notification.message}

</div>

))}

</div>

);

};

// Login Component

const LoginForm = () => {

const [email, setEmail] = useState('');

const [password, setPassword] = useState('');

const [showPassword, setShowPassword] = useState(false);

const [isSignUp, setIsSignUp] = useState(false);

const { login } = useContext(AuthContext);

const { addNotification } = useContext(NotificationContext);

const handleSubmit = () => {

if (email && password) {

login(email);

addNotification(`Welcome ${isSignUp ? 'to' : 'back to'} Perfect Prompt Builder!`, 'success');

}

};

const handleDemoLogin = (licenseType) => {

const emails = {

basic: 'demo@basic.com',

pro: 'demo@pro.com',

enterprise: 'demo@enterprise.com',

whitelabel: 'demo@whitelabel.com'

};

login(emails[licenseType], licenseType);

addNotification(`Logged in as ${licenseType.toUpperCase()} demo user!`, 'success');

};

return (

<div className="min-h-screen bg-gradient-to-br from-blue-50 via-white to-purple-50 flex items-center justify-center p-4">

<div className="max-w-md w-full bg-white rounded-2xl shadow-xl p-8">

<div className="text-center mb-8">

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

<Zap className="h-8 w-8 text-blue-600 mr-2" />

<span className="text-2xl font-bold text-gray-900">PERFECT PROMPT BUILDER</span>

</div>

<h2 className="text-xl text-gray-600">

{isSignUp ? 'Create your account' : 'Sign in to continue building perfect prompts'}

</h2>

</div>

<div className="space-y-4 mb-6">

<div>

<label className="block text-sm font-medium text-gray-700 mb-2">Email</label>

<input

type="email"

value={email}

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

className="w-full px-4 py-3 border border-gray-300 rounded-lg focus:ring-2 focus:ring-blue-500 focus:border-blue-500"

placeholder="your@email.com"

/>

</div>

<div>

<label className="block text-sm font-medium text-gray-700 mb-2">Password</label>

<div className="relative">

<input

type={showPassword ? 'text' : 'password'}

value={password}

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

className="w-full px-4 py-3 border border-gray-300 rounded-lg focus:ring-2 focus:ring-blue-500 focus:border-blue-500 pr-12"

placeholder="Your password"

/>

<button

onClick={() => setShowPassword(!showPassword)}

className="absolute right-3 top-3 text-gray-400 hover:text-gray-600"

>

{showPassword ? <EyeOff className="h-5 w-5" /> : <Eye className="h-5 w-5" />}

</button>

</div>

</div>

<button

onClick={handleSubmit}

className="w-full bg-blue-600 text-white py-3 rounded-lg hover:bg-blue-700 font-medium"

>

{isSignUp ? 'Create Account' : 'Sign In'}

</button>

</div>

<div className="text-center mb-6">

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

{isSignUp ? 'Already have an account?' : "Don't have an account?"}

</span>

<button

onClick={() => setIsSignUp(!isSignUp)}

className="text-blue-600 hover:text-blue-700 ml-2 font-medium"

>

{isSignUp ? 'Sign In' : 'Create Account'}

</button>

</div>

<div className="border-t border-gray-200 pt-6">

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

<Crown className="h-5 w-5 text-yellow-500 mr-2" />

<span className="text-lg font-semibold">Try All Plans Instantly!</span>

</div>

<p className="text-center text-sm text-gray-600 mb-4">

Experience every tier with pre-loaded demo accounts

</p>

<div className="grid grid-cols-2 gap-3">

<button

onClick={() => handleDemoLogin('basic')}

className="p-3 border border-gray-300 rounded-lg hover:border-blue-500 hover:bg-blue-50 transition-colors"

>

<Zap className="h-5 w-5 text-gray-600 mx-auto mb-1" />

<div className="text-xs font-medium">Basic Demo</div>

<div className="text-xs text-gray-500">7930+ Templates</div>

</button>

<button

onClick={() => handleDemoLogin('pro')}

className="p-3 border border-purple-300 rounded-lg hover:border-purple-500 hover:bg-purple-50 transition-colors"

>

<Crown className="h-5 w-5 text-purple-600 mx-auto mb-1" />

<div className="text-xs font-medium">Pro Demo</div>

<div className="text-xs text-gray-500">19760+ Templates + Teams</div>

</button>

<button

onClick={() => handleDemoLogin('enterprise')}

className="p-3 border border-blue-300 rounded-lg hover:border-blue-500 hover:bg-blue-50 transition-colors"

>

<Briefcase className="h-5 w-5 text-blue-600 mx-auto mb-1" />

<div className="text-xs font-medium">Enterprise Demo</div>

<div className="text-xs text-gray-500">49710+ Templates + API</div>

</button>

<button

onClick={() => handleDemoLogin('whitelabel')}

className="p-3 border border-green-300 rounded-lg hover:border-green-500 hover:bg-green-50 transition-colors"

>

<Star className="h-5 w-5 text-green-600 mx-auto mb-1" />

<div className="text-xs font-medium">White Label Demo</div>

<div className="text-xs text-gray-500">997 Full Branding</div>

</button>

</div>

<p className="text-xs text-center text-gray-500 mt-4">

<CheckCircle className="h-4 w-4 inline mr-1" />

No registration required • Full feature access • Reset anytime

</p>

</div>

</div>

</div>

);

};

// Sidebar Navigation

const Sidebar = () => {

const { user } = useContext(AuthContext);

const { currentSection, setCurrentSection } = useContext(AppStateContext);

const { addNotification } = useContext(NotificationContext);

const navigationItems = [

{ id: 'prompt-builder', icon: Edit3, label: 'Prompt Builder', description: 'Build perfect prompts' },

{ id: 'templates', icon: Library, label: 'Template Library', description: 'Browse professional templates' },

{ id: 'pricing', icon: DollarSign, label: 'Pricing', description: 'View all plans' },

{ id: 'analytics', icon: BarChart3, label: 'Privacy Management', description: 'Requires Pro license', premium: true },

{ id: 'advanced-analytics', icon: TrendingUp, label: 'Advanced Analytics', description: 'Requires Pro license', premium: true },

{ id: 'api', icon: Code, label: 'API Usage', description: 'Requires Enterprise license', enterprise: true },

{ id: 'branding', icon: Palette, label: 'White Label Branding', description: 'Requires Whitelabel license', whitelabel: true },

{ id: 'profile', icon: User, label: 'Profile', description: 'Account settings' }

];

const handleNavigation = (item) => {

if (item.premium && user?.licenseType === 'basic') {

addNotification('Upgrade to Pro to access this feature!', 'warning');

setCurrentSection('pricing');

return;

}

if (item.enterprise && !['enterprise', 'whitelabel'].includes(user?.licenseType)) {

addNotification('Upgrade to Enterprise to access this feature!', 'warning');

setCurrentSection('pricing');

return;

}

if (item.whitelabel && user?.licenseType !== 'whitelabel') {

addNotification('Upgrade to White Label to access this feature!', 'warning');

setCurrentSection('pricing');

return;

}

setCurrentSection(item.id);

};

return (

<div className="w-64 bg-white border-r border-gray-200 h-full flex flex-col">

<div className="p-6 border-b border-gray-200">

<div className="flex items-center">

<Zap className="h-6 w-6 text-blue-600 mr-2" />

<span className="text-lg font-bold text-gray-900">PERFECT PROMPT BUILDER</span>

</div>

<div className="mt-2 text-sm text-gray-600">

Welcome back, {user?.email}

</div>

</div>

<nav className="flex-1 p-4 space-y-2">

{navigationItems.map((item) => {

const Icon = item.icon;

const isActive = currentSection === item.id;

const isLocked = (item.premium && user?.licenseType === 'basic') ||

(item.enterprise && !['enterprise', 'whitelabel'].includes(user?.licenseType)) ||

(item.whitelabel && user?.licenseType !== 'whitelabel');

return (

<button

key={item.id}

onClick={() => handleNavigation(item)}

className={`w-full text-left p-3 rounded-lg transition-colors flex items-start ${

isActive

? 'bg-blue-50 text-blue-700 border border-blue-200'

: 'hover:bg-gray-50 text-gray-700'

}`}

>

<Icon className={`h-5 w-5 mt-0.5 mr-3 ${isActive ? 'text-blue-600' : 'text-gray-500'}`} />

<div className="flex-1">

<div className="font-medium flex items-center">

{item.label}

{isLocked && <Lock className="h-4 w-4 ml-1 text-gray-400" />}

</div>

<div className="text-xs text-gray-500 mt-1">{item.description}</div>

</div>

</button>

);

})}

</nav>

</div>

);

};

// Prompt Builder Component

const PromptBuilder = () => {

const [prompt, setPrompt] = useState('');

const [title, setTitle] = useState('');

const [category, setCategory] = useState('general');

const [variables, setVariables] = useState([]);

const [generatedPrompt, setGeneratedPrompt] = useState('');

const [isGenerating, setIsGenerating] = useState(false);

const { addPrompt } = useContext(AppStateContext);

const { addNotification } = useContext(NotificationContext);

const categories = ['General', 'Business', 'Creative', 'Technical', 'Marketing', 'Education'];

const detectVariables = (text) => {

const matches = text.match(/\[([A-Z\_]+)\]/g);

return matches ? matches.map(match => match.slice(1, -1)) : [];

};

const handlePromptChange = (value) => {

setPrompt(value);

const detectedVars = detectVariables(value);

setVariables(detectedVars);

};

const generatePrompt = async () => {

if (!prompt.trim()) {

addNotification('Please enter a prompt first!', 'error');

return;

}

setIsGenerating(true);

addNotification('Generating improved prompt with Claude AI...', 'info');

try {

// Use serverless function for production, direct API for local development

const apiUrl = process.env.NODE\_ENV === 'production'

? '/api/claude'

: 'https://api.anthropic.com/v1/messages';

let response;

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

// Production: Use serverless function

response = await fetch(apiUrl, {

method: 'POST',

headers: {

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

},

body: JSON.stringify({

prompt,

category,

title

})

});

const data = await response.json();

if (data.success) {

setGeneratedPrompt(data.improvedPrompt);

// Update detected variables from the improved prompt

const newVariables = detectVariables(data.improvedPrompt);

setVariables(newVariables);

addNotification('Prompt improved successfully with Claude AI!', 'success');

} else {

throw new Error(data.error || 'Failed to improve prompt');

}

} else {

// Local development: Direct API call (will CORS error, fallback)

throw new Error('CORS - using fallback for local development');

}

} catch (error) {

console.error('API Error:', error);

addNotification('Using enhanced local processing (deploy for full AI power)', 'warning');

// Enhanced fallback processing

let improvedPrompt = prompt;

// Add structure and improvements

if (category === 'business') {

improvedPrompt = `Professional ${title}:\n\n${prompt}\n\nEnsure the response is:\n- Professional and clear\n- Action-oriented\n- Includes specific examples\n- Follows business communication standards`;

} else if (category === 'creative') {

improvedPrompt = `Creative ${title}:\n\n${prompt}\n\nMake the response:\n- Engaging and imaginative\n- Rich in descriptive detail\n- Emotionally compelling\n- Original and unique`;

} else if (category === 'technical') {

improvedPrompt = `Technical ${title}:\n\n${prompt}\n\nEnsure the response:\n- Is technically accurate\n- Includes step-by-step instructions\n- Provides relevant examples\n- Considers edge cases and best practices`;

} else {

improvedPrompt = `${title}:\n\n${prompt}\n\nPlease provide a comprehensive response that:\n- Directly addresses the request\n- Includes relevant examples\n- Is well-structured and clear\n- Provides actionable information`;

}

setGeneratedPrompt(improvedPrompt);

// Update variables

const newVariables = detectVariables(improvedPrompt);

setVariables(newVariables);

} finally {

setIsGenerating(false);

}

};

const savePrompt = () => {

if (!title.trim() || !prompt.trim()) {

addNotification('Please add a title and prompt!', 'error');

return;

}

addPrompt({

title,

prompt,

category,

variables,

generatedPrompt

});

addNotification('Prompt saved successfully!', 'success');

// Reset form

setTitle('');

setPrompt('');

setCategory('general');

setVariables([]);

setGeneratedPrompt('');

};

const copyToClipboard = (text) => {

navigator.clipboard.writeText(text);

addNotification('Copied to clipboard!', 'success');

};

return (

<div className="p-8 max-w-4xl mx-auto">

<div className="mb-8">

<h1 className="text-3xl font-bold text-gray-900 mb-2">Build Your Perfect Prompt</h1>

<p className="text-gray-600">Create, optimize, and save professional prompts with variable support</p>

</div>

<div className="grid grid-cols-1 lg:grid-cols-2 gap-8">

{/\* Input Section \*/}

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

<div>

<label className="block text-sm font-medium text-gray-700 mb-2">Prompt Title</label>

<input

type="text"

value={title}

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

className="w-full px-4 py-3 border border-gray-300 rounded-lg focus:ring-2 focus:ring-blue-500 focus:border-blue-500"

placeholder="Enter a descriptive title..."

/>

</div>

<div>

<label className="block text-sm font-medium text-gray-700 mb-2">Category</label>

<select

value={category}

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

className="w-full px-4 py-3 border border-gray-300 rounded-lg focus:ring-2 focus:ring-blue-500 focus:border-blue-500"

>

{categories.map(cat => (

<option key={cat} value={cat.toLowerCase()}>{cat}</option>

))}

</select>

</div>

<div>

<label className="block text-sm font-medium text-gray-700 mb-2">

Prompt Content

<span className="text-xs text-gray-500 ml-2">(Use [VARIABLE\_NAME] for dynamic content)</span>

</label>

<textarea

value={prompt}

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

className="w-full h-64 px-4 py-3 border border-gray-300 rounded-lg focus:ring-2 focus:ring-blue-500 focus:border-blue-500 resize-none"

placeholder="Write your prompt here... Use [TOPIC], [AUDIENCE], [TONE] for variables"

/>

</div>

{variables.length > 0 && (

<div>

<label className="block text-sm font-medium text-gray-700 mb-2">Detected Variables</label>

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

{variables.map(variable => (

<span key={variable} className="px-3 py-1 bg-blue-100 text-blue-800 rounded-full text-sm">

[{variable}]

</span>

))}

</div>

</div>

)}

<div className="flex gap-3">

<button

onClick={generatePrompt}

disabled={isGenerating}

className="flex-1 bg-blue-600 text-white py-3 px-4 rounded-lg hover:bg-blue-700 disabled:bg-blue-400 font-medium flex items-center justify-center"

>

{isGenerating ? (

<>

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

Improving with Claude AI...

</>

) : (

<>

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

Improve with Claude AI

</>

)}

</button>

<button

onClick={savePrompt}

className="flex-1 bg-green-600 text-white py-3 px-4 rounded-lg hover:bg-green-700 font-medium flex items-center justify-center"

>

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

Save Prompt

</button>

</div>

</div>

{/\* Output Section \*/}

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

<div>

<div className="flex items-center justify-between mb-2">

<label className="block text-sm font-medium text-gray-700">Claude AI Improved Prompt</label>

{generatedPrompt && (

<button

onClick={() => copyToClipboard(generatedPrompt)}

className="text-sm text-blue-600 hover:text-blue-700 flex items-center"

>

<Copy className="h-4 w-4 mr-1" />

Copy

</button>

)}

</div>

<div className="h-64 p-4 border border-gray-300 rounded-lg bg-gray-50 overflow-y-auto">

{isGenerating ? (

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

<RefreshCw className="h-6 w-6 animate-spin text-blue-600 mr-2" />

<span className="text-gray-600">Claude AI is improving your prompt...</span>

</div>

) : generatedPrompt ? (

<pre className="whitespace-pre-wrap text-sm text-gray-800">{generatedPrompt}</pre>

) : (

<p className="text-gray-500 text-sm">AI-improved prompt will appear here...</p>

)}

</div>

</div>

<div>

<h3 className="text-lg font-medium text-gray-900 mb-3">Quick Tips</h3>

<div className="space-y-2 text-sm text-gray-600">

<div className="flex items-start">

<Lightbulb className="h-4 w-4 text-yellow-500 mr-2 mt-0.5" />

<span>Use [BRACKETS] for variables that users can customize</span>

</div>

<div className="flex items-start">

<Target className="h-4 w-4 text-blue-500 mr-2 mt-0.5" />

<span>Be specific about desired output format and length</span>

</div>

<div className="flex items-start">

<MessageSquare className="h-4 w-4 text-green-500 mr-2 mt-0.5" />

<span>Include context and examples for better results</span>

</div>

</div>

</div>

</div>

</div>

</div>

);

};

// Templates Component

const Templates = () => {

const { templates } = useContext(AppStateContext);

const { user } = useContext(AuthContext);

const { addNotification } = useContext(NotificationContext);

const { setCurrentSection } = useContext(AppStateContext);

const [selectedCategory, setSelectedCategory] = useState('all');

const [searchTerm, setSearchTerm] = useState('');

const [selectedTemplate, setSelectedTemplate] = useState(null);

const categories = ['all', 'content', 'marketing', 'business', 'development', 'creative'];

const filteredTemplates = templates.filter(template => {

const matchesCategory = selectedCategory === 'all' || template.category.toLowerCase() === selectedCategory;

const matchesSearch = template.title.toLowerCase().includes(searchTerm.toLowerCase()) ||

template.description.toLowerCase().includes(searchTerm.toLowerCase());

return matchesCategory && matchesSearch;

});

const handleUseTemplate = (template) => {

if (template.isPremium && user?.licenseType === 'basic') {

addNotification('Upgrade to Pro to access premium templates!', 'warning');

setCurrentSection('pricing');

return;

}

setSelectedTemplate(template);

addNotification(`Loaded template: ${template.title}`, 'success');

};

const closeTemplate = () => {

setSelectedTemplate(null);

};

const copyTemplate = () => {

if (selectedTemplate) {

navigator.clipboard.writeText(selectedTemplate.prompt);

addNotification('Template copied to clipboard!', 'success');

}

};

const getLicenseInfo = () => {

const counts = {

basic: 30,

pro: 150,

enterprise: 500,

whitelabel: 997

};

return counts[user?.licenseType] || 30;

};

return (

<div className="p-8">

<div className="mb-8">

<h1 className="text-3xl font-bold text-gray-900 mb-2">Professional Template Library</h1>

<div className="flex items-center text-gray-600">

<Library className="h-5 w-5 mr-2" />

<span>{filteredTemplates.length} of {getLicenseInfo()} templates available</span>

</div>

</div>

{user?.licenseType === 'basic' && (

<div className="mb-6 p-4 bg-yellow-50 border border-yellow-200 rounded-lg">

<div className="flex items-center">

<Crown className="h-5 w-5 text-yellow-600 mr-2" />

<span className="font-medium text-yellow-800">Unlock 120 More Templates</span>

</div>

<p className="text-yellow-700 text-sm mt-1">

Upgrade to Pro for 60+ premium templates and advanced features

</p>

<button

onClick={() => setCurrentSection('pricing')}

className="mt-2 px-4 py-2 bg-yellow-600 text-white rounded-lg hover:bg-yellow-700 text-sm"

>

Upgrade Now

</button>

</div>

)}

{/\* Search and Filters \*/}

<div className="flex gap-4 mb-6">

<div className="flex-1 relative">

<Search className="h-5 w-5 absolute left-3 top-3 text-gray-400" />

<input

type="text"

placeholder="Search templates..."

value={searchTerm}

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

className="w-full pl-10 pr-4 py-3 border border-gray-300 rounded-lg focus:ring-2 focus:ring-blue-500 focus:border-blue-500"

/>

</div>

<select

value={selectedCategory}

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

className="px-4 py-3 border border-gray-300 rounded-lg focus:ring-2 focus:ring-blue-500 focus:border-blue-500"

>

{categories.map(category => (

<option key={category} value={category}>

{category.charAt(0).toUpperCase() + category.slice(1)}

</option>

))}

</select>

</div>

{/\* Templates Grid \*/}

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

{filteredTemplates.map(template => (

<div key={template.id} className="bg-white border border-gray-200 rounded-lg p-6 hover:shadow-md transition-shadow">

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

<div>

<h3 className="font-semibold text-gray-900">{template.title}</h3>

<span className="text-xs text-gray-500 uppercase tracking-wide">{template.category}</span>

</div>

{template.isPremium && (

<Crown className="h-4 w-4 text-yellow-500" />

)}

</div>

<p className="text-gray-600 text-sm mb-4">{template.description}</p>

<button

onClick={() => handleUseTemplate(template)}

className="w-full bg-blue-600 text-white py-2 px-4 rounded-lg hover:bg-blue-700 text-sm font-medium"

>

Use Template

</button>

</div>

))}

</div>

{/\* Template Modal \*/}

{selectedTemplate && (

<div className="fixed inset-0 bg-black bg-opacity-50 flex items-center justify-center p-4 z-50">

<div className="bg-white rounded-lg max-w-2xl w-full max-h-[80vh] overflow-hidden">

<div className="flex items-center justify-between p-6 border-b border-gray-200">

<h2 className="text-xl font-semibold text-gray-900">{selectedTemplate.title}</h2>

<button onClick={closeTemplate} className="text-gray-400 hover:text-gray-600">

<X className="h-5 w-5" />

</button>

</div>

<div className="p-6 overflow-y-auto">

<p className="text-gray-600 mb-4">{selectedTemplate.description}</p>

<div className="mb-4">

<label className="block text-sm font-medium text-gray-700 mb-2">Template Content</label>

<div className="bg-gray-50 border border-gray-200 rounded-lg p-4">

<pre className="whitespace-pre-wrap text-sm text-gray-800">{selectedTemplate.prompt}</pre>

</div>

</div>

<div className="flex gap-3">

<button

onClick={copyTemplate}

className="flex-1 bg-blue-600 text-white py-2 px-4 rounded-lg hover:bg-blue-700 flex items-center justify-center"

>

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

Copy Template

</button>

<button

onClick={() => {

setCurrentSection('prompt-builder');

closeTemplate();

}}

className="flex-1 bg-green-600 text-white py-2 px-4 rounded-lg hover:bg-green-700 flex items-center justify-center"

>

<Edit3 className="h-4 w-4 mr-2" />

Edit in Builder

</button>

</div>

</div>

</div>

</div>

)}

</div>

);

};

// Pricing Component

const Pricing = () => {

const { user } = useContext(AuthContext);

const { addNotification } = useContext(NotificationContext);

const plans = [

{

id: 'basic',

name: 'Basic',

price: 'Free',

icon: Zap,

color: 'gray',

features: [

'30 professional templates',

'Basic prompt builder',

'Community support',

'Standard generation speed'

],

limitations: [

'Limited template access',

'No advanced features',

'No priority support'

]

},

{

id: 'pro',

name: 'Pro',

price: '$29/month',

icon: Crown,

color: 'purple',

popular: true,

features: [

'150+ premium templates',

'Advanced prompt builder',

'Team collaboration',

'Priority generation',

'Analytics dashboard',

'Email support'

],

limitations: [

'No API access',

'Limited customization'

]

},

{

id: 'enterprise',

name: 'Enterprise',

price: '$99/month',

icon: Briefcase,

color: 'blue',

features: [

'500+ enterprise templates',

'Full API access',

'Advanced analytics',

'Custom integrations',

'Dedicated support',

'SSO authentication',

'Custom training'

],

limitations: []

},

{

id: 'whitelabel',

name: 'White Label',

price: '$299/month',

icon: Star,

color: 'green',

features: [

'Everything in Enterprise',

'Complete white labeling',

'Custom branding',

'Reseller program',

'Revenue sharing',

'Dedicated account manager'

],

limitations: []

}

];

const handleUpgrade = (planId) => {

addNotification(`Upgrade to ${plans.find(p => p.id === planId)?.name} feature coming soon!`, 'info');

};

return (

<div className="p-8">

<div className="text-center mb-12">

<h1 className="text-3xl font-bold text-gray-900 mb-4">Choose Your Perfect Plan</h1>

<p className="text-gray-600 max-w-2xl mx-auto">

Unlock the full power of AI prompt engineering with our professional plans.

Start free and scale as you grow.

</p>

</div>

<div className="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-4 gap-8 max-w-7xl mx-auto">

{plans.map(plan => {

const Icon = plan.icon;

const isCurrentPlan = user?.licenseType === plan.id;

return (

<div

key={plan.id}

className={`relative bg-white rounded-2xl shadow-lg border-2 transition-transform hover:scale-105 ${

plan.popular ? 'border-purple-200 ring-4 ring-purple-50' : 'border-gray-200'

}`}

>

{plan.popular && (

<div className="absolute -top-3 left-1/2 transform -translate-x-1/2">

<span className="bg-purple-600 text-white px-4 py-1 rounded-full text-sm font-medium">

Most Popular

</span>

</div>

)}

<div className="p-6">

<div className="text-center mb-6">

<div className={`w-12 h-12 rounded-full flex items-center justify-center mx-auto mb-4 ${

plan.color === 'purple' ? 'bg-purple-100' :

plan.color === 'blue' ? 'bg-blue-100' :

plan.color === 'green' ? 'bg-green-100' :

'bg-gray-100'

}`}>

<Icon className={`h-6 w-6 ${

plan.color === 'purple' ? 'text-purple-600' :

plan.color === 'blue' ? 'text-blue-600' :

plan.color === 'green' ? 'text-green-600' :

'text-gray-600'

}`} />

</div>

<h3 className="text-xl font-bold text-gray-900">{plan.name}</h3>

<div className="text-3xl font-bold text-gray-900 mt-2">{plan.price}</div>

</div>

<div className="space-y-4 mb-6">

<div>

<h4 className="font-medium text-gray-900 mb-2">Includes:</h4>

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

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

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

<Check className="h-4 w-4 text-green-500 mt-0.5 mr-2 flex-shrink-0" />

<span className="text-sm text-gray-600">{feature}</span>

</li>

))}

</ul>

</div>

{plan.limitations.length > 0 && (

<div>

<h4 className="font-medium text-gray-900 mb-2">Limitations:</h4>

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

{plan.limitations.map((limitation, index) => (

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

<X className="h-4 w-4 text-gray-400 mt-0.5 mr-2 flex-shrink-0" />

<span className="text-sm text-gray-500">{limitation}</span>

</li>

))}

</ul>

</div>

)}

</div>

<button

onClick={() => handleUpgrade(plan.id)}

disabled={isCurrentPlan}

className={`w-full py-3 px-4 rounded-lg font-medium transition-colors ${

isCurrentPlan

? 'bg-gray-100 text-gray-500 cursor-not-allowed'

: plan.popular

? 'bg-purple-600 text-white hover:bg-purple-700'

: 'bg-gray-900 text-white hover:bg-gray-800'

}`}

>

{isCurrentPlan ? 'Current Plan' : `Upgrade to ${plan.name}`}

</button>

</div>

</div>

);

})}

</div>

<div className="mt-16 bg-gradient-to-r from-blue-50 to-purple-50 rounded-2xl p-8 text-center">

<h2 className="text-2xl font-bold text-gray-900 mb-4">Need a Custom Solution?</h2>

<p className="text-gray-600 mb-6 max-w-2xl mx-auto">

For large teams, custom integrations, or specific requirements,

we offer tailored enterprise solutions with dedicated support.

</p>

<button

onClick={() => addNotification('Contact sales feature coming soon!', 'info')}

className="bg-blue-600 text-white px-8 py-3 rounded-lg hover:bg-blue-700 font-medium"

>

Contact Sales

</button>

</div>

</div>

);

};

// Analytics Component

const Analytics = () => {

const { user } = useContext(AuthContext);

const { prompts } = useContext(AppStateContext);

const stats = [

{ label: 'Prompts Created', value: user?.usage?.promptsCreated || 0, icon: Edit3, color: 'blue' },

{ label: 'Templates Used', value: user?.usage?.templatesUsed || 0, icon: Library, color: 'purple' },

{ label: 'API Calls', value: user?.usage?.apiCalls || 0, icon: Code, color: 'green' },

{ label: 'Success Rate', value: '94%', icon: TrendingUp, color: 'yellow' }

];

return (

<div className="p-8">

<div className="mb-8">

<h1 className="text-3xl font-bold text-gray-900 mb-2">Analytics Dashboard</h1>

<p className="text-gray-600">Track your prompt performance and usage statistics</p>

</div>

<div className="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-4 gap-6 mb-8">

{stats.map((stat, index) => {

const Icon = stat.icon;

return (

<div key={index} className="bg-white rounded-lg p-6 border border-gray-200">

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

<div>

<p className="text-gray-500 text-sm">{stat.label}</p>

<p className="text-2xl font-bold text-gray-900">{stat.value}</p>

</div>

<div className={`p-3 rounded-lg ${

stat.color === 'blue' ? 'bg-blue-100' :

stat.color === 'purple' ? 'bg-purple-100' :

stat.color === 'green' ? 'bg-green-100' :

'bg-yellow-100'

}`}>

<Icon className={`h-6 w-6 ${

stat.color === 'blue' ? 'text-blue-600' :

stat.color === 'purple' ? 'text-purple-600' :

stat.color === 'green' ? 'text-green-600' :

'text-yellow-600'

}`} />

</div>

</div>

</div>

);

})}

</div>

<div className="bg-white rounded-lg border border-gray-200 p-6">

<h2 className="text-xl font-semibold text-gray-900 mb-4">Recent Activity</h2>

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

{prompts.length > 0 ? (

prompts.slice(0, 5).map(prompt => (

<div key={prompt.id} className="flex items-center p-4 bg-gray-50 rounded-lg">

<Edit3 className="h-5 w-5 text-blue-600 mr-3" />

<div className="flex-1">

<h3 className="font-medium text-gray-900">{prompt.title}</h3>

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

Created {new Date(prompt.createdAt).toLocaleDateString()}

</p>

</div>

<span className="text-xs bg-blue-100 text-blue-800 px-2 py-1 rounded-full">

{prompt.category}

</span>

</div>

))

) : (

<p className="text-gray-500 text-center py-8">No activity yet. Start building prompts to see analytics!</p>

)}

</div>

</div>

</div>

);

};

// Profile Component

const Profile = () => {

const { user, logout } = useContext(AuthContext);

const { addNotification } = useContext(NotificationContext);

const handleLogout = () => {

logout();

addNotification('Logged out successfully!', 'success');

};

return (

<div className="p-8 max-w-2xl">

<div className="mb-8">

<h1 className="text-3xl font-bold text-gray-900 mb-2">Profile Settings</h1>

<p className="text-gray-600">Manage your account and preferences</p>

</div>

<div className="bg-white rounded-lg border border-gray-200 p-6 mb-6">

<h2 className="text-xl font-semibold text-gray-900 mb-4">Account Information</h2>

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

<div>

<label className="block text-sm font-medium text-gray-700 mb-1">Email</label>

<div className="text-gray-900">{user?.email}</div>

</div>

<div>

<label className="block text-sm font-medium text-gray-700 mb-1">License Type</label>

<div className="flex items-center">

<span className={`px-3 py-1 rounded-full text-sm font-medium ${

user?.licenseType === 'basic' ? 'bg-gray-100 text-gray-800' :

user?.licenseType === 'pro' ? 'bg-purple-100 text-purple-800' :

user?.licenseType === 'enterprise' ? 'bg-blue-100 text-blue-800' :

'bg-green-100 text-green-800'

}`}>

{user?.licenseType?.toUpperCase()}

</span>

</div>

</div>

<div>

<label className="block text-sm font-medium text-gray-700 mb-1">Member Since</label>

<div className="text-gray-900">

{user?.joinDate ? new Date(user.joinDate).toLocaleDateString() : 'Today'}

</div>

</div>

</div>

</div>

<div className="bg-white rounded-lg border border-gray-200 p-6">

<h2 className="text-xl font-semibold text-gray-900 mb-4">Account Actions</h2>

<div className="space-y-3">

<button

onClick={() => addNotification('Password change feature coming soon!', 'info')}

className="w-full text-left p-3 border border-gray-200 rounded-lg hover:bg-gray-50 transition-colors"

>

<div className="font-medium text-gray-900">Change Password</div>

<div className="text-sm text-gray-500">Update your account password</div>

</button>

<button

onClick={() => addNotification('Export feature coming soon!', 'info')}

className="w-full text-left p-3 border border-gray-200 rounded-lg hover:bg-gray-50 transition-colors"

>

<div className="font-medium text-gray-900">Export Data</div>

<div className="text-sm text-gray-500">Download your prompts and data</div>

</button>

<button

onClick={handleLogout}

className="w-full text-left p-3 border border-red-200 rounded-lg hover:bg-red-50 transition-colors"

>

<div className="font-medium text-red-600">Sign Out</div>

<div className="text-sm text-red-500">Sign out of your account</div>

</button>

</div>

</div>

</div>

);

};

// Main Application Component

const MainApp = () => {

const { currentSection } = useContext(AppStateContext);

const renderSection = () => {

switch (currentSection) {

case 'prompt-builder':

return <PromptBuilder />;

case 'templates':

return <Templates />;

case 'pricing':

return <Pricing />;

case 'analytics':

case 'advanced-analytics':

return <Analytics />;

case 'profile':

return <Profile />;

case 'api':

return (

<div className="p-8">

<h1 className="text-3xl font-bold text-gray-900 mb-4">API Documentation</h1>

<p className="text-gray-600">Enterprise API features coming soon...</p>

</div>

);

case 'branding':

return (

<div className="p-8">

<h1 className="text-3xl font-bold text-gray-900 mb-4">White Label Branding</h1>

<p className="text-gray-600">Customize the platform with your branding...</p>

</div>

);

default:

return <PromptBuilder />;

}

};

return (

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

<Sidebar />

<div className="flex-1 overflow-auto">

{renderSection()}

</div>

</div>

);

};

// Root Component

const PerfectPromptBuilderComplete = () => {

const [isAuthenticated, setIsAuthenticated] = useState(false);

return (

<AuthProvider>

<NotificationProvider>

<AppStateProvider>

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

<AuthContext.Consumer>

{({ isAuthenticated, user }) => (

<>

{!isAuthenticated ? <LoginForm /> : <MainApp />}

<NotificationDisplay />

</>

)}

</AuthContext.Consumer>

</div>

</AppStateProvider>

</NotificationProvider>

</AuthProvider>

);

};

export default PerfectPromptBuilderComplete;