all_features.txt

```
// ===== AiFeatureBuilder.tsx =====



import React, { useState, useCallback, useRef } from 'react';
import type { GeneratedFile } from '../../types.ts';
import { generateFeature, generateUnitTests, generateCommitMessage } from
'../../services/geminiService.ts';
import { CpuChipIcon, DocumentIcon } from '../icons/InterfaceIcons.tsx';
import TerminalComponent, { TerminalHandle } from '../Terminal.tsx';
import { useGlobalState } from '../../contexts/GlobalStateContext.tsx';

const LoadingSpinner: React.FC = () => (
    <div className="flex items-center justify-center space-x-2">
        <div className="w-2 h-2 rounded-full bg-cyan-400 animate-pulse" style={{
        animationDelay: '0s' }}></div>
        <div className="w-2 h-2 rounded-full bg-cyan-400 animate-pulse" style={{
        animationDelay: '0.2s' }}></div>
        <div className="w-2 h-2 rounded-full bg-cyan-400 animate-pulse" style={{
        animationDelay: '0.4s' }}></div>
    </div>
);

type ActiveTab = 'CODE' | 'TESTS' | 'COMMIT';

export const AiFeatureBuilder: React.FC = () => {
    const { dispatch } = useGlobalState();
    const [prompt, setPrompt] = useState<string>('A simple "Hello World" React
    component with a button that shows an alert.');
    const [generatedFiles, setGeneratedFiles] = useState<GeneratedFile[]>([]);
    const [unitTests, setUnitTests] = useState<string>('');
    const [commitMessage, setCommitMessage] = useState<string>('');
    const [selectedFile, setSelectedFile] = useState<GeneratedFile | null>(null);
    const [isLoading, setIsLoading] = useState<boolean>(false);
    const [error, setError] = useState<string>('');
    const [activeTab, setActiveTab] = useState<ActiveTab>('CODE');
    const terminalRef = useRef<TerminalHandle>(null);

    const handleGenerate = useCallback(async () => {
        if (!prompt.trim()) {
            setError('Please enter a feature description.');
            return;
        }
        setIsLoading(true);
        setError('');
        setGeneratedFiles([]);
        setUnitTests('');
        setCommitMessage('');
        setSelectedFile(null);
        setActiveTab('CODE');
        terminalRef.current?.reset();
        terminalRef.current?.writeln('Sending request to Gemini for feature files...');

        try {
            // 1. Generate feature files
            const resultFiles = await generateFeature(prompt);
            setGeneratedFiles(resultFiles);
            dispatch({ type: 'SET_GENERATED_FILES', payload: resultFiles }); // <-- Update
```

```
        global state
        terminalRef.current?.writeln(`■ Feature files generated successfully!
        (${resultFiles.length} files)`);
        if (resultFiles.length > 0) {
            setSelectedFile(resultFiles[0]);
        }

        // 2. Generate Unit Tests
        const componentFile = resultFiles.find(f => f.filePath.endsWith('.tsx'));
        if (componentFile) {
            terminalRef.current?.writeln('■ Generating unit tests...');
            const tests = await generateUnitTests(componentFile.content);
            setUnitTests(tests);
            terminalRef.current?.writeln('■ Unit tests generated!');
        }

        // 3. Generate Commit Message
        terminalRef.current?.writeln('■ Generating commit message...');
        const diffContext = resultFiles.map(f => `File:
        ${f.filePath}\n\n${f.content}`).join('\n---\n');
        const commit = await generateCommitMessage(diffContext);
        setCommitMessage(commit);
        terminalRef.current?.writeln('■ Commit message generated!');
        terminalRef.current?.writeln('\n■ Feature package complete.');

    } catch (err) {
        const errorMessage = err instanceof Error ? err.message : 'An unknown error
        occurred.';
        setError(`Failed to generate feature: ${errorMessage}`);
        terminalRef.current?.writeln(`■ Error: ${errorMessage}`);
    } finally {
        setIsLoading(false);
    }
}, [prompt, dispatch]);

const renderContent = () => {
    switch (activeTab) {
        case 'TESTS':
            return <pre className="w-full h-full p-4 bg-transparent resize-none font-mono
            text-sm text-cyan-300 whitespace-pre-wrap">{unitTests}</pre>
        case 'COMMIT':
            return <pre className="w-full h-full p-4 bg-transparent resize-none font-sans
            text-sm text-slate-200 whitespace-pre-wrap">{commitMessage}</pre>
        case 'CODE':
        default:
             return selectedFile ? (
                <textarea
                    readOnly
                    value={selectedFile.content}
                    className="w-full h-full p-4 bg-transparent resize-none font-mono text-sm text-
                    cyan-300 focus:outline-none"
                />
            ) : (
                <div className="flex items-center justify-center h-full text-slate-500">
                    Select a file to view its content.
                </div>
            )
    }
}

return (
    <div className="h-full flex flex-col text-slate-300">
```

```
<header className="p-4 border-b border-slate-800 flex-shrink-0">
    <h1 className="text-xl font-bold text-slate-100 flex items-center"><CpuChipIcon
    /><span className="ml-3">AI Feature Builder</span></h1>
    <p className="text-slate-400 mt-1 text-sm">Describe a new feature, and watch
    Gemini build the code, tests, and commit message.</p>
</header>

<div className="flex-grow flex min-h-0">
    <aside className="w-64 bg-slate-900/70 border-r border-slate-800 p-4 flex flex-
    col space-y-2 overflow-y-auto">
        <h2 className="text-sm font-semibold text-slate-400 mb-2">Generated Files</h2>
        {generatedFiles.map(file => (
            <div key={file.filePath} onClick={() => { setSelectedFile(file);
            setActiveTab('CODE'); }} className={`flex items-center space-x-2 p-2 rounded-md
            cursor-pointer text-sm ${selectedFile?.filePath === file.filePath && activeTab
            === 'CODE' ? 'bg-cyan-500/20 text-cyan-300' : 'hover:bg-slate-800'}`}>
                <DocumentIcon /><span>{file.filePath.split('/').pop()}</span>
            </div>
        ))}
    </aside>

    <main className="flex-1 flex flex-col min-w-0">
        <div className="flex-grow flex flex-col bg-slate-900">
             <div className="border-b border-slate-800 flex items-center justify-between">
                <div className="flex">
                    {generatedFiles.length > 0 && ['CODE', 'TESTS', 'COMMIT'].map(tab => {
                        if (tab === 'TESTS' && !unitTests) return null;
                        if (tab === 'COMMIT' && !commitMessage) return null;
                        return <button key={tab} onClick={() => setActiveTab(tab as ActiveTab)}
                        className={`px-4 py-2 text-sm ${activeTab === tab ? 'bg-slate-800 text-
                        slate-100' : 'text-slate-400'}`}>{tab}</button>
                    })}
                </div>
                {selectedFile && activeTab==='CODE' && <h3 className="font-mono text-sm text-
                slate-400 pr-4">{selectedFile.filePath}</h3>}
            </div>
            {renderContent()}
        </div>

        <div className="flex-shrink-0 h-64 border-t border-slate-800 flex">
            <div className="w-1/2 p-4 flex flex-col border-r border-slate-800">
                 <label htmlFor="prompt-input" className="text-sm font-medium text-slate-400
                 mb-2">Feature Request</label>
                <textarea id="prompt-input" value={prompt} onChange={(e) =>
                setPrompt(e.target.value)} placeholder="e.g., A user profile card with an
                avatar, name, and bio." className="flex-grow p-2 bg-slate-800 border border-
                slate-700 rounded-md resize-none text-sm text-slate-300"/>
                 <button onClick={handleGenerate} disabled={isLoading} className="mt-2 w-full
                 flex items-center justify-center gap-2 px-4 py-2 bg-cyan-500 text-slate-900
                 font-bold rounded-md hover:bg-cyan-400 disabled:bg-slate-600">
                    {isLoading ? <><LoadingSpinner /> Generating...</> : 'Generate Feature'}
                </button>
            </div>
            <div className="w-1/2 p-2 flex flex-col">
                <label className="text-sm font-medium text-slate-400 mb-1 px-2">Terminal</label>
                <div className="flex-grow">
                    <TerminalComponent ref={terminalRef} initialMessage='Welcome to the AI Feature
                    Builder terminal.' isReadOnly={true} />
                </div>
            </div>
        </div>
    </main>
```

```
            </div>
        </div>
    );
};

// ===== AiStyleTransfer.tsx =====



import React, { useState, useCallback } from 'react';
import { transferCodeStyleStream } from '../../services/geminiService.ts';
import { LoadingSpinner } from '../shared/LoadingSpinner.tsx';
import { marked } from 'marked';
import { useFeatureState } from '../../hooks/useFeatureState.ts';
import { FeatureHeader } from '../shared/FeatureHeader.tsx';
import type { Feature } from '../../types.ts';

const exampleCode = `function my_func(x,y){return x+y;}`;
const exampleStyleGuide = `- Use camelCase for function names.
- Add a space after commas in argument lists.
- Use semicolons at the end of statements.`;

export const AiStyleTransfer: React.FC<{ feature: Feature }> = ({ feature }) =>
{
    const [state, setState, saveState] = useFeatureState('styleTransfer', {
        inputCode: exampleCode,
        styleGuide: exampleStyleGuide,
        outputCode: '',
    });
    const { inputCode, styleGuide, outputCode } = state;

    const [isLoading, setIsLoading] = useState<boolean>(false);
    const [error, setError] = useState<string>('');

    const handleGenerate = useCallback(async () => {
        if (!inputCode.trim() || !styleGuide.trim()) {
            setError('Please provide both code and a style guide.');
            return;
        }
        setIsLoading(true);
        setError('');
        setState(s => ({ ...s, outputCode: '' }));
        try {
            const stream = transferCodeStyleStream({ code: inputCode, styleGuide });
            let fullResponse = '';
            for await (const chunk of stream) {
                fullResponse += chunk;
                setState(s => ({ ...s, outputCode: fullResponse }));
            }
        } catch (err) {
            const errorMessage = err instanceof Error ? err.message : 'An unknown error
            occurred.';
            setError(`Failed to transfer style: ${errorMessage}`);
        } finally {
            setIsLoading(false);
        }
    }, [inputCode, styleGuide, setState]);

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8">
            <FeatureHeader feature={feature} onSaveState={saveState} />
            <div className="flex-grow grid grid-cols-1 lg:grid-cols-2 gap-6 h-full overflow-
```

```
            hidden">
                <div className="flex flex-col h-full gap-4">
                    <div className="flex flex-col flex-1">
                        <label htmlFor="input-code" className="text-sm font-medium text-slate-400
                        mb-2">Original Code</label>
                        <textarea
                            id="input-code"
                            value={inputCode}
                            onChange={(e) => setState(s => ({ ...s, inputCode: e.target.value }))}
                            className="flex-grow p-4 bg-slate-900 border border-slate-700 rounded-md resize-
                            none font-mono text-sm"
                        />
                    </div>
                    <div className="flex flex-col flex-1">
                        <label htmlFor="style-guide" className="text-sm font-medium text-slate-400
                        mb-2">Style Guide</label>
                        <textarea
                            id="style-guide"
                            value={styleGuide}
                            onChange={(e) => setState(s => ({ ...s, styleGuide: e.target.value }))}
                            className="flex-grow p-4 bg-slate-900 border border-slate-700 rounded-md resize-
                            none font-mono text-sm"
                        />
                    </div>
                </div>
                <div className="flex flex-col h-full">
                    <label className="text-sm font-medium text-slate-400 mb-2">Rewritten
                    Code</label>
                    <div className="flex-grow p-1 bg-slate-800/50 border border-slate-700/50
                    rounded-md overflow-y-auto">
                        {isLoading && <div className="flex items-center justify-center
                        h-full"><LoadingSpinner /></div>}
                        {error && <p className="p-4 text-red-400">{error}</p>}
                        {outputCode && !isLoading && (
                            <div
                                className="prose prose-sm prose-invert max-w-none prose-pre:bg-transparent
                                prose-pre:p-4 prose-pre:m-0"
                                dangerouslySetInnerHTML={{ __html: marked.parse(outputCode) as string }}
                            />
                        )}
                         {!isLoading && !outputCode && !error && <div className="text-slate-500 h-full
                         flex items-center justify-center">Rewritten code will appear here.</div>}
                    </div>
                </div>
            </div>
            <div className="flex-shrink-0 pt-4">
                <button
                    onClick={handleGenerate}
                    disabled={isLoading}
                    className="w-full max-w-md mx-auto flex items-center justify-center px-6 py-3
                    bg-cyan-500 text-slate-900 font-bold rounded-md hover:bg-cyan-400 disabled:bg-
                    slate-600"
                >
                    {isLoading ? <LoadingSpinner /> : 'Rewrite Code'}
                </button>
            </div>
        </div>
    );
};

// ===== AiUnitTestGenerator.tsx =====
import React, { useState, useCallback } from 'react';
```

```
import { generateUnitTestsStream } from '../../services/geminiService.ts';
import { LoadingSpinner } from '../shared/LoadingSpinner.tsx';
import { marked } from 'marked';
import { useFeatureState } from '../../hooks/useFeatureState.ts';
import { FeatureHeader } from '../shared/FeatureHeader.tsx';
import type { Feature } from '../../types.ts';

const exampleCode = `export function calculateTotalPrice(items, taxRate) {
  const subtotal = items.reduce((sum, item) => sum + item.price * item.quantity,
  0);
  return subtotal * (1 + taxRate);
}`;

export const AiUnitTestGenerator: React.FC<{ feature: Feature }> = ({ feature })
=> {
    const [state, setState, saveState] = useFeatureState('unitTestGenerator', {
    code: exampleCode, tests: '' });
    const { code, tests } = state;

    const [isLoading, setIsLoading] = useState<boolean>(false);
    const [error, setError] = useState<string>('');

    const handleGenerate = useCallback(async () => {
        if (!code.trim()) {
            setError('Please enter some code to generate tests for.');
            return;
        }
        setIsLoading(true);
        setError('');
        setState(s => ({ ...s, tests: ''}));
        try {
            const stream = generateUnitTestsStream(code);
            let fullResponse = '';
            for await (const chunk of stream) {
                fullResponse += chunk;
                setState(s => ({ ...s, tests: fullResponse}));
            }
        } catch (err) {
            const errorMessage = err instanceof Error ? err.message : 'An unknown error
            occurred.';
            setError(`Failed to generate tests: ${errorMessage}`);
        } finally {
            setIsLoading(false);
        }
    }, [code, setState]);

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8">
            <FeatureHeader feature={feature} onSaveState={saveState} />
            <div className="flex-grow grid grid-cols-1 lg:grid-cols-2 gap-6 h-full overflow-
            hidden">
                <div className="flex flex-col h-full">
                    <label htmlFor="code-input" className="text-sm font-medium text-slate-400
                    mb-2">Source Code</label>
                    <textarea
                        id="code-input"
                        value={code}
                        onChange={(e) => setState(s => ({ ...s, code: e.target.value}))}
                        placeholder="Paste your source code here..."
                        className="flex-grow p-4 bg-slate-900 border border-slate-700 rounded-md resize-
                        none font-mono text-sm text-cyan-300 focus:ring-2 focus:ring-cyan-500
                        focus:outline-none"
```

```
                    />
                    <button
                        onClick={handleGenerate}
                        disabled={isLoading}
                        className="mt-4 w-full flex items-center justify-center px-6 py-3 bg-cyan-500
                        text-slate-900 font-bold rounded-md hover:bg-cyan-400 transition-colors
                        disabled:bg-slate-600 disabled:cursor-not-allowed"
                    >
                        {isLoading ? <LoadingSpinner /> : 'Generate Unit Tests'}
                    </button>
                </div>
                <div className="flex flex-col h-full">
                    <label className="text-sm font-medium text-slate-400 mb-2">Generated
                    Tests</label>
                    <div className="flex-grow p-1 bg-slate-800/50 border border-slate-700/50
                    rounded-md overflow-y-auto">
                        {isLoading && (
                            <div className="flex items-center justify-center h-full">
                                <LoadingSpinner />
                            </div>
                        )}
                        {error && <p className="p-4 text-red-400">{error}</p>}
                        {tests && !isLoading && (
                            <div
                                className="prose prose-sm prose-invert max-w-none prose-pre:bg-transparent
                                prose-pre:p-4 prose-pre:m-0 prose-code:text-cyan-300"
                                dangerouslySetInnerHTML={{ __html: marked.parse(tests) as string }}
                            />
                        )}
                        {!isLoading && !tests && !error && (
                            <div className="text-slate-500 h-full flex items-center justify-center">
                                The generated tests will appear here.
                            </div>
                        )}
                    </div>
                </div>
            </div>
        </div>
    );
};

// ===== AsyncCallTreeViewer.tsx =====


import React from 'react';
import { ChartBarIcon } from '../icons/FeatureIcons.tsx';

const mockAsyncTree = {
    name: 'startApp',
    duration: 500,
    children: [
        {
            name: 'fetchUserData',
            duration: 300,
            children: [
                { name: 'authenticate', duration: 100, children: [] },
                { name: 'fetchProfile', duration: 150, children: [] },
            ],
        },
        {
            name: 'loadInitialAssets',
            duration: 450,
```

```
            children: [
                { name: 'loadImage.png', duration: 200, children: [] },
                { name: 'loadScript.js', duration: 250, children: [] },
            ],
        },
    ],
};

const TreeNode: React.FC<{ node: typeof mockAsyncTree, level: number }> = ({
node, level }) => {
    const [isOpen, setIsOpen] = React.useState(true);
    const hasChildren = node.children && node.children.length > 0;

    return (
        <div>
            <div
                className="flex items-center p-2 rounded-md hover:bg-slate-800"
                style={{ marginLeft: `${level * 20}px` }}
            >
                {hasChildren && (
                    <button onClick={() => setIsOpen(!isOpen)} className="mr-2 text-slate-500 w-4
                    h-4">
                       {isOpen ? '▼' : '■'}
                    </button>
                )}
                 <div className="flex-grow flex items-center justify-between">
                    <span>{node.name}</span>
                    <span className="text-cyan-400">{node.duration}ms</span>
                </div>
            </div>
            {isOpen && hasChildren && (
                <div>
                    {node.children.map((child, index) => (
                        <TreeNode key={index} node={child} level={level + 1} />
                    ))}
                </div>
            )}
        </div>
    );
};


export const AsyncCallTreeViewer: React.FC = () => {
    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8">
            <header className="mb-6">
                <h1 className="text-3xl font-bold text-slate-100 flex items-center">
                    <ChartBarIcon />
                    <span className="ml-3">Async Call Tree (Simulation)</span>
                </h1>
                <p className="text-slate-400 mt-1">Visualize a simulated tree of asynchronous
                function calls.</p>
            </header>
            <div className="flex-grow bg-slate-900 p-4 rounded-lg font-mono text-sm
            overflow-y-auto">
                <TreeNode node={mockAsyncTree} level={0} />
            </div>
        </div>
    );
};

// ===== AudioToCode.tsx =====
```

```tsx
import React, { useState, useRef, useCallback } from 'react';
import { transcribeAudioToCodeStream } from '../../services/geminiService.ts';
import { CpuChipIcon } from '../icons/FeatureIcons.tsx';
import { LoadingSpinner } from '../shared/LoadingSpinner.tsx';
import { marked } from 'marked';
import { blobToBase64 } from '../../services/fileUtils.ts';

export const AudioToCode: React.FC = () => {
    const [isRecording, setIsRecording] = useState(false);
    const [isLoading, setIsLoading] = useState(false);
    const [code, setCode] = useState('');
    const [error, setError] = useState('');
    const mediaRecorderRef = useRef<MediaRecorder | null>(null);
    const audioChunksRef = useRef<Blob[]>([]);

    const handleStartRecording = async () => {
        setError('');
        setCode('');
        try {
            const stream = await navigator.mediaDevices.getUserMedia({ audio: true });
            mediaRecorderRef.current = new MediaRecorder(stream);
            mediaRecorderRef.current.ondataavailable = event => {
                audioChunksRef.current.push(event.data);
            };
            mediaRecorderRef.current.onstop = handleTranscribe;
            mediaRecorderRef.current.start();
            setIsRecording(true);
        } catch (err) {
            setError('Microphone access was denied. Please enable it in your browser
            settings.');
        }
    };

    const handleStopRecording = () => {
        mediaRecorderRef.current?.stop();
        setIsRecording(false);
        setIsLoading(true);
    };

    const handleTranscribe = async () => {
        const audioBlob = new Blob(audioChunksRef.current, { type: 'audio/webm' });
        audioChunksRef.current = [];
        try {
            const base64Audio = await blobToBase64(audioBlob);
            const stream = transcribeAudioToCodeStream(base64Audio, 'audio/webm');
            let fullResponse = '';
            for await (const chunk of stream) {
                fullResponse += chunk;
                setCode(fullResponse);
            }
        } catch (err) {
            const errorMessage = err instanceof Error ? err.message : 'An unknown error
            occurred.';
            setError(`Failed to transcribe audio: ${errorMessage}`);
        } finally {
            setIsLoading(false);
        }
    };

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8">
            <header className="mb-6">
```

```
                <h1 className="text-3xl font-bold text-slate-100 flex items-center">
                    <CpuChipIcon />
                    <span className="ml-3">AI Audio-to-Code</span>
                </h1>
                <p className="text-slate-400 mt-1">Speak your programming ideas and watch them
                turn into code.</p>
            </header>
            <div className="flex-grow grid grid-cols-1 lg:grid-cols-2 gap-6 min-h-0">
                <div className="flex flex-col items-center justify-center bg-slate-800/50 p-6
                rounded-lg">
                     <button
                        onClick={isRecording ? handleStopRecording : handleStartRecording}
                        className={`w-32 h-32 rounded-full flex items-center justify-center text-white
                        font-bold text-lg transition-all ${isRecording ? 'bg-red-500 animate-pulse' :
                        'bg-cyan-500'}`}
                        disabled={isLoading}
                    >
                        {isRecording ? 'Stop' : 'Record'}
                    </button>
                    <p className="mt-4 text-slate-400">
                        {isRecording ? 'Recording in progress...' : 'Click to start recording'}
                    </p>
                    {error && <p className="text-red-400 mt-4">{error}</p>}
                </div>
                 <div className="flex flex-col h-full">
                    <label className="text-sm font-medium text-slate-400 mb-2">Generated
                    Code</label>
                    <div className="flex-grow p-1 bg-slate-900 border border-slate-700 rounded-md
                    overflow-y-auto">
                        {isLoading && (
                            <div className="flex items-center justify-center h-full"><LoadingSpinner
                            /></div>
                        )}
                        {code && !isLoading && (
                             <div
                                className="prose prose-sm prose-invert max-w-none prose-pre:bg-transparent
                                prose-pre:p-4 prose-pre:m-0 prose-code:text-cyan-300"
                                dangerouslySetInnerHTML={{ __html: marked.parse(code) as string }}
                            />
                        )}
                        {!isLoading && !code && !error && (
                            <div className="text-slate-500 h-full flex items-center justify-center">Code
                            will appear here.</div>
                        )}
                    </div>
                </div>
            </div>
        </div>
    );
};

// ===== ChangelogGenerator.tsx =====

import React, { useState, useCallback } from 'react';
import { GitBranchIcon } from '../icons/FeatureIcons.tsx';
import { generateChangelogStream } from '../../services/geminiService.ts';
import { useFeatureState } from '../../hooks/useFeatureState.ts';
import { FeatureHeader } from '../shared/FeatureHeader.tsx';
import type { Feature } from '../../types.ts';
import { LoadingSpinner } from '../shared/LoadingSpinner.tsx';
import { MarkdownRenderer } from '../shared/MarkdownRenderer.tsx';
const sampleCommits = `feat: add user login page
```

```
fix: correct typo in header
docs: update readme with setup instructions
feat(api): implement user endpoint
chore: upgrade dependencies
fix(button): prevent double click`;

export const ChangelogGenerator: React.FC<{ feature: Feature }> = ({ feature })
=> {
    const [state, setState, saveState] = useFeatureState('changelogGenerator', {
    commits: sampleCommits, changelog: '' });
    const { commits, changelog } = state;

    const [isLoading, setIsLoading] = useState(false);
    const [error, setError] = useState('');

    const handleGenerate = useCallback(async () => {
        if (!commits.trim()) {
            setError('Please enter some commit messages.');
            return;
        }
        setIsLoading(true);
        setError('');
        setState(s => ({ ...s, changelog: '' }));
        try {
            const stream = generateChangelogStream(commits);
            let fullResponse = '';
            for await (const chunk of stream) {
                fullResponse += chunk;
                setState(s => ({ ...s, changelog: fullResponse }));
            }
        } catch (err) {
            const errorMessage = err instanceof Error ? err.message : 'An unknown error
            occurred.';
            setError(`Failed to generate changelog: ${errorMessage}`);
        } finally {
            setIsLoading(false);
        }
    }, [commits, setState]);


    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8">
            <FeatureHeader feature={feature} onSaveState={saveState} />
            <div className="flex-grow grid grid-cols-1 lg:grid-cols-2 gap-6 h-full overflow-
            hidden">
                <div className="flex flex-col h-full">
                    <label htmlFor="commit-input" className="text-sm font-medium text-slate-400
                    mb-2">Commit Messages (one per line)</label>
                    <textarea
                        id="commit-input"
                        value={commits}
                        onChange={(e) => setState(s => ({ ...s, commits: e.target.value }))}
                        className="flex-grow p-4 bg-slate-900 border border-slate-700 rounded-md resize-
                        none font-mono text-sm"
                    />
                     <button
                        onClick={handleGenerate}
                        disabled={isLoading}
                        className="mt-4 w-full flex items-center justify-center px-6 py-3 bg-cyan-500
                        text-slate-900 font-bold rounded-md hover:bg-cyan-400 disabled:bg-slate-600"
                    >
                        {isLoading ? <LoadingSpinner /> : 'Generate Changelog'}
```

```
                    </button>
                </div>
                <div className="flex flex-col h-full">
                    <label className="text-sm font-medium text-slate-400 mb-2">Generated
                    Changelog.md</label>
                    <div className="relative flex-grow p-4 bg-slate-800/50 border border-
                    slate-700/50 rounded-md overflow-y-auto">
                        {isLoading && !changelog && <div className="flex items-center justify-center
                        h-full"><LoadingSpinner /></div>}
                        {error && <p className="text-red-400">{error}</p>}
                        {changelog && <MarkdownRenderer content={changelog} />}
                        {!isLoading && !changelog && !error && <div className="text-slate-500 h-full
                        flex items-center justify-center">Changelog will appear here.</div>}
                    </div>
                </div>
            </div>
        </div>
    );
};


// ===== CodeDiffGhost.tsx =====


import React, { useState, useEffect } from 'react';
import { EyeIcon } from '../icons/FeatureIcons.tsx';

const oldCode = `function UserProfile({ user }) {
  return (
    <div className="profile">
      <h1>{user.name}</h1>
      <p>{user.email}</p>
    </div>
  );
}`;

const newCode = `function UserProfile({ user }) {
  const { name, email, avatar } = user;
  return (
    <div className="profile-card">
      <img src={avatar} alt={name} />
      <h2>{name}</h2>
      <a href={\`mailto:\${email}\`}>{email}</a>
    </div>
  );
}`;

export const CodeDiffGhost: React.FC = () => {
    const [typedCode, setTypedCode] = useState('');
    const [isRunning, setIsRunning] = useState(false);

    useEffect(() => {
        if (isRunning) {
            setTypedCode('');
            const intervalId = window.setInterval(() => {
                setTypedCode(prev => {
                    if (prev.length < newCode.length) {
                        return newCode.substring(0, prev.length + 1);
                    }
                    window.clearInterval(intervalId);
                    setIsRunning(false);
                    return newCode;
```

```
                });
            }, 20);
            return () => window.clearInterval(intervalId);
        }
    }, [isRunning]);

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8">
            <header className="mb-6">
                <h1 className="text-3xl font-bold text-slate-100 flex items-center">
                    <EyeIcon />
                    <span className="ml-3">Code Diff Ghost (Simulation)</span>
                </h1>
                <p className="text-slate-400 mt-1">A simulation showing code changes with a
                "ghost typing" effect.</p>
            </header>
            <div className="flex justify-center mb-4">
                <button
                    onClick={() => setIsRunning(true)}
                    disabled={isRunning}
                    className="px-6 py-2 bg-cyan-500 text-slate-900 font-bold rounded-md
                    disabled:bg-slate-600"
                >
                    {isRunning ? 'Visualizing...' : 'Show Changes'}
                </button>
            </div>
            <div className="flex-grow grid grid-cols-1 lg:grid-cols-2 gap-6 h-full overflow-
            hidden font-mono text-sm">
                <div className="flex flex-col h-full">
                    <label className="text-sm font-medium text-slate-400 mb-2">Before</label>
                    <pre className="flex-grow p-4 bg-slate-900 border border-slate-700 rounded-md
                    text-red-300 whitespace-pre-wrap">
                        {oldCode}
                    </pre>
                </div>
                 <div className="flex flex-col h-full">
                    <label className="text-sm font-medium text-slate-400 mb-2">After</label>
                     <pre className="flex-grow p-4 bg-slate-900 border border-slate-700 rounded-md
                     text-green-300 whitespace-pre-wrap">
                        {typedCode}<span className="animate-pulse">|</span>
                    </pre>
                </div>
            </div>
        </div>
    );
};

// ===== CodeFormatter.tsx =====



import React, { useState, useCallback } from 'react';
import { formatCodeStream } from '../../services/geminiService.ts';
import { LoadingSpinner } from '../shared/LoadingSpinner.tsx';
import { marked } from 'marked';
import { useFeatureState } from '../../hooks/useFeatureState.ts';
import { FeatureHeader } from '../shared/FeatureHeader.tsx';
import type { Feature } from '../../types.ts';

const exampleCode = `const MyComponent = (props) => {
  const {name, items}=props
    if(!items || items.length === 0){
```

```
  return <p>No items found for {name}</p>;
    }
  return <ul>{items.map(item=> <li key={item.id}>{item.name}</li>)}</ul>
}`;

export const CodeFormatter: React.FC<{ feature: Feature }> = ({ feature }) => {
    const [state, setState, saveState] = useFeatureState('codeFormatter', {
        inputCode: exampleCode,
        formattedCode: '',
    });
    const { inputCode, formattedCode } = state;

    const [isLoading, setIsLoading] = useState<boolean>(false);
    const [error, setError] = useState<string>('');

    const handleFormat = useCallback(async () => {
        if (!inputCode.trim()) {
            setError('Please enter some code to format.');
            return;
        }
        setIsLoading(true);
        setError('');
        setState(s => ({ ...s, formattedCode: '' }));
        try {
            const stream = formatCodeStream(inputCode);
            let fullResponse = '';
            for await (const chunk of stream) {
                fullResponse += chunk;
                setState(s => ({ ...s, formattedCode: fullResponse }));
            }
        } catch (err) {
            const errorMessage = err instanceof Error ? err.message : 'An unknown error
            occurred.';
            setError(`Failed to format code: ${errorMessage}`);
        } finally {
            setIsLoading(false);
        }
    }, [inputCode, setState]);

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8">
            <FeatureHeader feature={feature} onSaveState={saveState} />
            <div className="flex-grow flex flex-col min-h-0">
                <div className="grid grid-cols-1 lg:grid-cols-2 gap-6 flex-grow min-h-0">
                    <div className="flex flex-col h-full">
                        <label htmlFor="code-input" className="text-sm font-medium text-slate-400
                        mb-2">Input</label>
                        <textarea
                            id="code-input"
                            value={inputCode}
                            onChange={(e) => setState(s => ({ ...s, inputCode: e.target.value }))}
                            placeholder="Paste your unformatted code here..."
                            className="flex-grow p-4 bg-slate-900 border border-slate-700 rounded-md resize-
                            none font-mono text-sm text-cyan-300 focus:ring-2 focus:ring-cyan-500
                            focus:outline-none"
                        />
                    </div>
                    <div className="flex flex-col h-full">
                        <label className="text-sm font-medium text-slate-400 mb-2">Output</label>
                        <div className="flex-grow p-1 bg-slate-800/50 border border-slate-700/50
                        rounded-md overflow-y-auto">
                           {isLoading && (
```

```
                                <div className="flex items-center justify-center h-full">
                                    <LoadingSpinner />
                                </div>
                            )}
                            {error && <p className="p-4 text-red-400">{error}</p>}
                            {formattedCode && !isLoading && (
                                <div
                                    className="prose prose-sm prose-invert max-w-none prose-pre:bg-transparent
                                    prose-pre:p-4 prose-pre:m-0 prose-code:text-cyan-300"
                                    dangerouslySetInnerHTML={{ __html: marked.parse(formattedCode) as string }}
                                />
                            )}
                            {!isLoading && !formattedCode && !error && (
                                <div className="text-slate-500 h-full flex items-center justify-center">
                                    Formatted code will appear here.
                                </div>
                            )}
                        </div>
                    </div>
                </div>
                 <button
                    onClick={handleFormat}
                    disabled={isLoading}
                    className="mt-4 w-full max-w-sm mx-auto flex items-center justify-center px-6
                    py-3 bg-cyan-500 text-slate-900 font-bold rounded-md hover:bg-cyan-400
                    transition-colors disabled:bg-slate-600 disabled:cursor-not-allowed"
                >
                    {isLoading ? <LoadingSpinner /> : 'Format Code'}
                </button>
            </div>
        </div>
    );
};

// ===== CodeReviewBot.tsx =====



import React, { useState, useCallback } from 'react';
import { reviewCodeStream } from '../../services/geminiService.ts';
import { LoadingSpinner } from '../shared/LoadingSpinner.tsx';
import { marked } from 'marked';
import { useFeatureState } from '../../hooks/useFeatureState.ts';
import { FeatureHeader } from '../shared/FeatureHeader.tsx';
import type { Feature } from '../../types.ts';

const exampleCode = `function UserList(users) {
  if (users.length = 0) {
    return "no users";
  } else {
    return (
      users.map(u => {
        return <li>{u.name}</li>
      })
    )
  }
}`;

export const CodeReviewBot: React.FC<{ feature: Feature }> = ({ feature }) => {
    const [state, setState, saveState] = useFeatureState('codeReviewBot', { code:
    exampleCode, review: '' });
    const { code, review } = state;
```

```
const [isLoading, setIsLoading] = useState<boolean>(false);
const [error, setError] = useState<string>('');

const handleGenerate = useCallback(async () => {
    if (!code.trim()) {
        setError('Please enter some code to review.');
        return;
    }
    setIsLoading(true);
    setError('');
    setState(s => ({ ...s, review: '' }));
    try {
        const stream = reviewCodeStream(code);
        let fullResponse = '';
        for await (const chunk of stream) {
            fullResponse += chunk;
            setState(s => ({ ...s, review: fullResponse }));
        }
    } catch (err) {
        const errorMessage = err instanceof Error ? err.message : 'An unknown error
        occurred.';
        setError(`Failed to get review: ${errorMessage}`);
    } finally {
        setIsLoading(false);
    }
}, [code, setState]);

return (
    <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8">
        <FeatureHeader feature={feature} onSaveState={saveState} />
        <div className="flex-grow grid grid-cols-1 lg:grid-cols-2 gap-6 h-full overflow-
        hidden">
            <div className="flex flex-col h-full">
                <label htmlFor="code-input" className="text-sm font-medium text-slate-400
                mb-2">Code to Review</label>
                <textarea
                    id="code-input"
                    value={code}
                    onChange={(e) => setState(s => ({ ...s, code: e.target.value }))}
                    placeholder="Paste your code here..."
                    className="flex-grow p-4 bg-slate-900 border border-slate-700 rounded-md resize-
                    none font-mono text-sm text-cyan-300"
                />
                <button
                    onClick={handleGenerate}
                    disabled={isLoading}
                    className="mt-4 w-full flex items-center justify-center px-6 py-3 bg-cyan-500
                    text-slate-900 font-bold rounded-md hover:bg-cyan-400 disabled:bg-slate-600"
                >
                    {isLoading ? <LoadingSpinner /> : 'Request Review'}
                </button>
            </div>
            <div className="flex flex-col h-full">
                <label className="text-sm font-medium text-slate-400 mb-2">AI Feedback</label>
                <div className="flex-grow p-4 bg-slate-800/50 border border-slate-700/50
                rounded-md overflow-y-auto">
                    {isLoading && <div className="flex items-center justify-center
                    h-full"><LoadingSpinner /></div>}
                    {error && <p className="text-red-400">{error}</p>}
                    {review && !isLoading && (
                        <div
                            className="prose prose-sm prose-invert max-w-none"
```

```
                                dangerouslySetInnerHTML={{ __html: marked.parse(review) as string }}
                            />
                        )}
                        {!isLoading && !review && !error && <div className="text-slate-500 h-full flex
                        items-center justify-center">Review will appear here.</div>}
                    </div>
                </div>
            </div>
        </div>
    );
};

// ===== CodeSpellChecker.tsx =====

import React, { useState, useMemo } from 'react';
import { BeakerIcon } from '../icons/FeatureIcons.tsx';

const typos = ['funtion', 'consle', 'varable', 'docment', 'componnet'];
const typoRegex = new RegExp(`\\b(${typos.join('|')})\\b`, 'gi');

const HighlightedText: React.FC<{ text: string }> = ({ text }) => {
    const parts = useMemo(() => {
        return text.split(typoRegex).map((part, i) => {
            if (typoRegex.test(part)) {
                return <span key={i} className="underline decoration-red-500 decoration-
                wavy">{part}</span>;
            }
            return part;
        });
    }, [text]);

    return <>{parts}</>;
};

export const CodeSpellChecker: React.FC = () => {
    const [code, setCode] = useState('funtion myFunction() {\n  consle.log("Hello
    World");\n  const myVarable = docment.getElementById("root");\n  // This is a
    React componnet\n}');

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8">
            <header className="mb-6">
                <h1 className="text-3xl font-bold text-slate-100 flex items-center">
                    <BeakerIcon />
                    <span className="ml-3">Code Spell Checker (Simulation)</span>
                </h1>
                <p className="text-slate-400 mt-1">A simulation of a spell checker that finds
                common typos in code.</p>
            </header>
            <div className="relative flex-grow font-mono text-sm bg-slate-900 border border-
            slate-700 rounded-lg p-4 overflow-auto">
                <textarea
                    value={code}
                    onChange={(e) => setCode(e.target.value)}
                    className="absolute inset-0 w-full h-full p-4 bg-transparent text-transparent
                    caret-cyan-400 resize-none z-10"
                    spellCheck="false"
                />
                <pre className="absolute inset-0 w-full h-full p-4 pointer-events-none
                whitespace-pre-wrap" aria-hidden="true">
                    <HighlightedText text={code} />
                </pre>
```

```
            </div>
        </div>
    );
};

// ===== ColorPaletteGenerator.tsx =====



import React, { useState, useCallback } from 'react';
import { HexColorPicker } from 'react-colorful';
import { generateColorPalette } from '../../services/geminiService.ts';
import { LoadingSpinner } from '../shared/LoadingSpinner.tsx';
import { useFeatureState } from '../../hooks/useFeatureState.ts';
import { FeatureHeader } from '../shared/FeatureHeader.tsx';
import type { Feature } from '../../types.ts';

export const ColorPaletteGenerator: React.FC<{ feature: Feature }> = ({ feature
}) => {
    const [state, setState, saveState] = useFeatureState('colorPaletteGenerator', {
        baseColor: "#06b6d4",
        palette: ['#06b6d4', '#0891b2', '#0e7490', '#155e75', '#164e63', '#083344'],
    });
    const { baseColor, palette } = state;

    const [isLoading, setIsLoading] = useState<boolean>(false);
    const [error, setError] = useState<string>('');

    const handleGenerate = useCallback(async () => {
        setIsLoading(true);
        setError('');
        try {
            const result = await generateColorPalette(baseColor);
            setState(s => ({...s, palette: result.colors}));
        } catch (err) {
            const errorMessage = err instanceof Error ? err.message : 'An unknown error
            occurred.';
            setError(`Failed to generate palette: ${errorMessage}`);
        } finally {
            setIsLoading(false);
        }
    }, [baseColor, setState]);

    const handleCopy = (color: string) => {
        navigator.clipboard.writeText(color);
    };

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8">
            <FeatureHeader feature={feature} onSaveState={saveState} />
            <div className="flex-grow flex flex-col md:flex-row items-center justify-center
            gap-8">
                <div className="flex flex-col items-center gap-4">
                     <HexColorPicker color={baseColor} onChange={c => setState(s => ({...s,
                     baseColor: c}))} className="!w-64 !h-64"/>
                     <div className="p-2 bg-slate-800 rounded-md font-mono text-lg" style={{border:
                     `1px solid ${baseColor}`}}>{baseColor}</div>
                      <button
                        onClick={handleGenerate}
                        disabled={isLoading}
                        className="w-full flex items-center justify-center px-6 py-3 bg-cyan-500 text-
                        slate-900 font-bold rounded-md hover:bg-cyan-400 transition-colors disabled:bg-
```

```
                        slate-600 disabled:cursor-not-allowed"
                    >
                        {isLoading ? <LoadingSpinner /> : 'Generate Palette'}
                    </button>
                    {error && <p className="text-red-400 text-sm mt-2">{error}</p>}
                </div>
                <div className="flex flex-col gap-2 w-full max-w-sm">
                    <label className="text-sm font-medium text-slate-400 mb-2">Generated
                    Palette:</label>
                    {isLoading ? (
                         <div className="flex items-center justify-center h-48"><LoadingSpinner /></div>
                    ) : (
                        palette.map((color) => (
                            <div key={color} className="group flex items-center justify-between p-4 rounded-
                            md" style={{ backgroundColor: color }}>
                                <span className="font-mono font-bold text-black/70 mix-blend-
                                overlay">{color}</span>
                                <button
                                    onClick={() => handleCopy(color)}
                                    className="opacity-0 group-hover:opacity-100 transition-opacity bg-white/30
                                    hover:bg-white/50 px-3 py-1 rounded text-xs text-black font-semibold">
                                    Copy
                                </button>
                            </div>
                        ))
                    )}
                </div>
            </div>
        </div>
    );
};


// ===== CommandPaletteTrigger.tsx =====


import React from 'react';
import { CommandLineIcon } from '../icons/FeatureIcons';

export const CommandPaletteTrigger: React.FC = () => {
    return (
        <div className="flex flex-col items-center justify-center h-full p-8 text-center
        text-slate-400">
            <div className="text-6xl mb-4" aria-hidden="true">
                <CommandLineIcon />
            </div>
            <h1 className="text-3xl font-bold text-slate-200 mb-2">
                Command Palette
            </h1>
            <p className="text-lg mb-4 max-w-md">
                The Command Palette provides quick access to all features and commands.
            </p>
            <div className="bg-slate-800 text-cyan-300 border border-slate-700 rounded-lg
            px-6 py-4">
                <p className="font-semibold">Press <kbd className="mx-1 font-sans px-2 py-1.5
                text-xs font-semibold text-gray-800 bg-gray-100 border border-gray-200 rounded-
                lg">Ctrl</kbd> + <kbd className="mx-1 font-sans px-2 py-1.5 text-xs font-
                semibold text-gray-800 bg-gray-100 border border-gray-200 rounded-lg">K</kbd> to
                open.</p>
            </div>
        </div>
    );
```

```
};


// ===== CronJobBuilder.tsx =====


import React, { useState, useMemo } from 'react';
import { CommandLineIcon } from '../icons/FeatureIcons';

const CronPartSelector: React.FC<{ label: string, value: string, onChange:
(value: string) => void, options: (string|number)[] }> = ({ label, value,
onChange, options }) => {
    return (
        <div>
            <label className="block text-sm font-medium text-slate-400">{label}</label>
            <select value={value} onChange={e => onChange(e.target.value)} className="w-full
            mt-1 px-3 py-2 rounded-md bg-slate-800 border border-slate-700">
                <option value="*">* (every)</option>
                {options.map(o => <option key={o} value={o}>{o}</option>)}
            </select>
        </div>
    );
};

export const CronJobBuilder: React.FC = () => {
    const [minute, setMinute] = useState('0');
    const [hour, setHour] = useState('0');
    const [dayOfMonth, setDayOfMonth] = useState('*');
    const [month, setMonth] = useState('*');
    const [dayOfWeek, setDayOfWeek] = useState('*');

    const cronExpression = useMemo(() => {
        return `${minute} ${hour} ${dayOfMonth} ${month} ${dayOfWeek}`;
    }, [minute, hour, dayOfMonth, month, dayOfWeek]);

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8">
            <header className="mb-6">
                <h1 className="text-3xl font-bold text-slate-100 flex items-center">
                    <CommandLineIcon />
                    <span className="ml-3">Cron Job Builder</span>
                </h1>
                <p className="text-slate-400 mt-1">Visually construct a cron expression for
                scheduling tasks.</p>
            </header>
            <div className="grid grid-cols-2 md:grid-cols-5 gap-4 mb-6">
                <CronPartSelector label="Minute" value={minute} onChange={setMinute}
                options={Array.from({length: 60}, (_, i) => i)} />
                <CronPartSelector label="Hour" value={hour} onChange={setHour}
                options={Array.from({length: 24}, (_, i) => i)} />
                <CronPartSelector label="Day (Month)" value={dayOfMonth}
                onChange={setDayOfMonth} options={Array.from({length: 31}, (_, i) => i + 1)} />
                <CronPartSelector label="Month" value={month} onChange={setMonth}
                options={Array.from({length: 12}, (_, i) => i + 1)} />
                <CronPartSelector label="Day (Week)" value={dayOfWeek} onChange={setDayOfWeek}
                options={Array.from({length: 7}, (_, i) => i)} />
            </div>
            <div className="bg-slate-900 p-4 rounded-lg text-center">
                <p className="text-slate-400 text-sm">Generated Expression</p>
                <p className="font-mono text-cyan-400 text-2xl mt-1">{cronExpression}</p>
                 <button onClick={() => navigator.clipboard.writeText(cronExpression)}
                 className="mt-4 px-3 py-1 bg-slate-700 hover:bg-slate-600 rounded-md text-
```

```
                xs">Copy</button>
            </div>
        </div>
    );
};


// ===== CssGridEditor.tsx =====


import React, { useState, useMemo, useCallback } from 'react';
import { CodeBracketSquareIcon } from '../icons/FeatureIcons.tsx';
import { useFeatureState } from '../../hooks/useFeatureState.ts';
import { FeatureHeader } from '../shared/FeatureHeader.tsx';
import type { Feature } from '../../types.ts';
import { generateCssGridStream } from '../../services/geminiService.ts';
import { LoadingSpinner } from '../shared/LoadingSpinner.tsx';
import { MarkdownRenderer } from '../shared/MarkdownRenderer.tsx';

export const CssGridEditor: React.FC<{ feature: Feature }> = ({ feature }) => {
    const [state, setState, saveState] = useFeatureState('cssGridEditor', {
        aiPrompt: 'a simple 3x2 grid with a 1rem gap',
        cssCode: `.grid-container {
  display: grid;
  grid-template-columns: repeat(3, 1fr);
  grid-template-rows: repeat(2, 1fr);
  gap: 1rem;
}`,
        rows: 2,
        cols: 3,
    });
    const { aiPrompt, cssCode, rows, cols } = state;
    const [isLoading, setIsLoading] = useState(false);

    const handleGenerate = useCallback(async () => {
        setIsLoading(true);
        try {
            const stream = generateCssGridStream(aiPrompt);
            let fullResponse = '';
            for await (const chunk of stream) {
                fullResponse += chunk;
                const cleanedCode = fullResponse.replace(/^```(?:css\n)?/, '').replace(/```$/,
                '');
                setState(s => ({...s, cssCode: cleanedCode }));
            }
        } catch (e) {
            console.error(e);
        } finally {
            setIsLoading(false);
        }
    }, [aiPrompt, setState]);

    // Simple parser to make the visual preview work with the generated code
    useMemo(() => {
        const rowsMatch = cssCode.match(/grid-template-rows:\s*repeat\((\d+)/);
        const colsMatch = cssCode.match(/grid-template-columns:\s*repeat\((\d+)/);
        const newRows = rowsMatch ? parseInt(rowsMatch[1]) : 2;
        const newCols = colsMatch ? parseInt(colsMatch[1]) : 3;
        setState(s => ({...s, rows: newRows, cols: newCols }));
    }, [cssCode, setState]);
    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8">
```

```
            <FeatureHeader feature={feature} onSaveState={saveState} />
            <div className="flex-grow grid grid-cols-1 lg:grid-cols-3 gap-6 min-h-0">
                <div className="lg:col-span-1 flex flex-col gap-4 bg-slate-800/50 p-6 rounded-
                lg">
                    <h3 className="text-xl font-bold">AI Prompt</h3>
                     <textarea
                        value={aiPrompt}
                        onChange={e => setState(s => ({...s, aiPrompt: e.target.value}))}
                        placeholder="Describe your grid layout..."
                        className="w-full h-24 p-2 bg-slate-800 border border-slate-700 rounded-md
                        resize-none text-sm"
                    />
                    <button onClick={handleGenerate} disabled={isLoading} className="flex items-
                    center justify-center py-2 bg-cyan-500 text-slate-900 font-bold rounded-md">
                        {isLoading ? <LoadingSpinner/> : 'Generate Grid'}
                    </button>
                     <div className="flex-grow mt-4">
                        <label className="block text-sm font-medium text-slate-400 mb-2">Generated
                        CSS</label>
                        <div className="relative h-full">
                           <MarkdownRenderer content={'```css\n' + cssCode + '\n```'} />
                        </div>
                    </div>
                </div>
                <div className="lg:col-span-2 bg-slate-900 rounded-lg p-4">
                    <div className="w-full h-full grid" style={{gridTemplateColumns:
                    `repeat(${cols}, 1fr)`, gridTemplateRows: `repeat(${rows}, 1fr)`, gap: '1rem'}}>
                        {Array.from({ length: rows * cols }).map((_, i) => (
                            <div key={i} className="bg-cyan-500/20 rounded-lg border-2 border-dashed border-
                            cyan-400/50 flex items-center justify-center text-cyan-300">
                                <span className="text-xs opacity-70">{i + 1}</span>
                            </div>
                        ))}
                    </div>
                </div>
            </div>
        </div>
    );
};


// ===== DevNotesStickyPanel.tsx =====

import React, { useState, useCallback, useEffect } from 'react';
import { FileCodeIcon, SparklesIcon } from '../icons/FeatureIcons.tsx';
import { summarizeNotesStream } from '../../services/geminiService.ts';
import { LoadingSpinner } from '../shared/LoadingSpinner.tsx';
import { MarkdownRenderer } from '../shared/MarkdownRenderer.tsx';
import * as storageService from '../../services/storageService.ts';

interface Note {
    id: number;
    text: string;
    x: number;
    y: number;
    color: string;
}

const colors = ['bg-yellow-300', 'bg-green-300', 'bg-blue-300', 'bg-pink-300',
'bg-purple-300'];
const STORAGE_KEY = 'devcore_notes';
export const DevNotesStickyPanel: React.FC = () => {
```

```
const [notes, setNotes] = useState<Note[]>([]);
const [dragging, setDragging] = useState<{ id: number; offsetX: number; offsetY:
number } | null>(null);
const [isSummarizing, setIsSummarizing] = useState(false);
const [summary, setSummary] = useState('');

useEffect(() => {
    const loadNotes = async () => {
        const savedNotes = await storageService.dbGet(STORAGE_KEY);
        if (savedNotes) {
            setNotes(savedNotes);
        }
    };
    loadNotes();
}, []);

useEffect(() => {
    // Debounce saving to IndexedDB
    const handler = setTimeout(() => {
        if (notes.length > 0) {
             storageService.dbSet(STORAGE_KEY, notes);
        }
    }, 500);
    return () => clearTimeout(handler);
}, [notes]);

const handleSummarize = useCallback(async () => {
    if (notes.length === 0) return;
    setIsSummarizing(true);
    setSummary('');
    try {
        const allNotesText = notes.map((n: Note) => `- ${n.text}`).join('\n');
        const stream = summarizeNotesStream(allNotesText);
        let fullResponse = '';
        for await (const chunk of stream) {
            fullResponse += chunk;
            setSummary(fullResponse);
        }
    } catch (error) {
        console.error(error);
        setSummary('Sorry, an error occurred while summarizing.');
    } finally {
        setIsSummarizing(false);
    }
}, [notes]);


const addNote = () => {
    const newNote: Note = {
        id: Date.now(),
        text: 'New note...',
        x: 50 + (notes.length % 10) * 20,
        y: 50 + (notes.length % 10) * 20,
        color: colors[notes.length % colors.length],
    };
    setNotes([...notes, newNote]);
};

const updateText = (id: number, text: string) => {
    setNotes(notes.map((n: Note) => n.id === id ? { ...n, text } : n));
};
const deleteNote = (id: number) => {
```

```
    setNotes(notes.filter((n: Note) => n.id !== id));
};

const onMouseDown = (e: React.MouseEvent<HTMLDivElement>, id: number) => {
    if((e.target as HTMLElement).tagName === 'TEXTAREA' || (e.target as
    HTMLElement).tagName === 'BUTTON') return;
    const noteElement = e.currentTarget;
    const rect = noteElement.getBoundingClientRect();
    setDragging({ id, offsetX: e.clientX - rect.left, offsetY: e.clientY - rect.top
    });
};

const onMouseMove = (e: React.MouseEvent<HTMLDivElement>) => {
    if (!dragging) return;
    const boardRect = e.currentTarget.getBoundingClientRect();
    setNotes(
        notes.map((n: Note) =>
            n.id === dragging.id
                ? { ...n, x: e.clientX - dragging.offsetX - boardRect.left, y: e.clientY -
                dragging.offsetY - boardRect.top }
                : n
        )
    );
};

const onMouseUp = () => setDragging(null);

return (
    <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8">
        <header className="mb-6 flex justify-between items-center">
             <div>
                <h1 className="text-3xl font-bold text-slate-100 flex items-
                center"><FileCodeIcon /><span className="ml-3">Dev Notes Sticky
                Panel</span></h1>
                <p className="text-slate-400 mt-1">A place for your thoughts, todos, and random
                ideas.</p>
            </div>
            <div className="flex gap-2">
                <button onClick={handleSummarize} disabled={isSummarizing || notes.length === 0}
                className="flex items-center gap-2 px-4 py-2 bg-purple-500 text-slate-900 font-
                bold rounded-md disabled:bg-slate-600">
                    <SparklesIcon/> {isSummarizing ? 'Summarizing...' : 'AI Summarize'}
                </button>
                <button onClick={addNote} className="px-6 py-2 bg-cyan-500 text-slate-900 font-
                bold rounded-md">Add Note</button>
            </div>
        </header>
        <div
            className="relative flex-grow bg-slate-900/50 border-2 border-dashed border-
            slate-700 rounded-lg overflow-hidden"
            onMouseMove={onMouseMove}
            onMouseUp={onMouseUp}
            onMouseLeave={onMouseUp}
        >
            {notes.map((note: Note) => (
                <div
                    key={note.id}
                    className={`absolute w-48 h-48 p-2 flex flex-col shadow-lg cursor-grab
                    active:cursor-grabbing ${note.color}`}
                    style={{ top: note.y, left: note.x, transform: dragging?.id === note.id ?
                    'scale(1.05) rotate(3deg)' : 'scale(1)' }}
                    onMouseDown={e => onMouseDown(e, note.id)}
```

```
                    >
                         <button onClick={() => deleteNote(note.id)} className="absolute -top-2 -right-2
                         w-6 h-6 rounded-full bg-red-500 text-white font-bold text-xs flex items-center
                         justify-center opacity-0 hover:opacity-100">&times;</button>
                        <textarea
                            value={note.text}
                            onChange={(e) => updateText(note.id, e.target.value)}
                            className="w-full h-full bg-transparent text-black resize-none focus:outline-
                            none font-medium p-1"
                        />
                    </div>
                ))}
            </div>
            {(isSummarizing || summary) && (
                 <div className="fixed inset-0 bg-slate-900/80 backdrop-blur-sm z-50 flex items-
                 center justify-center" onClick={() => setSummary('')}>
                    <div className="w-full max-w-2xl bg-slate-800 border border-slate-700 rounded-lg
                    shadow-2xl p-6" onClick={e => e.stopPropagation()}>
                        <h2 className="text-xl font-bold mb-4">AI Summary of Notes</h2>
                        {isSummarizing && !summary ? <LoadingSpinner /> : <MarkdownRenderer
                        content={summary} />}
                    </div>
                </div>
            )}
        </div>
    );
};


// ===== FontPairingTool.tsx =====

import React, { useState, useEffect } from 'react';
import { EyeIcon } from '../icons/FeatureIcons.tsx';

const popularFonts = [
    'Roboto', 'Open Sans', 'Lato', 'Montserrat', 'Oswald', 'Source Sans Pro',
    'Raleway', 'Poppins', 'Nunito', 'Merriweather'
];

export const FontPairingTool: React.FC = () => {
    const [headingFont, setHeadingFont] = useState('Oswald');
    const [bodyFont, setBodyFont] = useState('Roboto');

    useEffect(() => {
        const fontsToLoad = [headingFont, bodyFont].filter(f => f).join('|');
        if (fontsToLoad) {
            const link = document.createElement('link');
            link.href = `https://fonts.googleapis.com/css?family=${fontsToLoad.replace(/ /g,
            '+')}:400,700&display=swap`;
            link.rel = 'stylesheet';
            document.head.appendChild(link);
            return () => {
                document.head.removeChild(link);
            };
        }
    }, [headingFont, bodyFont]);

    const FontSelector: React.FC<{ label: string, value: string, onChange: (font:
    string) => void }> = ({ label, value, onChange }) => (
        <div>
            <label className="block text-sm font-medium text-slate-400">{label}</label>
            <select value={value} onChange={e => onChange(e.target.value)} className="w-full
```

```
            mt-1 px-3 py-2 rounded-md bg-slate-800 border border-slate-700">
                {popularFonts.map(font => <option key={font} value={font}>{font}</option>)}
            </select>
        </div>
    );

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8">
            <header className="mb-6">
                <h1 className="text-3xl font-bold text-slate-100 flex items-center">
                    <EyeIcon />
                    <span className="ml-3">Font Pairing Tool</span>
                </h1>
                <p className="text-slate-400 mt-1">Preview Google Font combinations for your
                projects.</p>
            </header>
            <div className="flex-grow grid grid-cols-1 lg:grid-cols-3 gap-6 min-h-0">
                <div className="lg:col-span-1 flex flex-col gap-4 bg-slate-800/50 p-6 rounded-
                lg">
                    <h3 className="text-xl font-bold">Controls</h3>
                    <FontSelector label="Heading Font" value={headingFont} onChange={setHeadingFont}
                    />
                    <FontSelector label="Body Font" value={bodyFont} onChange={setBodyFont} />
                </div>
                <div className="lg:col-span-2 bg-slate-900 rounded-lg p-8 overflow-y-auto">
                    <h2 className="text-4xl font-bold mb-4" style={{ fontFamily: headingFont }}>
                        The Quick Brown Fox Jumps Over the Lazy Dog
                    </h2>
                    <p className="text-lg" style={{ fontFamily: bodyFont }}>
                        Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed non risus.
                        Suspendisse lectus tortor, dignissim sit amet, adipiscing nec, ultricies sed,
                        dolor. Cras elementum ultrices diam. Maecenas ligula massa, varius a, semper
                        congue, euismod non, mi. Proin porttitor, orci nec nonummy molestie, enim est
                        eleifend mi, non fermentum diam nisl sit amet erat.
                    </p>
                </div>
            </div>
        </div>
    );
};

// ===== FontPreviewPicker.tsx =====

import React, { useState, useEffect } from 'react';
import { EyeIcon } from '../icons/FeatureIcons.tsx';

const popularFonts = [
    'Roboto', 'Open Sans', 'Lato', 'Montserrat', 'Oswald', 'Source Sans Pro',
    'Raleway', 'Poppins', 'Nunito', 'Merriweather'
];

export const FontPreviewPicker: React.FC = () => {
    const [selectedFont, setSelectedFont] = useState('Roboto');
    const [previewText, setPreviewText] = useState('The quick brown fox jumps over
    the lazy dog.');

    useEffect(() => {
        if (selectedFont) {
            const link = document.createElement('link');
            link.href = `https://fonts.googleapis.com/css?family=${selectedFont.replace(/
            /g, '+')}&display=swap`;
            link.rel = 'stylesheet';
```

```
        document.head.appendChild(link);
        return () => {
            document.head.removeChild(link);
        };
    }
}, [selectedFont]);

const importRule = `@import
url('https://fonts.googleapis.com/css?family=${selectedFont.replace(/ /g,
'+')}&display=swap');`;

return (
    <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8">
        <header className="mb-6">
            <h1 className="text-3xl font-bold text-slate-100 flex items-center">
                <EyeIcon />
                <span className="ml-3">Font Preview & Picker</span>
            </h1>
            <p className="text-slate-400 mt-1">Preview Google Fonts and get the CSS import
            rule.</p>
        </header>
        <div className="flex-grow grid grid-cols-1 lg:grid-cols-3 gap-6 min-h-0">
            <div className="lg:col-span-1 flex flex-col gap-4 bg-slate-800/50 p-6 rounded-
            lg">
                <h3 className="text-xl font-bold">Controls</h3>
                <div>
                    <label htmlFor="font-select" className="block text-sm font-medium text-
                    slate-400">Select Font</label>
                    <select id="font-select" value={selectedFont} onChange={e =>
                    setSelectedFont(e.target.value)} className="w-full mt-1 px-3 py-2 rounded-md bg-
                    slate-800 border border-slate-700">
                        {popularFonts.map(font => <option key={font} value={font}>{font}</option>)}
                    </select>
                </div>
                <div>
                    <label htmlFor="preview-text" className="block text-sm font-medium text-
                    slate-400">Preview Text</label>
                    <textarea id="preview-text" value={previewText} onChange={e =>
                    setPreviewText(e.target.value)} className="w-full mt-1 p-2 rounded-md bg-
                    slate-800 border border-slate-700 h-24 resize-none"></textarea>
                </div>
                <div>
                    <label className="block text-sm font-medium text-slate-400">CSS Import
                    Rule</label>
                    <div className="relative mt-1">
                        <pre className="bg-slate-900 p-2 rounded-md text-cyan-300 text-xs overflow-x-
                        auto">{importRule}</pre>
                        <button onClick={() => navigator.clipboard.writeText(importRule)}
                        className="absolute top-1 right-1 px-2 py-0.5 bg-slate-700 hover:bg-slate-600
                        rounded-md text-xs">Copy</button>
                    </div>
                </div>
            </div>
            <div className="lg:col-span-2 bg-slate-900 rounded-lg p-8 flex items-center
            justify-center">
                <p className="text-4xl" style={{ fontFamily: selectedFont }}>
                    {previewText}
                </p>
            </div>
        </div>
    </div>
);
```

```
};

// ===== JsonTreeNavigator.tsx =====


import React, { useState } from 'react';
import { FileCodeIcon } from '../icons/FeatureIcons.tsx';

interface JsonNodeProps {
    data: any;
    nodeKey: string;
    isRoot?: boolean;
}

const JsonNode: React.FC<JsonNodeProps> = ({ data, nodeKey, isRoot = false }) =>
{
    const [isOpen, setIsOpen] = useState(isRoot);
    const isObject = typeof data === 'object' && data !== null;

    const toggleOpen = () => setIsOpen(!isOpen);

    if (!isObject) {
        return (
            <div className="ml-4 pl-4 border-l border-slate-700">
                <span className="text-purple-400">{nodeKey}: </span>
                <span className={typeof data === 'string' ? 'text-green-400' : 'text-
                orange-400'}>
                    {typeof data === 'string' ? `"${data}"` : String(data)}
                </span>
            </div>
        );
    }

    const entries = Object.entries(data);
    const bracket = Array.isArray(data) ? '[]' : '{}';

    return (
        <div className={`ml-4 ${!isRoot ? 'pl-4 border-l border-slate-700' : ''}`}>
            <button onClick={toggleOpen} className="flex items-center cursor-pointer">
                <span className={`transform transition-transform ${isOpen ? 'rotate-90' :
                'rotate-0'}`}>■</span>
                <span className="ml-1 text-purple-400">{nodeKey}:</span>
                <span className="ml-2 text-slate-500">{bracket[0]}</span>
                {!isOpen && <span className="text-slate-500">...{bracket[1]}</span>}
            </button>
            {isOpen && (
                <div>
                    {entries.map(([key, value]) => (
                        <JsonNode key={key} nodeKey={key} data={value} />
                    ))}
                    <div className="text-slate-500 ml-4">{bracket[1]}</div>
                </div>
            )}
        </div>
    );
};

interface JsonTreeNavigatorProps {
    data?: any;
}

export const JsonTreeNavigator: React.FC<JsonTreeNavigatorProps> = ({ data }) =>
```

```
{
    const [jsonInput, setJsonInput] = useState('{\n  "id": "devcore-001",\n
    "active": true,\n  "features": [\n    "ai-explainer",\n    "api-tester"\n  ],\n
    "config": {\n    "theme": "dark",\n    "version": 1\n  }\n}');
    const [parsedData, setParsedData] = useState<any>(null);
    const [error, setError] = useState('');

    const parseJson = () => {
        try {
            const parsed = JSON.parse(jsonInput);
            setParsedData(parsed);
            setError('');
        } catch (e) {
            if (e instanceof Error) setError(e.message);
            setParsedData(null);
        }
    };

    // If data is passed as a prop, use it directly
    if (data) {
        return (
            <div className="font-mono text-sm">
                <JsonNode data={data} nodeKey="root" isRoot />
            </div>
        );
    }

    // Standalone mode with textarea
    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8">
            <header className="mb-6">
                <h1 className="text-3xl font-bold text-slate-100 flex items-center">
                    <FileCodeIcon />
                    <span className="ml-3">JSON Tree Navigator</span>
                </h1>
                <p className="text-slate-400 mt-1">Paste your JSON data to visualize it as a
                collapsible tree.</p>
            </header>
            <div className="flex-grow grid grid-cols-1 lg:grid-cols-2 gap-6 h-full overflow-
            hidden">
                <div className="flex flex-col h-full">
                    <label htmlFor="json-input" className="text-sm font-medium text-slate-400
                    mb-2">JSON Input</label>
                    <textarea
                        id="json-input"
                        value={jsonInput}
                        onChange={(e) => setJsonInput(e.target.value)}
                        className={`flex-grow p-4 bg-slate-900 border ${error ? 'border-red-500' :
                        'border-slate-700'} rounded-md resize-none font-mono text-sm text-cyan-300
                        focus:ring-2 focus:ring-cyan-500 focus:outline-none`}
                    />
                    {error && <p className="text-red-400 text-xs mt-1">{error}</p>}
                    <button onClick={parseJson} className="mt-4 w-full px-6 py-3 bg-cyan-500 text-
                    slate-900 font-bold rounded-md hover:bg-cyan-400">
                        Render Tree
                    </button>
                </div>
                 <div className="flex flex-col h-full">
                    <label className="text-sm font-medium text-slate-400 mb-2">Tree View</label>
                    <div className="flex-grow p-4 bg-slate-800/50 border border-slate-700/50
                    rounded-md overflow-y-auto">
                        {parsedData ? <JsonTreeNavigator data={parsedData} /> : <div className="text-
```

```
                        slate-500">Click "Render Tree" to view</div>}
                    </div>
                </div>
            </div>
        </div>
    );
};

// ===== LogicFlowBuilder.tsx =====

import React from 'react';
import { MapIcon } from '../icons/FeatureIcons.tsx';

const Node: React.FC<{ title: string, x: number, y: number, color: string }> =
({ title, x, y, color }) => (
    <div
        className={`absolute border rounded-lg px-4 py-2 shadow-lg ${color}`}
        style={{ left: `${x}%`, top: `${y}%`, transform: 'translate(-50%, -50%)' }}
    >
        {title}
    </div>
);

const Edge: React.FC<{ x1: number, y1: number, x2: number, y2: number }> = ({
x1, y1, x2, y2 }) => (
    <line x1={`${x1}%`} y1={`${y1}%`} x2={`${x2}%`} y2={`${y2}%`} stroke="#475569"
    strokeWidth="2" />
);

export const LogicFlowBuilder: React.FC = () => {
    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8">
            <header className="mb-6">
                <h1 className="text-3xl font-bold text-slate-100 flex items-center">
                    <MapIcon />
                    <span className="ml-3">Logic Flow Builder (Simulation)</span>
                </h1>
                <p className="text-slate-400 mt-1">A simulation of a visual tool for building
                application logic.</p>
            </header>
            <div className="flex-grow relative bg-slate-900/50 p-8 rounded-lg border-2
            border-dashed border-slate-700">
                <svg className="absolute inset-0 w-full h-full">
                    <Edge x1={20} y1={20} x2={50} y2={50} />
                    <Edge x1={50} y1={50} x2={80} y2={20} />
                    <Edge x1={50} y1={50} x2={80} y2={80} />
                </svg>
                <Node title="API Request" x={20} y={20} color="border-purple-500 bg-
                purple-500/10" />
                <Node title="Process Data" x={50} y={50} color="border-cyan-500 bg-cyan-500/10"
                />
                <Node title="Update UI" x={80} y={20} color="border-green-500 bg-green-500/10"
                />
                <Node title="Log Error" x={80} y={80} color="border-red-500 bg-red-500/10" />
            </div>
        </div>
    );
};

// ===== MarkdownSlides.tsx =====
import React, { useState, useMemo } from 'react';
import { marked } from 'marked';
```

```
import { PhotoIcon } from '../icons/FeatureIcons.tsx';

const exampleMarkdown = `# Slide 1: Welcome

This is a slide deck generated from Markdown.

- Use standard markdown syntax
- Like lists, headers, and **bold** text.

---

# Slide 2: Features

Navigate using the buttons below.

\`\`\`javascript
console.log("Code blocks work too!");
\`\`\`

---

# Slide 3: The End

Easy to create and present.
`;

export const MarkdownSlides: React.FC = () => {
    const [markdown, setMarkdown] = useState(exampleMarkdown);
    const [currentSlide, setCurrentSlide] = useState(0);

    const slides = useMemo(() => markdown.split(/^-{3,}\s*$/m), [markdown]);

    const goToNext = () => setCurrentSlide(s => Math.min(s + 1, slides.length - 1));
    const goToPrev = () => setCurrentSlide(s => Math.max(s - 1, 0));

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8">
            <header className="mb-6">
                <h1 className="text-3xl font-bold text-slate-100 flex items-center">
                    <PhotoIcon />
                    <span className="ml-3">Markdown to Slides</span>
                </h1>
                <p className="text-slate-400 mt-1">Write markdown, present it as a slideshow.
                Use '---' to separate slides.</p>
            </header>
            <div className="flex-grow grid grid-cols-1 lg:grid-cols-2 gap-6 h-full overflow-
            hidden">
                <div className="flex flex-col h-full">
                     <label htmlFor="md-input" className="text-sm font-medium text-slate-400
                     mb-2">Markdown Editor</label>
                     <textarea
                        id="md-input"
                        value={markdown}
                        onChange={e => setMarkdown(e.target.value)}
                        className="flex-grow p-4 bg-slate-900 border border-slate-700 rounded-md resize-
                        none font-mono text-sm text-cyan-300 focus:ring-2 focus:ring-cyan-500
                        focus:outline-none"
                    />
                </div>
                 <div className="flex flex-col h-full">
                    <label className="text-sm font-medium text-slate-400 mb-2">Presentation
                    View</label>
```

```
                    <div className="relative flex-grow flex flex-col justify-center items-center p-8
                    bg-slate-800/50 border border-slate-700/50 rounded-md overflow-y-auto">
                        <div
                            className="prose prose-lg prose-invert max-w-none w-full"
                            dangerouslySetInnerHTML={{ __html: marked.parse(slides[currentSlide] || '') as
                            string }}
                        />
                         <div className="absolute bottom-4 left-4 right-4 flex justify-between items-
                         center">
                            <button onClick={goToPrev} disabled={currentSlide === 0} className="px-4 py-2
                            bg-slate-700 rounded-md disabled:opacity-50">Prev</button>
                            <span className="text-sm text-slate-400">{currentSlide + 1} /
                            {slides.length}</span>
                            <button onClick={goToNext} disabled={currentSlide === slides.length - 1}
                            className="px-4 py-2 bg-slate-700 rounded-md disabled:opacity-50">Next</button>
                        </div>
                    </div>
                </div>
            </div>
        </div>
    );
};

// ===== MetaTagEditor.tsx =====


import React, { useState, useMemo, useCallback } from 'react';
import { CodeBracketSquareIcon } from '../icons/FeatureIcons.tsx';
import { useFeatureState } from '../../hooks/useFeatureState.ts';
import { FeatureHeader } from '../shared/FeatureHeader.tsx';
import type { Feature } from '../../types.ts';
import { generateMetaTagsStream } from '../../services/geminiService.ts';
import { LoadingSpinner } from '../shared/LoadingSpinner.tsx';
import { MarkdownRenderer } from '../shared/MarkdownRenderer.tsx';


export const MetaTagEditor: React.FC<{ feature: Feature }> = ({ feature }) => {
    const [state, setState, saveState] = useFeatureState('metaTagEditor', {
        aiPrompt: 'A personal portfolio website for a frontend developer named Jane
        Doe.',
        generatedHtml: `<!-- Primary Meta Tags -->
<title>Jane Doe - Frontend Developer</title>
<meta name="title" content="Jane Doe - Frontend Developer" />
<meta name="description" content="Welcome to the portfolio of Jane Doe, a
passionate frontend developer specializing in React and modern web
technologies." />`
    });
    const { aiPrompt, generatedHtml } = state;
    const [isLoading, setIsLoading] = useState(false);

    const handleGenerate = useCallback(async () => {
        if (!aiPrompt.trim()) return;
        setIsLoading(true);
        try {
            const stream = generateMetaTagsStream(aiPrompt);
            let fullResponse = '';
            for await (const chunk of stream) {
                fullResponse += chunk;
                const cleanedHtml = fullResponse.replace(/^```(?:html\n)?/, '').replace(/```$/,
                '');
                setState(s => ({...s, generatedHtml: cleanedHtml}));
            }
```

```
        } catch(e) {
            console.error(e);
        } finally {
            setIsLoading(false);
        }
    }, [aiPrompt, setState]);

    const handleCopy = () => {
        navigator.clipboard.writeText(generatedHtml);
    };

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8">
            <FeatureHeader feature={feature} onSaveState={saveState} />
            <div className="flex-grow grid grid-cols-1 lg:grid-cols-2 gap-6 min-h-0">
                <div className="flex flex-col gap-4 bg-slate-800/50 p-6 rounded-lg">
                    <h3 className="text-xl font-bold">AI Prompt</h3>
                    <div>
                        <label htmlFor="title" className="block text-sm font-medium text-
                        slate-400">Describe your webpage</label>
                        <textarea
                         value={aiPrompt}
                         onChange={e => setState(s => ({...s, aiPrompt: e.target.value}))}
                         className="w-full mt-1 h-32 p-2 rounded-md bg-slate-800 border border-slate-700
                         resize-none"
                        />
                    </div>
                     <button onClick={handleGenerate} disabled={isLoading} className="flex items-
                     center justify-center py-2 bg-cyan-500 text-slate-900 font-bold rounded-md">
                        {isLoading ? <LoadingSpinner/> : 'Generate Tags'}
                    </button>
                </div>
                <div className="flex flex-col">
                     <label className="text-sm font-medium text-slate-400 mb-2">Generated
                     HTML</label>
                     <div className="relative flex-grow">
                        <div className="w-full h-full bg-slate-900 p-1 rounded-md text-cyan-300 text-sm
                        overflow-auto">
                            <MarkdownRenderer content={'```html\n' + generatedHtml + '\n```'} />
                        </div>
                        <button onClick={handleCopy} className="absolute top-2 right-2 px-2 py-1 bg-
                        slate-700 hover:bg-slate-600 rounded-md text-xs">Copy</button>
                    </div>
                </div>
            </div>
        </div>
    );
};


// ===== NetworkVisualizer.tsx =====

import React, { useState, useEffect, useMemo } from 'react';
import { ChartBarIcon } from '../icons/FeatureIcons.tsx';

type SortKey = 'name' | 'type' | 'size' | 'duration';
type SortDirection = 'asc' | 'desc';

export const NetworkVisualizer: React.FC = () => {
    const [requests, setRequests] = useState<PerformanceResourceTiming[]>([]);
    const [sortKey, setSortKey] = useState<SortKey>('duration');
    const [sortDirection, setSortDirection] = useState<SortDirection>('desc');
```

```
useEffect(() => {
    const entries = performance.getEntriesByType("resource") as
    PerformanceResourceTiming[];
    setRequests(entries);
}, []);

const sortedRequests = useMemo(() => {
    return [...requests].sort((a, b) => {
        let valA, valB;
        if(sortKey === 'size') {
             valA = a.transferSize;
             valB = b.transferSize;
        } else {
             valA = a[sortKey];
             valB = b[sortKey];
        }

        if (valA < valB) return sortDirection === 'asc' ? -1 : 1;
        if (valA > valB) return sortDirection === 'asc' ? 1 : -1;
        return 0;
    });
}, [requests, sortKey, sortDirection]);

const handleSort = (key: SortKey) => {
    if (sortKey === key) {
        setSortDirection(prev => prev === 'asc' ? 'desc' : 'asc');
    } else {
        setSortKey(key);
        setSortDirection('desc');
    }
};

const formatBytes = (bytes: number) => {
    if (bytes === 0) return '0 B';
    const k = 1024;
    const sizes = ['B', 'KB', 'MB'];
    const i = Math.floor(Math.log(bytes) / Math.log(k));
    return parseFloat((bytes / Math.pow(k, i)).toFixed(1)) + ' ' + sizes[i];
};

const SortableHeader: React.FC<{
    skey: SortKey;
    label: string;
}> = ({ skey, label }) => (
    <th onClick={() => handleSort(skey)} className="p-2 text-left cursor-pointer
    hover:bg-slate-800">
        {label} {sortKey === skey && (sortDirection === 'asc' ? '▲' : '▼')}
    </th>
);

return (
    <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8">
        <header className="mb-6">
            <h1 className="text-3xl font-bold text-slate-100 flex items-center">
                <ChartBarIcon />
                <span className="ml-3">Network Visualizer</span>
            </h1>
            <p className="text-slate-400 mt-1">Inspect network resources loaded by this
            page.</p>
        </header>
        <div className="flex-grow overflow-y-auto bg-slate-900 rounded-lg">
            <table className="w-full text-sm text-left">
```

```
                    <thead className="sticky top-0 bg-slate-900">
                        <tr>
                            <SortableHeader skey="name" label="Name" />
                            <SortableHeader skey="type" label="Type" />
                            <SortableHeader skey="size" label="Size" />
                            <SortableHeader skey="duration" label="Time" />
                        </tr>
                    </thead>
                    <tbody>
                        {sortedRequests.map((req, i) => (
                            <tr key={i} className="border-b border-slate-800 hover:bg-slate-800/50">
                                <td className="p-2 text-cyan-400 truncate max-w-sm"
                                title={req.name}>{req.name.split('/').pop()}</td>
                                <td className="p-2">{req.initiatorType}</td>
                                <td className="p-2">{formatBytes(req.transferSize)}</td>
                                <td className="p-2">{req.duration.toFixed(0)}ms</td>
                            </tr>
                        ))}
                    </tbody>
                </table>
            </div>
        </div>
    );
};

// ===== Placeholder.tsx =====


import React from 'react';
import type { Feature } from '../types.ts';

export const Placeholder: React.FC<{ feature?: Feature }> = ({ feature }) => {
    return (
        <div className="flex flex-col items-center justify-center h-full p-8 text-center
        text-slate-400">
            <div className="text-6xl mb-4" aria-hidden="true">
                {feature?.icon}
            </div>
            <h1 className="text-3xl font-bold text-slate-200 mb-2">
                {feature?.name || 'Feature'}
            </h1>
            <p className="text-lg mb-4 max-w-md">
                {feature?.description || 'Feature description.'}
            </p>
            <div className="bg-yellow-900/50 text-yellow-300 border border-yellow-800/80
            rounded-lg px-6 py-3">
                <p className="font-semibold">Under Construction</p>
                <p className="text-sm">This feature is not yet available.</p>
            </div>
        </div>
    );
};

// ===== PrGenerator.tsx =====

import React, { useState } from 'react';
import { GitBranchIcon } from '../icons/FeatureIcons.tsx';

export const PrGenerator: React.FC = () => {
    const [title, setTitle] = useState('');
    const [description, setDescription] = useState('');
    const [fromBranch, setFromBranch] = useState('feature/new-login');
```

```
    const [toBranch, setToBranch] = useState('main');
    const [isSubmitted, setIsSubmitted] = useState(false);

    const handleSubmit = (e: React.FormEvent) => {
        e.preventDefault();
        setIsSubmitted(true);
        setTimeout(() => setIsSubmitted(false), 3000);
    };

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8">
            <header className="mb-6">
                <h1 className="text-3xl font-bold text-slate-100 flex items-center">
                    <GitBranchIcon />
                    <span className="ml-3">Pull Request Generator (Simulation)</span>
                </h1>
                <p className="text-slate-400 mt-1">A simple form to draft a pull request.</p>
            </header>
            {isSubmitted ? (
                 <div className="flex-grow flex items-center justify-center text-center bg-
                 green-500/10 text-green-300 rounded-lg">
                    <div>
                        <h2 className="text-2xl font-bold">Pull Request Created!</h2>
                        <p>Your PR from {fromBranch} to {toBranch} has been submitted.</p>
                    </div>
                </div>
            ) : (
                <form onSubmit={handleSubmit} className="flex-grow flex flex-col gap-4">
                    <div className="bg-slate-800/50 p-4 rounded-lg flex items-center gap-4">
                        <input type="text" value={fromBranch} onChange={e =>
                        setFromBranch(e.target.value)} className="px-3 py-1.5 rounded-md bg-slate-700
                        text-sm font-mono"/>
                        <span className="font-bold text-slate-400">→</span>
                         <input type="text" value={toBranch} onChange={e => setToBranch(e.target.value)}
                         className="px-3 py-1.5 rounded-md bg-slate-700 text-sm font-mono"/>
                    </div>
                    <div>
                        <label htmlFor="title" className="block text-sm font-medium text-
                        slate-400">Title</label>
                        <input type="text" id="title" value={title} onChange={e =>
                        setTitle(e.target.value)} className="w-full mt-1 px-3 py-2 rounded-md bg-
                        slate-800 border border-slate-700"/>
                    </div>
                    <div className="flex-grow flex flex-col">
                        <label htmlFor="description" className="block text-sm font-medium text-
                        slate-400">Description (Markdown supported)</label>
                        <textarea id="description" value={description} onChange={e =>
                        setDescription(e.target.value)} className="w-full mt-1 flex-grow p-3 rounded-md
                        bg-slate-800 border border-slate-700 resize-none"/>
                    </div>
                    <button type="submit" className="w-full px-6 py-3 bg-cyan-500 text-slate-900
                    font-bold rounded-md">Create Pull Request</button>
                </form>
            )}
        </div>
    );
};

// ===== ProjectMoodboard.tsx =====

import React, { useState } from 'react';
import { PhotoIcon } from '../icons/FeatureIcons.tsx';
```

```
import { useLocalStorage } from '../../hooks/useLocalStorage.ts';

interface Note {
    id: number;
    text: string;
    x: number;
    y: number;
    color: string;
}

const colors = ['bg-yellow-300', 'bg-green-300', 'bg-blue-300', 'bg-pink-300'];

export const ProjectMoodboard: React.FC = () => {
    const [notes, setNotes] = useLocalStorage<Note[]>('devcore_moodboard', []);
    const [dragging, setDragging] = useState<{ id: number; offsetX: number; offsetY:
    number } | null>(null);

    const addNote = () => {
        const newNote: Note = {
            id: Date.now(),
            text: 'New idea...',
            x: 50,
            y: 50,
            color: colors[Math.floor(Math.random() * colors.length)],
        };
        setNotes([...notes, newNote]);
    };

    const updateText = (id: number, text: string) => {
        setNotes(notes.map((n: Note) => n.id === id ? { ...n, text } : n));
    };

    const onMouseDown = (e: React.MouseEvent<HTMLDivElement>, id: number) => {
        const noteElement = e.currentTarget;
        const rect = noteElement.getBoundingClientRect();
        setDragging({
            id,
            offsetX: e.clientX - rect.left,
            offsetY: e.clientY - rect.top,
        });
    };

    const onMouseMove = (e: React.MouseEvent<HTMLDivElement>) => {
        if (!dragging) return;
        setNotes(
            notes.map((n: Note) =>
                n.id === dragging.id
                    ? { ...n, x: e.clientX - dragging.offsetX -
                    e.currentTarget.getBoundingClientRect().left, y: e.clientY - dragging.offsetY -
                    e.currentTarget.getBoundingClientRect().top }
                    : n
            )
        );
    };

    const onMouseUp = () => {
        setDragging(null);
    };

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8">
            <header className="mb-6 flex justify-between items-center">
```

```
                <div>
                    <h1 className="text-3xl font-bold text-slate-100 flex items-center">
                        <PhotoIcon />
                        <span className="ml-3">Project Moodboard</span>
                    </h1>
                    <p className="text-slate-400 mt-1">Organize your ideas with draggable sticky
                    notes.</p>
                </div>
                <button onClick={addNote} className="px-6 py-2 bg-cyan-500 text-slate-900 font-
                bold rounded-md">Add Note</button>
            </header>
            <div
                className="relative flex-grow bg-slate-900/50 border-2 border-dashed border-
                slate-700 rounded-lg overflow-hidden"
                onMouseMove={onMouseMove}
                onMouseUp={onMouseUp}
                onMouseLeave={onMouseUp}
            >
                {notes.map((note: Note) => (
                    <div
                        key={note.id}
                        className={`absolute w-48 h-48 p-2 flex flex-col shadow-lg cursor-grab
                        active:cursor-grabbing ${note.color}`}
                        style={{ top: note.y, left: note.x, transform: dragging?.id === note.id ?
                        'scale(1.05)' : 'scale(1)' }}
                        onMouseDown={e => onMouseDown(e, note.id)}
                    >
                        <textarea
                            value={note.text}
                            onChange={(e) => updateText(note.id, e.target.value)}
                             onMouseDown={(e) => e.stopPropagation()}
                            className="w-full h-full bg-transparent text-black resize-none focus:outline-
                            none font-medium"
                        />
                    </div>
                ))}
            </div>
        </div>
    );
};

// ===== PromptCraftPad.tsx =====

import React, { useState, useEffect } from 'react';
import { SparklesIcon } from '../icons/FeatureIcons.tsx';
import * as storageService from '../../services/storageService.ts';

interface Prompt {
    id: number;
    name: string;
    text: string;
}

const STORAGE_KEY = 'devcore_prompts';
const INITIAL_PROMPT = [{ id: 1, name: 'React Component Generator', text:
'Generate a React component named {name} that {description}. Style it with
Tailwind CSS.'}];

export const PromptCraftPad: React.FC = () => {
    const [prompts, setPrompts] = useState<Prompt[]>([]);
    const [activePrompt, setActivePrompt] = useState<Prompt | null>(null);
    useEffect(() => {
```

```
    const loadPrompts = async () => {
        const saved = await storageService.dbGet(STORAGE_KEY);
        const initialPrompts = saved || INITIAL_PROMPT;
        setPrompts(initialPrompts);
        if (!activePrompt && initialPrompts.length > 0) {
            setActivePrompt(initialPrompts[0]);
        }
    };
    loadPrompts();
}, []);

useEffect(() => {
    const handler = setTimeout(() => {
        if (prompts.length > 0) {
            storageService.dbSet(STORAGE_KEY, prompts);
        }
    }, 500);
    return () => clearTimeout(handler);
}, [prompts]);

useEffect(() => {
    if (activePrompt) {
        const freshPrompt = prompts.find((p: Prompt) => p.id === activePrompt.id);
        setActivePrompt(freshPrompt || prompts[0] || null);
    } else if (prompts.length > 0) {
        setActivePrompt(prompts[0]);
    }
}, [prompts]);


const handleTextChange = (e: React.ChangeEvent<HTMLTextAreaElement>) => {
    if (!activePrompt) return;
    const updatedPrompt = { ...activePrompt, text: e.target.value };
    setActivePrompt(updatedPrompt);
    setPrompts(prompts.map((p: Prompt) => p.id === updatedPrompt.id ? updatedPrompt
    : p));
};

const handleAddNew = () => {
    const newPrompt = { id: Date.now(), name: 'New Untitled Prompt', text: '' };
    setPrompts([...prompts, newPrompt]);
    setActivePrompt(newPrompt);
};

const handleDelete = (id: number) => {
    setPrompts(prompts.filter((p: Prompt) => p.id !== id));
    if(activePrompt?.id === id) setActivePrompt(prompts[0] || null);
}

return (
    <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8">
        <header className="mb-6">
            <h1 className="text-3xl font-bold text-slate-100 flex items-center">
                <SparklesIcon />
                <span className="ml-3">Prompt Craft Pad</span>
            </h1>
            <p className="text-slate-400 mt-1">Create, save, and manage your favorite AI
            prompts.</p>
        </header>
        <div className="flex-grow flex gap-6 min-h-0">
            <aside className="w-1/3 bg-slate-800/50 p-4 rounded-lg flex flex-col">
                <h3 className="font-bold mb-2">My Prompts</h3>
```

```
                    <ul className="space-y-2 flex-grow overflow-y-auto">
                        {prompts.map((prompt: Prompt) => (
                            <li key={prompt.id} className="group flex items-center justify-between">
                                <button onClick={() => setActivePrompt(prompt)} className={`w-full text-left
                                px-3 py-2 rounded-md ${activePrompt?.id === prompt.id ? 'bg-cyan-500/20 text-
                                cyan-300' : 'hover:bg-slate-700/50'}`}>
                                    {prompt.name}
                                </button>
                                 <button onClick={() => handleDelete(prompt.id)} className="ml-2 p-1 text-
                                 slate-500 hover:text-red-400 opacity-0 group-hover:opacity-100">&times;</button>
                            </li>
                        ))}
                    </ul>
                    <div className="mt-4 pt-4 border-t border-slate-700">
                         <button onClick={handleAddNew} className="w-full text-sm py-2 bg-cyan-500/80
                         text-white rounded-md">Add New Prompt</button>
                    </div>
                </aside>
                <main className="w-2/3 flex flex-col">
                    {activePrompt ? (
                        <>
                            <label htmlFor="prompt-editor" className="text-sm font-medium text-slate-400
                            mb-2">{activePrompt.name}</label>
                            <textarea
                                id="prompt-editor"
                                value={activePrompt.text}
                                onChange={handleTextChange}
                                className="flex-grow p-4 bg-slate-900 border border-slate-700 rounded-md resize-
                                none font-mono text-sm text-cyan-300 focus:ring-2 focus:ring-cyan-500
                                focus:outline-none"
                            />
                        </>
                    ) : (
                        <div className="flex-grow flex items-center justify-center bg-slate-900 rounded-
                        lg text-slate-500">
                            Select a prompt or create a new one.
                        </div>
                    )}
                </main>
            </div>
        </div>
    );
};


// ===== PrSummaryGenerator.tsx =====



import React, { useState, useCallback } from 'react';
import { generatePrSummaryStream } from '../../services/geminiService.ts';
import { LoadingSpinner } from '../shared/LoadingSpinner.tsx';
import { marked } from 'marked';
import { useFeatureState } from '../../hooks/useFeatureState.ts';
import { FeatureHeader } from '../shared/FeatureHeader.tsx';
import type { Feature } from '../../types.ts';

const exampleDiff = `diff --git a/src/App.tsx b/src/App.tsx
--- a/src/App.tsx
+++ b/src/App.tsx
@@ -1,5 +1,6 @@
 import React from 'react';
```

```
 import './App.css';
+import LoginButton from './LoginButton';

 function App() {
   return (
     <div className="App">
       <header className="App-header">
         <p>
           Hello, world!
         </p>
+        <LoginButton />
       </header>
     </div>
   );
 }
`;

export const PrSummaryGenerator: React.FC<{ feature: Feature }> = ({ feature })
=> {
    const [state, setState, saveState] = useFeatureState('prSummaryGenerator', {
    diff: exampleDiff, summary: '' });
    const { diff, summary } = state;

    const [isLoading, setIsLoading] = useState<boolean>(false);
    const [error, setError] = useState<string>('');

    const handleGenerate = useCallback(async () => {
        if (!diff.trim()) {
            setError('Please paste a diff to generate a summary.');
            return;
        }
        setIsLoading(true);
        setError('');
        setState(s => ({ ...s, summary: ''}));
        try {
            const stream = generatePrSummaryStream(diff);
            let fullResponse = '';
            for await (const chunk of stream) {
                fullResponse += chunk;
                setState(s => ({ ...s, summary: fullResponse}));
            }
        } catch (err) {
            const errorMessage = err instanceof Error ? err.message : 'An unknown error
            occurred.';
            setError(`Failed to generate summary: ${errorMessage}`);
        } finally {
            setIsLoading(false);
        }
    }, [diff, setState]);

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8">
            <FeatureHeader feature={feature} onSaveState={saveState} />
            <div className="flex-grow grid grid-cols-1 lg:grid-cols-2 gap-6 h-full overflow-
            hidden">
                <div className="flex flex-col h-full">
                    <label htmlFor="diff-input" className="text-sm font-medium text-slate-400
                    mb-2">Git Diff</label>
                    <textarea
                        id="diff-input"
                        value={diff}
                        onChange={(e) => setState(s => ({...s, diff: e.target.value}))}
```

```
                        className="flex-grow p-4 bg-slate-900 border border-slate-700 rounded-md resize-
                        none font-mono text-sm"
                    />
                    <button
                        onClick={handleGenerate}
                        disabled={isLoading}
                        className="mt-4 w-full flex items-center justify-center px-6 py-3 bg-cyan-500
                        text-slate-900 font-bold rounded-md hover:bg-cyan-400 disabled:bg-slate-600
                        disabled:cursor-not-allowed"
                    >
                        {isLoading ? <LoadingSpinner /> : 'Generate Summary'}
                    </button>
                </div>
                <div className="flex flex-col h-full">
                    <label className="text-sm font-medium text-slate-400 mb-2">Generated
                    Summary</label>
                    <div className="flex-grow p-4 bg-slate-800/50 border border-slate-700/50
                    rounded-md overflow-y-auto">
                        {isLoading && (
                            <div className="flex items-center justify-center h-full">
                                <LoadingSpinner />
                            </div>
                        )}
                        {error && <p className="text-red-400">{error}</p>}
                        {summary && !isLoading && (
                             <div
                                className="prose prose-sm prose-invert max-w-none"
                                dangerouslySetInnerHTML={{ __html: marked.parse(summary) as string }}
                            />
                        )}
                         {!isLoading && !summary && !error && (
                            <div className="text-slate-500 h-full flex items-center justify-center">
                                The summary will appear here.
                            </div>
                        )}
                    </div>
                </div>
            </div>
        </div>
    );
};

// ===== PwaManifestEditor.tsx =====


import React, { useState, useMemo } from 'react';
import { CodeBracketSquareIcon } from '../icons/FeatureIcons.tsx';

interface ManifestData {
    name: string;
    short_name: string;
    start_url: string;
    display: 'standalone' | 'fullscreen' | 'minimal-ui';
    background_color: string;
    theme_color: string;
}

export const PwaManifestEditor: React.FC = () => {
    const [manifest, setManifest] = useState<ManifestData>({
        name: 'DevCore 100 Progressive Web App',
        short_name: 'DevCore100',
        start_url: '/',
```

```
    display: 'standalone',
    background_color: '#1e293b',
    theme_color: '#06b6d4',
});

const handleChange = (e: React.ChangeEvent<HTMLInputElement |
HTMLSelectElement>) => {
    setManifest({ ...manifest, [e.target.name]: e.target.value });
};

const generatedJson = useMemo(() => {
    const fullManifest = {
        ...manifest,
        icons: [
            {
                "src": "icon-192.png",
                "type": "image/png",
                "sizes": "192x192"
            },
            {
                "src": "icon-512.png",
                "type": "image/png",
                "sizes": "512x512"
            }
        ]
    }
    return JSON.stringify(fullManifest, null, 2);
}, [manifest]);

const handleCopy = () => {
    navigator.clipboard.writeText(generatedJson);
};

return (
    <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8">
        <header className="mb-6">
            <h1 className="text-3xl font-bold text-slate-100 flex items-center">
                <CodeBracketSquareIcon />
                <span className="ml-3">PWA Manifest Editor</span>
            </h1>
            <p className="text-slate-400 mt-1">Configure and generate the `manifest.json`
            file for your PWA.</p>
        </header>
        <div className="flex-grow grid grid-cols-1 lg:grid-cols-2 gap-6 min-h-0">
            <div className="flex flex-col gap-4 bg-slate-800/50 p-6 rounded-lg overflow-y-
            auto">
                <h3 className="text-xl font-bold">Configuration</h3>
                <div>
                    <label htmlFor="name" className="block text-sm font-medium text-slate-400">App
                    Name</label>
                    <input type="text" name="name" value={manifest.name} onChange={handleChange}
                    className="w-full mt-1 px-3 py-2 rounded-md bg-slate-800 border border-
                    slate-700"/>
                </div>
                <div>
                    <label htmlFor="short_name" className="block text-sm font-medium text-
                    slate-400">Short Name</label>
                    <input type="text" name="short_name" value={manifest.short_name}
                    onChange={handleChange} className="w-full mt-1 px-3 py-2 rounded-md bg-slate-800
                    border border-slate-700"/>
                </div>
                 <div>
```

```
                        <label htmlFor="start_url" className="block text-sm font-medium text-
                        slate-400">Start URL</label>
                        <input type="text" name="start_url" value={manifest.start_url}
                        onChange={handleChange} className="w-full mt-1 px-3 py-2 rounded-md bg-slate-800
                        border border-slate-700"/>
                    </div>
                     <div>
                        <label htmlFor="display" className="block text-sm font-medium text-
                        slate-400">Display Mode</label>
                        <select name="display" value={manifest.display} onChange={handleChange}
                        className="w-full mt-1 px-3 py-2 rounded-md bg-slate-800 border border-
                        slate-700">
                            <option>standalone</option>
                            <option>fullscreen</option>
                            <option>minimal-ui</option>
                        </select>
                    </div>
                     <div className="flex gap-4">
                        <div className="w-1/2">
                            <label htmlFor="background_color" className="block text-sm font-medium text-
                            slate-400">Background Color</label>
                            <input type="color" name="background_color" value={manifest.background_color}
                            onChange={handleChange} className="w-full mt-1 h-10 rounded-md bg-slate-800
                            border border-slate-700"/>
                        </div>
                        <div className="w-1/2">
                            <label htmlFor="theme_color" className="block text-sm font-medium text-
                            slate-400">Theme Color</label>
                            <input type="color" name="theme_color" value={manifest.theme_color}
                            onChange={handleChange} className="w-full mt-1 h-10 rounded-md bg-slate-800
                            border border-slate-700"/>
                        </div>
                     </div>
                </div>
                <div className="flex flex-col">
                     <label className="text-sm font-medium text-slate-400 mb-2">Generated
                     manifest.json</label>
                     <div className="relative flex-grow">
                        <pre className="w-full h-full bg-slate-900 p-4 rounded-md text-cyan-300 text-sm
                        overflow-auto">{generatedJson}</pre>
                        <button onClick={handleCopy} className="absolute top-2 right-2 px-2 py-1 bg-
                        slate-700 hover:bg-slate-600 rounded-md text-xs">Copy</button>
                    </div>
                </div>
            </div>
        </div>
    );
};

// ===== RegexSandbox.tsx =====



import React, { useState, useMemo, useCallback } from 'react';
import { generateRegExStream } from '../../services/geminiService.ts';
import { LoadingSpinner } from '../shared/LoadingSpinner.tsx';
import { useFeatureState } from '../../hooks/useFeatureState.ts';
import { FeatureHeader } from '../shared/FeatureHeader.tsx';
import type { Feature } from '../../types.ts';

export const RegexSandbox: React.FC<{ feature: Feature }> = ({ feature }) => {
    const [state, setState, saveState] = useFeatureState('regexSandbox', {
```

```
    pattern: '/\\b([a-z]+)ing\\b/g',
    testString: 'The quick brown fox is jumping over the lazy dog. Testing...',
    aiPrompt: 'find words ending in "ing"',
});
const { pattern, testString, aiPrompt } = state;

const [isAiLoading, setIsAiLoading] = useState<boolean>(false);

const { matches, error } = useMemo(() => {
    try {
        const patternParts = pattern.match(/^\/(.*)\/([gimyus]*)$/);
        if (!patternParts) {
            return { matches: null, error: 'Invalid regex literal format. Use
            /pattern/flags.' };
        }
        const [, regexBody, regexFlags] = patternParts;
        const regex = new RegExp(regexBody, regexFlags);
        const allMatches = [...testString.matchAll(regex)];
        return { matches: allMatches, error: null };
    } catch (e) {
        if (e instanceof Error) {
            return { matches: null, error: e.message };
        }
        return { matches: null, error: 'An unknown error occurred.' };
    }
}, [pattern, testString]);

const handleGenerateRegex = useCallback(async () => {
    if (!aiPrompt) return;
    setIsAiLoading(true);
    try {
        const stream = generateRegExStream(aiPrompt);
        let fullResponse = '';
        for await (const chunk of stream) {
            fullResponse += chunk;
        }
        setState(s => ({...s, pattern: fullResponse.trim() }));
    } catch(e) {
        console.error(e);
    } finally {
        setIsAiLoading(false);
    }
}, [aiPrompt, setState]);

const highlightedString = useMemo(() => {
    if (!matches || matches.length === 0) return testString;
    let lastIndex = 0;
    const parts = [];
    matches.forEach((match, i) => {
        if (match.index === undefined) return;
        parts.push(testString.substring(lastIndex, match.index));
        parts.push(<mark key={i} className="bg-cyan-500/30 text-cyan-200 rounded
        px-1">{match[0]}</mark>);
        lastIndex = match.index + match[0].length;
    });
    parts.push(testString.substring(lastIndex));
    return parts;
}, [matches, testString]);

return (
    <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8">
        <FeatureHeader feature={feature} onSaveState={saveState} />
```

```
            <div className="flex flex-col gap-4 flex-grow min-h-0">
                <div className="flex gap-2">
                    <input
                        type="text"
                        value={aiPrompt}
                        onChange={(e) => setState(s => ({ ...s, aiPrompt: e.target.value }))}
                        placeholder="Describe the pattern to find..."
                        className="flex-grow px-3 py-1.5 rounded-md bg-slate-800 border border-slate-700
                        text-sm focus:ring-2 focus:ring-cyan-500"
                    />
                    <button onClick={handleGenerateRegex} disabled={isAiLoading} className="px-4
                    py-1.5 bg-cyan-500 text-slate-900 font-bold rounded-md flex items-center">
                        {isAiLoading ? <LoadingSpinner/> : 'Generate'}
                    </button>
                </div>
                <div>
                    <label htmlFor="regex-pattern" className="text-sm font-medium text-
                    slate-400">Regular Expression</label>
                    <input
                        id="regex-pattern"
                        type="text"
                        value={pattern}
                        onChange={(e) => setState(s => ({ ...s, pattern: e.target.value }))}
                        className={`w-full mt-1 px-3 py-2 rounded-md bg-slate-900 border ${error ?
                        'border-red-500' : 'border-slate-700'} font-mono text-sm focus:ring-2
                        focus:ring-cyan-500`}
                    />
                     {error && <p className="text-red-400 text-xs mt-1">{error}</p>}
                </div>
                <div className="flex flex-col flex-grow min-h-0">
                    <label htmlFor="test-string" className="text-sm font-medium text-slate-400">Test
                    String</label>
                    <textarea
                        id="test-string"
                        value={testString}
                        onChange={(e) => setState(s => ({ ...s, testString: e.target.value }))}
                        className="w-full mt-1 p-3 rounded-md bg-slate-900 border border-slate-700 font-
                        mono text-sm resize-y h-32"
                    />
                    <div className="mt-2 p-3 bg-slate-800/50 rounded-md border border-slate-700/50">
                        {highlightedString}
                    </div>
                </div>
                <div className="flex-shrink-0">
                    <h3 className="text-lg font-bold">Matches ({matches?.length || 0})</h3>
                     <div className="mt-2 p-2 bg-slate-900 rounded-md overflow-y-auto max-h-48">
                        {matches && matches.length > 0 ? (
                            <pre className="text-xs text-green-300">{JSON.stringify(matches, null, 2)}</pre>
                        ) : (
                            <p className="text-slate-500 text-sm">No matches found.</p>
                        )}
                    </div>
                </div>
            </div>
        </div>
    );
};


// ===== ResponsiveTester.tsx =====
import React, { useState } from 'react';
import { EyeIcon } from '../icons/FeatureIcons.tsx';
```

```
const devices = {
    'iPhone 12': { width: 390, height: 844 },
    'iPad Air': { width: 820, height: 1180 },
    'Laptop': { width: 1366, height: 768 },
    'Desktop': { width: 1920, height: 1080 },
    'Auto': { width: '100%', height: '100%' },
};

type DeviceName = keyof typeof devices;

export const ResponsiveTester: React.FC = () => {
    const [url, setUrl] = useState('https://react.dev');
    const [displayUrl, setDisplayUrl] = useState(url);
    const [activeDevice, setActiveDevice] = useState<DeviceName>('Auto');

    const iframeSize = devices[activeDevice];

    const handleUrlSubmit = (e: React.FormEvent) => {
        e.preventDefault();
        // A simple check to prepend https:// if missing
        setDisplayUrl(url.startsWith('http') ? url : `https://${url}`);
    };

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8">
            <header className="mb-6">
                <h1 className="text-3xl font-bold text-slate-100 flex items-center">
                    <EyeIcon />
                    <span className="ml-3">Responsive Tester</span>
                </h1>
                <p className="text-slate-400 mt-1">Preview your web pages at different screen
                sizes.</p>
            </header>
            <form onSubmit={handleUrlSubmit} className="flex items-center gap-2 mb-4">
                <input
                    type="text"
                    value={url}
                    onChange={(e) => setUrl(e.target.value)}
                    placeholder="https://example.com"
                    className="flex-grow px-4 py-2 rounded-md bg-slate-800 border border-slate-700
                    focus:ring-2 focus:ring-cyan-500 focus:outline-none"
                />
                <button type="submit" className="px-6 py-2 bg-cyan-500 text-slate-900 font-bold
                rounded-md hover:bg-cyan-400">Load</button>
            </form>
            <div className="bg-slate-800/50 p-2 rounded-lg flex justify-center items-center
            gap-2 mb-4">
                {Object.keys(devices).map(name => (
                    <button
                        key={name}
                        onClick={() => setActiveDevice(name as DeviceName)}
                        className={`px-3 py-1 rounded-md text-sm ${activeDevice === name ? 'bg-
                        cyan-500/20 text-cyan-300' : 'hover:bg-slate-700'}`}
                    >
                        {name}
                    </button>
                ))}
            </div>
            <div className="flex-grow flex items-center justify-center bg-slate-900 rounded-
            lg p-4 overflow-auto">
                <iframe
                    key={displayUrl} // Force re-render on URL change
```

```
                    src={displayUrl}
                    style={{ width: iframeSize.width, height: iframeSize.height, maxWidth: '100%',
                    maxHeight: '100%' }}
                    className="bg-white border-4 border-slate-700 rounded-md transition-all
                    duration-300"
                    title="Responsive Preview"
                />
            </div>
        </div>
    );
};

// ===== SassScssCompiler.tsx =====

import React, { useState, useMemo } from 'react';
import { CodeBracketSquareIcon } from '../icons/FeatureIcons.tsx';

const initialScss = `$primary-color: #06b6d4;\n\n.container {\n  padding:
20px;\n\n  .title {\n    color: $primary-color;\n    font-size: 24px;\n  }\n}`;

export const SassScssCompiler: React.FC = () => {
    const [scss, setScss] = useState(initialScss);

    const compiledCss = useMemo(() => {
        // This is a very basic simulation. A real implementation would use a SASS
        compiler.
        let css = scss.replace(/\$(\w+):\s*(.*?);/g, (_, name, value) => `--${name}:
        ${value};`);
        css = css.replace(/color:\s*\$(\w+);/g, 'color: var(--$1);');
        css = css.replace(/\.container\s*\{([\s\S]*?)\}/g, (match, content) => {
            const inner = content.replace(/\.title\s*\{([\s\S]*?)\}/g, (m, innerContent) =>
            {
                return `\n.container .title {\n  ${innerContent.trim()}\n}`;
            });
            return `.container {\n  ${inner.trim().split('\n').slice(0,-1).join('\n')}\n}`;
        });
        return css;
    }, [scss]);

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8">
            <header className="mb-6">
                <h1 className="text-3xl font-bold text-slate-100 flex items-center">
                    <CodeBracketSquareIcon />
                    <span className="ml-3">SASS/SCSS Compiler (Simulation)</span>
                </h1>
                <p className="text-slate-400 mt-1">A basic, real-time simulation of a SASS/SCSS
                to CSS compiler.</p>
            </header>
            <div className="flex-grow grid grid-cols-1 lg:grid-cols-2 gap-6 h-full overflow-
            hidden">
                <div className="flex flex-col h-full">
                    <label htmlFor="scss-input" className="text-sm font-medium text-slate-400
                    mb-2">SASS/SCSS Input</label>
                    <textarea
                        id="scss-input"
                        value={scss}
                        onChange={(e) => setScss(e.target.value)}
                        className="flex-grow p-4 bg-slate-900 border border-slate-700 rounded-md resize-
                        none font-mono text-sm text-pink-400"
                    />
                </div>
```

```
                <div className="flex flex-col h-full">
                    <label className="text-sm font-medium text-slate-400 mb-2">Compiled CSS
                    Output</label>
                    <pre className="flex-grow p-4 bg-slate-800/50 border border-slate-700/50
                    rounded-md overflow-y-auto text-cyan-300 font-mono text-sm">
                        {compiledCss}
                    </pre>
                </div>
            </div>
        </div>
    );
};

// ===== SchemaDesigner.tsx =====


import React, { useState } from 'react';
import { MapIcon } from '../icons/FeatureIcons.tsx';

interface Column {
    id: number;
    name: string;
    type: string;
}

interface Table {
    id: number;
    name: string;
    columns: Column[];
}

export const SchemaDesigner: React.FC = () => {
    const [tables, setTables] = useState<Table[]>([
        { id: 1, name: 'users', columns: [{ id: 1, name: 'id', type: 'INT' }, {id: 2,
        name: 'username', type: 'VARCHAR'}] },
        { id: 2, name: 'posts', columns: [{ id: 1, name: 'id', type: 'INT' }, {id: 2,
        name: 'user_id', type: 'INT'}, {id: 3, name: 'content', type: 'TEXT'}] },
    ]);
    const [newTableName, setNewTableName] = useState('');

    const addTable = () => {
        if(!newTableName) return;
        setTables([...tables, { id: Date.now(), name: newTableName, columns: [{ id:
        Date.now(), name: 'id', type: 'INT' }] }]);
        setNewTableName('');
    };

    const addColumn = (tableId: number) => {
        setTables(tables.map(t => t.id === tableId ? {...t, columns: [...t.columns, {id:
        Date.now(), name: 'new_column', type: 'VARCHAR'}]} : t));
    };

    const updateColumn = (tableId: number, colId: number, field: 'name' | 'type',
    value: string) => {
        setTables(tables.map(t => t.id === tableId ? {...t, columns: t.columns.map(c =>
        c.id === colId ? {...c, [field]: value} : c)} : t));
    };

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8">
            <header className="mb-6">
                <h1 className="text-3xl font-bold text-slate-100 flex items-center">
```

```
                    <MapIcon />
                    <span className="ml-3">Schema Designer</span>
                </h1>
                <p className="text-slate-400 mt-1">Visually design your database schema.</p>
            </header>
            <div className="flex-grow grid grid-cols-1 lg:grid-cols-2 gap-6 min-h-0">
                <div className="flex flex-col gap-4 overflow-y-auto pr-2">
                     <div className="flex gap-2">
                        <input type="text" value={newTableName} onChange={e =>
                        setNewTableName(e.target.value)} placeholder="New table name..."
                        className="flex-grow px-3 py-1.5 rounded-md bg-slate-800 border border-slate-700
                        text-sm"/>
                        <button onClick={addTable} className="px-4 py-1.5 bg-cyan-500 text-slate-900
                        font-bold rounded-md">Add Table</button>
                    </div>
                    <div className="space-y-4">
                    {tables.map(table => (
                        <div key={table.id} className="bg-slate-800/50 p-4 rounded-lg">
                            <h3 className="font-bold text-cyan-400 text-lg mb-2">{table.name}</h3>
                            <div className="space-y-2">
                                {table.columns.map(col => (
                                    <div key={col.id} className="flex items-center gap-2 font-mono text-sm">
                                        <input value={col.name} onChange={e => updateColumn(table.id, col.id, 'name',
                                        e.target.value)} className="w-1/2 px-2 py-1 rounded bg-slate-800 border border-
                                        slate-700"/>
                                        <input value={col.type} onChange={e => updateColumn(table.id, col.id, 'type',
                                        e.target.value)} className="w-1/2 px-2 py-1 rounded bg-slate-800 border border-
                                        slate-700"/>
                                    </div>
                                ))}
                            </div>
                            <button onClick={() => addColumn(table.id)} className="text-xs mt-3 px-3 py-1
                            bg-slate-700 rounded-md">Add Column</button>
                        </div>
                    ))}
                    </div>
                </div>
                <div className="flex flex-col">
                    <label className="text-sm font-medium text-slate-400 mb-2">JSON Output</label>
                    <div className="relative flex-grow">
                        <pre className="w-full h-full bg-slate-900 p-4 rounded-md text-cyan-300 text-sm
                        overflow-auto">
                            {JSON.stringify(tables, null, 2)}
                        </pre>
                        <button onClick={() => navigator.clipboard.writeText(JSON.stringify(tables,
                        null, 2))} className="absolute top-2 right-2 px-2 py-1 bg-slate-700 hover:bg-
                        slate-600 rounded-md text-xs">Copy</button>
                    </div>
                </div>
            </div>
        </div>
    );
};

// ===== ScratchpadTerminal.tsx =====

import React from 'react';
import { TerminalIcon } from '../icons/FeatureIcons.tsx';
import TerminalComponent from '../Terminal.tsx';

export const ScratchpadTerminal: React.FC = () => {
    return (
```

```
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8">
            <header className="mb-6">
                <h1 className="text-3xl font-bold text-slate-100 flex items-center">
                    <TerminalIcon />
                    <span className="ml-3">Scratchpad Terminal</span>
                </h1>
                <p className="text-slate-400 mt-1">An isolated terminal for quick tests and
                experiments.</p>
            </header>
            <div className="flex-grow bg-slate-900 rounded-lg overflow-hidden border border-
            slate-700">
                <TerminalComponent initialMessage="Welcome to the Scratchpad Terminal. This is a
                temporary, isolated environment." />
            </div>
        </div>
    );
};

// ===== ScreenshotToComponent.tsx =====



import React, { useState, useCallback } from 'react';
import { generateComponentFromImageStream } from
'../../services/geminiService.ts';
import { LoadingSpinner } from '../shared/LoadingSpinner.tsx';
import { marked } from 'marked';
import { blobToBase64, blobToDataURL } from '../../services/fileUtils.ts';
import { FeatureHeader } from '../shared/FeatureHeader.tsx';
import type { Feature } from '../../types.ts';


export const ScreenshotToComponent: React.FC<{ feature: Feature }> = ({ feature
}) => {
    const [pastedImage, setPastedImage] = useState<string | null>(null);
    const [code, setCode] = useState('');
    const [isLoading, setIsLoading] = useState(false);
    const [error, setError] = useState('');

    const handlePaste = useCallback(async (event: React.ClipboardEvent) => {
        const items = event.clipboardData.items;
        for (let i = 0; i < items.length; i++) {
            if (items[i].type.indexOf('image') !== -1) {
                const blob = items[i].getAsFile();
                if (blob) {
                    try {
                        const [dataUrl, base64Image] = await Promise.all([
                            blobToDataURL(blob),
                            blobToBase64(blob)
                        ]);
                        setPastedImage(dataUrl);
                        handleGenerate(base64Image);
                    } catch (e) {
                        setError('Could not process pasted image.');
                        console.error(e);
                    }
                }
                return;
            }
        }
    }, []);
    const handleGenerate = async (base64Image: string) => {
```

```
    setIsLoading(true);
    setError('');
    setCode('');
    try {
        const stream = generateComponentFromImageStream(base64Image);
        let fullResponse = '';
        for await (const chunk of stream) {
            fullResponse += chunk;
            setCode(fullResponse);
        }
    } catch (err) {
        const errorMessage = err instanceof Error ? err.message : 'An unknown error
        occurred.';
        setError(`Failed to generate component: ${errorMessage}`);
    } finally {
        setIsLoading(false);
    }
};

return (
    <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8">
        <FeatureHeader feature={feature} onSaveState={() => {}} />
        <div className="flex-grow grid grid-cols-1 lg:grid-cols-2 gap-6 min-h-0">
            <div
                onPaste={handlePaste}
                className="flex flex-col items-center justify-center bg-slate-800/50 p-6
                rounded-lg border-2 border-dashed border-slate-700 focus:outline-none
                focus:border-cyan-500"
                tabIndex={0}
            >
                {pastedImage ? (
                    <img src={pastedImage} alt="Pasted content" className="max-w-full max-h-full
                    object-contain rounded-md shadow-lg" />
                ) : (
                    <div className="text-center text-slate-400">
                        <h2 className="text-xl font-bold">Paste an image here</h2>
                        <p>(Cmd/Ctrl + V)</p>
                    </div>
                )}
            </div>
            <div className="flex flex-col h-full">
                <label className="text-sm font-medium text-slate-400 mb-2">Generated
                Code</label>
                <div className="flex-grow p-1 bg-slate-900 border border-slate-700 rounded-md
                overflow-y-auto">
                    {isLoading && (
                        <div className="flex items-center justify-center h-full"><LoadingSpinner
                        /></div>
                    )}
                    {error && <p className="p-4 text-red-400">{error}</p>}
                    {code && !isLoading && (
                        <div
                            className="prose prose-sm prose-invert max-w-none prose-pre:bg-transparent
                            prose-pre:p-4 prose-pre:m-0 prose-code:text-cyan-300"
                            dangerouslySetInnerHTML={{ __html: marked.parse(code) as string }}
                        />
                    )}
                    {!isLoading && !code && !error && (
                        <div className="text-slate-500 h-full flex items-center justify-
                        center">Generated component code will appear here.</div>
                    )}
                </div>
```

```
                </div>
            </div>
        </div>
    );
};

// ===== SnippetVault.tsx =====

import React, { useState, useEffect } from 'react';
import { LockClosedIcon, SparklesIcon } from '../icons/FeatureIcons.tsx';
import { enhanceSnippetStream } from '../../services/geminiService.ts';
import { LoadingSpinner } from '../shared/LoadingSpinner.tsx';
import * as storageService from '../../services/storageService.ts';

interface Snippet {
    id: number;
    name: string;
    code: string;
    language: string;
}

const STORAGE_KEY = 'devcore_snippets';
const INITIAL_SNIPPET = [{ id: 1, name: 'React Hook Boilerplate', language:
'javascript', code: `import { useState } from 'react';\n\nconst useCustomHook =
() => {\n  const [value, setValue] = useState(null);\n  return { value, setValue
};\n};`}];

export const SnippetVault: React.FC = () => {
    const [snippets, setSnippets] = useState<Snippet[]>([]);
    const [activeSnippet, setActiveSnippet] = useState<Snippet | null>(null);
    const [isEnhancing, setIsEnhancing] = useState(false);

    useEffect(() => {
        const loadSnippets = async () => {
            const saved = await storageService.dbGet(STORAGE_KEY);
            const initialSnippets = saved || INITIAL_SNIPPET;
            setSnippets(initialSnippets);
            if (!activeSnippet && initialSnippets.length > 0) {
                setActiveSnippet(initialSnippets[0]);
            }
        };
        loadSnippets();
    }, []);

    useEffect(() => {
        const handler = setTimeout(() => {
            if (snippets.length > 0) {
                 storageService.dbSet(STORAGE_KEY, snippets);
            }
        }, 500);
        return () => clearTimeout(handler);
    }, [snippets]);

    useEffect(() => {
        if (activeSnippet) {
            const freshSnippet = snippets.find((s: Snippet) => s.id === activeSnippet.id);
            setActiveSnippet(freshSnippet || snippets[0] || null);
        } else if (snippets.length > 0) {
            setActiveSnippet(snippets[0]);
        }
    }, [snippets]);
    const updateSnippetInState = (snippet: Snippet) => {
```

```
    setActiveSnippet(snippet);
    setSnippets(snippets.map((s: Snippet) => s.id === snippet.id ? snippet : s));
}

const handleCodeChange = (e: React.ChangeEvent<HTMLTextAreaElement>) => {
    if (!activeSnippet) return;
    updateSnippetInState({ ...activeSnippet, code: e.target.value });
};

const handleEnhance = async () => {
    if (!activeSnippet) return;
    setIsEnhancing(true);
    try {
        const stream = enhanceSnippetStream(activeSnippet.code);
        let fullResponse = '';
        for await (const chunk of stream) {
            fullResponse += chunk;
            // Clean markdown fences for live update
            const cleanedChunk = fullResponse.replace(/^```(?:\w+\n)?/, '').replace(/```$/,
            '');
            updateSnippetInState({ ...activeSnippet, code: cleanedChunk });
        }
    } catch (e) {
        console.error(e);
    } finally {
        setIsEnhancing(false);
    }
};

const handleAddNew = () => {
    const newSnippet = { id: Date.now(), name: 'New Snippet', language: 'plaintext',
    code: '' };
    setSnippets([...snippets, newSnippet]);
    setActiveSnippet(newSnippet);
};

const handleDelete = (id: number) => {
    setSnippets(snippets.filter((s: Snippet) => s.id !== id));
    if(activeSnippet?.id === id) setActiveSnippet(snippets[0] || null);
}

return (
    <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8">
        <header className="mb-6">
            <h1 className="text-3xl font-bold text-slate-100 flex items-
            center"><LockClosedIcon /><span className="ml-3">Portable Snippet
            Vault</span></h1>
            <p className="text-slate-400 mt-1">Save, reuse, and enhance your favorite code
            snippets with AI.</p>
        </header>
        <div className="flex-grow flex gap-6 min-h-0">
            <aside className="w-1/3 bg-slate-800/50 p-4 rounded-lg flex flex-col">
                <h3 className="font-bold mb-2">My Snippets</h3>
                <ul className="space-y-2 flex-grow overflow-y-auto">
                    {snippets.map((snippet: Snippet) => (
                        <li key={snippet.id} className="group flex items-center justify-between">
                            <button onClick={() => setActiveSnippet(snippet)} className={`w-full text-left
                            px-3 py-2 rounded-md ${activeSnippet?.id === snippet.id ? 'bg-cyan-500/20 text-
                            cyan-300' : 'hover:bg-slate-700/50'}`}>
                                {snippet.name}
                            </button>
                             <button onClick={() => handleDelete(snippet.id)} className="ml-2 p-1 text-
```

```
                                slate-500 hover:text-red-400 opacity-0 group-hover:opacity-100">&times;</button>
                            </li>
                        ))}
                    </ul>
                    <div className="mt-4 pt-4 border-t border-slate-700">
                        <button onClick={handleAddNew} className="w-full text-sm py-2 bg-cyan-500/80
                        text-white rounded-md">Add New Snippet</button>
                    </div>
                </aside>
                <main className="w-2/3 flex flex-col">
                    {activeSnippet ? (
                        <>
                            <div className="flex justify-between items-center mb-2">
                                <h3 className="text-lg font-bold text-slate-200">{activeSnippet.name}</h3>
                                <div className="flex gap-2">
                                    <button onClick={handleEnhance} disabled={isEnhancing} className="flex items-
                                    center gap-2 px-3 py-1 bg-purple-500 text-slate-900 font-bold text-xs rounded-md
                                    disabled:bg-slate-600">
                                        {isEnhancing ? <LoadingSpinner /> : <SparklesIcon />} AI Enhance
                                    </button>
                                    <button onClick={() => navigator.clipboard.writeText(activeSnippet.code)}
                                    className="px-3 py-1 bg-slate-700 text-xs rounded-md">Copy Code</button>
                                </div>
                            </div>
                            <textarea
                                value={activeSnippet.code}
                                onChange={handleCodeChange}
                                className="flex-grow p-4 bg-slate-900 border border-slate-700 rounded-md resize-
                                none font-mono text-sm text-cyan-300 focus:ring-2 focus:ring-cyan-500
                                focus:outline-none"
                            />
                        </>
                    ) : (
                        <div className="flex-grow flex items-center justify-center bg-slate-900 rounded-
                        lg text-slate-500">
                            Select a snippet or create a new one.
                        </div>
                    )}
                </main>
            </div>
        </div>
    );
};


// ===== SvgPathEditor.tsx =====

import React, { useState } from 'react';
import { CodeBracketSquareIcon } from '../icons/FeatureIcons.tsx';

const initialPath = "M10 80 Q 52.5 10, 95 80 T 180 80";

export const SvgPathEditor: React.FC = () => {
    const [pathData, setPathData] = useState(initialPath);

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8">
            <header className="mb-6">
                <h1 className="text-3xl font-bold text-slate-100 flex items-center">
                    <CodeBracketSquareIcon />
                    <span className="ml-3">SVG Path Editor</span>
                </h1>
```

```
                <p className="text-slate-400 mt-1">Visually create and manipulate SVG path
                data.</p>
            </header>
            <div className="flex-grow grid grid-cols-1 lg:grid-cols-2 gap-6 h-full overflow-
            hidden">
                <div className="flex flex-col h-full">
                    <label htmlFor="path-input" className="text-sm font-medium text-slate-400
                    mb-2">Path Data (d attribute)</label>
                    <textarea
                        id="path-input"
                        value={pathData}
                        onChange={(e) => setPathData(e.target.value)}
                        className="flex-grow p-4 bg-slate-900 border border-slate-700 rounded-md resize-
                        none font-mono text-sm text-cyan-300"
                    />
                </div>
                <div className="flex flex-col h-full">
                    <label className="text-sm font-medium text-slate-400 mb-2">Live Preview</label>
                    <div className="flex-grow p-4 bg-slate-800/50 border-2 border-dashed border-
                    slate-700 rounded-md overflow-y-auto flex items-center justify-center">
                        <svg viewBox="0 0 200 100" className="w-full h-full">
                            <path d={pathData} stroke="#06b6d4" fill="transparent" strokeWidth="2" />
                        </svg>
                    </div>
                </div>
            </div>
        </div>
    );
};

// ===== ThemeDesigner.tsx =====

import React, { useState } from 'react';
import { SparklesIcon } from '../icons/FeatureIcons.tsx';

interface Theme {
    primary: string;
    background: string;
    surface: string;
    textPrimary: string;
    textSecondary: string;
}

const ColorInput: React.FC<{ label: string, value: string, onChange: (color:
string) => void }> = ({ label, value, onChange }) => (
    <div>
        <label className="flex items-center justify-between text-sm font-medium text-
        slate-400">
            {label}
            <span className="font-mono">{value}</span>
        </label>
        <input
            type="color"
            value={value}
            onChange={e => onChange(e.target.value)}
            className="w-full mt-1 h-8 rounded-md bg-transparent border border-slate-700
            cursor-pointer"
        />
    </div>
);

export const ThemeDesigner: React.FC = () => {
```

```
    const [theme, setTheme] = useState<Theme>({
        primary: '#06b6d4',
        background: '#0f172a',
        surface: '#1e293b',
        textPrimary: '#f1f5f9',
        textSecondary: '#94a3b8',
    });

    const handleThemeChange = (key: keyof Theme, value: string) => {
        setTheme(prev => ({ ...prev, [key]: value }));
    };

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8">
            <header className="mb-6">
                <h1 className="text-3xl font-bold text-slate-100 flex items-center">
                    <SparklesIcon />
                    <span className="ml-3">Theme Designer</span>
                </h1>
                <p className="text-slate-400 mt-1">Design and preview a custom color scheme for
                the UI.</p>
            </header>
            <div className="flex-grow grid grid-cols-1 lg:grid-cols-3 gap-6 min-h-0">
                <div className="lg:col-span-1 flex flex-col gap-4 bg-slate-800/50 p-6 rounded-
                lg">
                    <h3 className="text-xl font-bold">Colors</h3>
                    <ColorInput label="Primary" value={theme.primary} onChange={v =>
                    handleThemeChange('primary', v)} />
                    <ColorInput label="Background" value={theme.background} onChange={v =>
                    handleThemeChange('background', v)} />
                    <ColorInput label="Surface" value={theme.surface} onChange={v =>
                    handleThemeChange('surface', v)} />
                    <ColorInput label="Text Primary" value={theme.textPrimary} onChange={v =>
                    handleThemeChange('textPrimary', v)} />
                    <ColorInput label="Text Secondary" value={theme.textSecondary} onChange={v =>
                    handleThemeChange('textSecondary', v)} />
                </div>
                <div className="lg:col-span-2 rounded-lg p-8 transition-colors" style={{
                backgroundColor: theme.background, color: theme.textPrimary }}>
                     <h3 className="text-2xl font-bold mb-4" style={{ color: theme.textPrimary
                     }}>Live Preview</h3>
                     <div className="p-6 rounded-lg transition-colors" style={{ backgroundColor:
                     theme.surface }}>
                        <h4 className="text-lg font-bold">Sample Card</h4>
                        <p className="text-sm mt-1" style={{color: theme.textSecondary}}>This is a
                        sample card to demonstrate the theme colors.</p>
                        <button className="px-4 py-2 mt-4 rounded-md font-bold transition-colors"
                        style={{ backgroundColor: theme.primary, color: theme.background }}>
                            Primary Button
                        </button>
                     </div>
                </div>
            </div>
        </div>
    );
};

// ===== VisualGitTree.tsx =====


import React, { useState, useCallback } from 'react';
import { analyzeGitLogStream } from '../../services/geminiService.ts';
```

```
import { LoadingSpinner } from '../shared/LoadingSpinner.tsx';
import { MarkdownRenderer } from '../shared/MarkdownRenderer.tsx';
import { useFeatureState } from '../../hooks/useFeatureState.ts';
import { FeatureHeader } from '../shared/FeatureHeader.tsx';
import type { Feature } from '../../types.ts';

const exampleLog = `*   commit 3a4b5c6d7e8f9g0h1i2j3k4l5m6n7o8p9q0r
|\\  Merge: 1a2b3c4 2d3e4f5
| | Author: Dev One <dev.one@example.com>
| | Date:   Mon Jul 15 11:30:00 2024 -0400
| |
| |     Merge pull request #42 from feature/new-sidebar
| |
| |     feat: Implement collapsible sidebar navigation
| |
* | commit 2d3e4f5g6h7i8j9k0l1m2n3o4p5q6r7s8t9u
| | Author: Dev Two <dev.two@example.com>
| | Date:   Mon Jul 15 10:00:00 2024 -0400
| |
| |     feat: Add icons to sidebar items
| |
* | commit 1a2b3c4d5e6f7g8h9i0j1k2l3m4n5o6p7q8r
|/  Author: Dev One <dev.one@example.com>
|   Date:   Fri Jul 12 16:45:00 2024 -0400
|
|       fix: Correct user authentication bug
|`;


export const VisualGitTree: React.FC<{ feature: Feature }> = ({ feature }) => {
  const [state, setState, saveState] = useFeatureState('gitLogAnalyzer', {
  logInput: exampleLog, analysis: '' });
  const { logInput, analysis } = state;

  const [isLoading, setIsLoading] = useState(false);
  const [error, setError] = useState('');

  const handleAnalyze = useCallback(async () => {
    if (!logInput.trim()) {
      setError('Please paste git log output.');
      return;
    }
    setIsLoading(true);
    setError('');
    setState(s => ({...s, analysis: ''}));
    try {
        const stream = analyzeGitLogStream(logInput);
        let fullResponse = '';
        for await (const chunk of stream) {
            fullResponse += chunk;
            setState(s => ({...s, analysis: fullResponse}));
        }
    } catch (err) {
        const errorMessage = err instanceof Error ? err.message : 'An unknown error
        occurred.';
        setError(`Failed to analyze log: ${errorMessage}`);
    } finally {
        setIsLoading(false);
    }
  }, [logInput, setState]);

  return (
```

```
    <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8">
      <FeatureHeader feature={feature} onSaveState={saveState} />
      <div className="flex-grow grid grid-cols-1 lg:grid-cols-2 gap-6 h-full overflow-
      hidden">
        <div className="flex flex-col h-full">
            <label htmlFor="log-input" className="text-sm font-medium text-slate-400
            mb-2">Git Log Output</label>
            <textarea
                id="log-input"
                value={logInput}
                onChange={(e) => setState(s => ({...s, logInput: e.target.value}))}
                placeholder="Paste your git log output here..."
                className="flex-grow p-4 bg-slate-900 border border-slate-700 rounded-md resize-
                none font-mono text-sm text-slate-300"
            />
            <button
                onClick={handleAnalyze}
                disabled={isLoading}
                className="mt-4 w-full flex items-center justify-center px-6 py-3 bg-cyan-500
                text-slate-900 font-bold rounded-md hover:bg-cyan-400 disabled:bg-slate-600"
            >
                {isLoading ? <LoadingSpinner /> : 'Analyze Log'}
            </button>
        </div>
        <div className="flex flex-col h-full">
            <label className="text-sm font-medium text-slate-400 mb-2">AI Analysis</label>
            <div className="flex-grow p-4 bg-slate-800/50 border border-slate-700/50
            rounded-md overflow-y-auto">
                {isLoading && !analysis && <div className="flex items-center justify-center
                h-full"><LoadingSpinner /></div>}
                {error && <p className="text-red-400">{error}</p>}
                {analysis && <MarkdownRenderer content={analysis} />}
                {!isLoading && !analysis && !error && <div className="text-slate-500 h-full flex
                items-center justify-center">Analysis will appear here.</div>}
            </div>
        </div>
      </div>
    </div>
  );
};


// ===== WorkerThreadDebugger.tsx =====


import React, { useState, useCallback } from 'react';
import { analyzeConcurrencyStream } from '../../services/geminiService.ts';
import { LoadingSpinner } from '../shared/LoadingSpinner.tsx';
import { MarkdownRenderer } from '../shared/MarkdownRenderer.tsx';
import { useFeatureState } from '../../hooks/useFeatureState.ts';
import { FeatureHeader } from '../shared/FeatureHeader.tsx';
import type { Feature } from '../../types.ts';

const exampleCode = `// main.js
const worker = new Worker('worker.js');
let sharedCounter = 0;

worker.onmessage = function(e) {
  sharedCounter = e.data.counter;
  console.log('Counter updated by worker:', sharedCounter);
};
function incrementCounter() {
```

```
  sharedCounter++;
  worker.postMessage({ counter: sharedCounter });
  console.log('Counter updated by main:', sharedCounter);
}

setInterval(incrementCounter, 1000);

// worker.js
// let workerCounter = 0;
// onmessage = function(e) {
//   workerCounter = e.data.counter + 1;
//   postMessage({ counter: workerCounter });
// }
`;

export const WorkerThreadDebugger: React.FC<{ feature: Feature }> = ({ feature
}) => {
    const [state, setState, saveState] = useFeatureState('concurrencyAnalyzer', {
    codeInput: exampleCode, analysis: '' });
    const { codeInput, analysis } = state;

    const [isLoading, setIsLoading] = useState(false);
    const [error, setError] = useState('');

    const handleAnalyze = useCallback(async () => {
        if (!codeInput.trim()) {
            setError('Please paste some code to analyze.');
            return;
        }
        setIsLoading(true);
        setError('');
        setState(s => ({ ...s, analysis: '' }));
        try {
            const stream = analyzeConcurrencyStream(codeInput);
            let fullResponse = '';
            for await (const chunk of stream) {
                fullResponse += chunk;
                setState(s => ({...s, analysis: fullResponse}));
            }
        } catch (err) {
            const errorMessage = err instanceof Error ? err.message : 'An unknown error
            occurred.';
            setError(`Failed to analyze code: ${errorMessage}`);
        } finally {
            setIsLoading(false);
        }
    }, [codeInput, setState]);

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8">
            <FeatureHeader feature={feature} onSaveState={saveState} />
            <div className="flex-grow grid grid-cols-1 lg:grid-cols-2 gap-6 h-full overflow-
            hidden">
                <div className="flex flex-col h-full">
                    <label htmlFor="code-input" className="text-sm font-medium text-slate-400
                    mb-2">JavaScript Code</label>
                    <textarea
                        id="code-input"
                        value={codeInput}
                        onChange={(e) => setState(s => ({...s, codeInput: e.target.value}))}
                        placeholder="Paste your worker-related JS code here..."
                        className="flex-grow p-4 bg-slate-900 border border-slate-700 rounded-md resize-
```

```
                        none font-mono text-sm"
                    />
                    <button
                        onClick={handleAnalyze}
                        disabled={isLoading}
                        className="mt-4 w-full flex items-center justify-center px-6 py-3 bg-cyan-500
                        text-slate-900 font-bold rounded-md hover:bg-cyan-400 disabled:bg-slate-600"
                    >
                        {isLoading ? <LoadingSpinner /> : 'Analyze Code'}
                    </button>
                </div>
                <div className="flex flex-col h-full">
                    <label className="text-sm font-medium text-slate-400 mb-2">AI Analysis</label>
                    <div className="flex-grow p-4 bg-slate-800/50 border border-slate-700/50
                    rounded-md overflow-y-auto">
                        {isLoading && !analysis && <div className="flex items-center justify-center
                        h-full"><LoadingSpinner /></div>}
                        {error && <p className="text-red-400">{error}</p>}
                        {analysis && <MarkdownRenderer content={analysis} />}
                        {!isLoading && !analysis && !error && <div className="text-slate-500 h-full flex
                        items-center justify-center">Analysis will appear here.</div>}
                    </div>
                </div>
            </div>
        </div>
    );
};


// ===== LoadingSpinner.tsx =====


import React from 'react';

export const LoadingSpinner: React.FC = () => (
    <div className="flex items-center justify-center space-x-1" aria-
    label="Loading">
        <div className="w-2 h-2 rounded-full bg-current animate-pulse" style={{
        animationDelay: '0s' }}></div>
        <div className="w-2 h-2 rounded-full bg-current animate-pulse" style={{
        animationDelay: '0.2s' }}></div>
        <div className="w-2 h-2 rounded-full bg-current animate-pulse" style={{
        animationDelay: '0.4s' }}></div>
    </div>
);


// ===== AiCodeMigrator.tsx =====



import React, { useState, useCallback } from 'react';
import { migrateCodeStream } from '../../services/geminiService.ts';
import { LoadingSpinner } from '../shared/LoadingSpinner.tsx';
import { MarkdownRenderer } from '../shared/MarkdownRenderer.tsx';
import { useFeatureState } from '../../hooks/useFeatureState.ts';
import { FeatureHeader } from '../shared/FeatureHeader.tsx';
import type { Feature } from '../../types.ts';

const languages = ['SASS', 'CSS', 'JavaScript', 'TypeScript', 'Python', 'Go',
'React', 'Vue', 'Angular', 'Tailwind CSS'];
const exampleCode = `// SASS
```

```
$primary-color: #333;

body {
  color: $primary-color;
  font-family: sans-serif;
}`;

export const AiCodeMigrator: React.FC<{ feature: Feature }> = ({ feature }) => {
    const [state, setState, saveState] = useFeatureState('codeMigrator', {
        inputCode: exampleCode,
        outputCode: '',
        fromLang: 'SASS',
        toLang: 'CSS',
    });
    const { inputCode, outputCode, fromLang, toLang } = state;

    const [isLoading, setIsLoading] = useState<boolean>(false);
    const [error, setError] = useState<string>('');

    const handleMigrate = useCallback(async () => {
        if (!inputCode.trim()) {
            setError('Please enter some code to migrate.');
            return;
        }
        setIsLoading(true);
        setError('');
        setState(s => ({ ...s, outputCode: '' }));
        try {
            const stream = migrateCodeStream(inputCode, fromLang, toLang);
            let fullResponse = '';
            for await (const chunk of stream) {
                fullResponse += chunk;
                setState(s => ({...s, outputCode: fullResponse}));
            }
        } catch (err) {
            const errorMessage = err instanceof Error ? err.message : 'An unknown error
            occurred.';
            setError(`Failed to migrate code: ${errorMessage}`);
        } finally {
            setIsLoading(false);
        }
    }, [inputCode, fromLang, toLang, setState]);

    const LanguageSelector: React.FC<{ value: string, onChange: (val: string) =>
    void }> = ({ value, onChange }) => (
        <select value={value} onChange={e => onChange(e.target.value)} className="w-full
        px-3 py-2 rounded-md bg-slate-800 border border-slate-700">
            {languages.map(lang => <option key={lang} value={lang}>{lang}</option>)}
        </select>
    );

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8">
            <FeatureHeader feature={feature} onSaveState={saveState} />
            <div className="flex-grow flex flex-col min-h-0">
                <div className="grid grid-cols-1 lg:grid-cols-2 gap-6 flex-grow min-h-0">
                    <div className="flex flex-col h-full">
                        <div className="mb-2">
                            <label className="text-sm font-medium text-slate-400">From:</label>
                            <LanguageSelector value={fromLang} onChange={val => setState(s => ({...s,
                            fromLang: val}))} />
                        </div>
```

```
                        <textarea
                            value={inputCode}
                            onChange={(e) => setState(s => ({...s, inputCode: e.target.value}))}
                            placeholder="Paste your source code here..."
                            className="flex-grow p-4 bg-slate-900 border border-slate-700 rounded-md resize-
                            none font-mono text-sm"
                        />
                    </div>
                    <div className="flex flex-col h-full">
                        <div className="mb-2">
                            <label className="text-sm font-medium text-slate-400">To:</label>
                            <LanguageSelector value={toLang} onChange={val => setState(s => ({...s, toLang:
                            val}))} />
                        </div>
                        <div className="flex-grow p-1 bg-slate-800/50 border border-slate-700/50
                        rounded-md overflow-y-auto">
                           {isLoading && <div className="flex items-center justify-center
                           h-full"><LoadingSpinner /></div>}
                            {error && <p className="p-4 text-red-400">{error}</p>}
                            {outputCode && !isLoading && <MarkdownRenderer content={outputCode} />}
                            {!isLoading && !outputCode && !error && <div className="text-slate-500 h-full
                            flex items-center justify-center">Migrated code will appear here.</div>}
                        </div>
                    </div>
                </div>
                 <button
                    onClick={handleMigrate}
                    disabled={isLoading}
                    className="mt-4 w-full max-w-sm mx-auto flex items-center justify-center px-6
                    py-3 bg-cyan-500 text-slate-900 font-bold rounded-md hover:bg-cyan-400
                    disabled:bg-slate-600"
                >
                    {isLoading ? <LoadingSpinner /> : 'Migrate Code'}
                </button>
            </div>
        </div>
    );
};


// ===== AiOracle.tsx =====

import React, { useState, useEffect, useRef } from 'react';
import { useGlobalState } from '../../contexts/GlobalStateContext.tsx';
import { startChat } from '../../services/geminiService.ts';
import { CpuChipIcon } from '../icons/FeatureIcons.tsx';
import { MarkdownRenderer } from '../shared/MarkdownRenderer.tsx';
import { LoadingSpinner } from '../shared/LoadingSpinner.tsx';

interface Message {
    role: 'user' | 'model';
    text: string;
}

export const AiOracle: React.FC = () => {
    const { state } = useGlobalState();
    const { generatedFiles } = state;
    const [chat, setChat] = useState<any>(null);
    const [messages, setMessages] = useState<Message[]>([]);
    const [input, setInput] = useState('');
    const [isLoading, setIsLoading] = useState(false);
    const messagesEndRef = useRef<HTMLDivElement>(null);
```

```
useEffect(() => {
    const fileContext = generatedFiles.length > 0
        ? "The user has just generated the following files:\n\n" + generatedFiles.map(f
        => `--- FILE: ${f.filePath} ---\n${f.content}`).join('\n\n')
        : "The user has not generated any files yet.";

    const systemInstruction = `You are the AI Oracle, a world-class senior software
    engineer. Your purpose is to help the user with the code they are working on.
    You have been provided with the context of files they have recently generated.
    Use this context to answer their questions, help them refactor, or add new
    features. Be concise and helpful. Start the conversation by introducing yourself
    and acknowledging the file context you have.`;

    const initialHistory = [{
        role: 'user',
        parts: [{ text: fileContext }]
    }];

    const chatInstance = startChat(systemInstruction, initialHistory);
    setChat(chatInstance);

    // Start the conversation
    const startConversation = async () => {
        setIsLoading(true);
        const initialPrompt = "Introduce yourself and acknowledge the file context you
        have.";
        let response = await chatInstance.sendMessageStream({ message: initialPrompt });
        let fullResponse = '';
        for await (const chunk of response) {
            fullResponse += chunk.text;
        }
        setMessages([{ role: 'model', text: fullResponse }]);
        setIsLoading(false);
    };

    startConversation();

}, [generatedFiles]);

const scrollToBottom = () => {
    messagesEndRef.current?.scrollIntoView({ behavior: "smooth" });
};

useEffect(scrollToBottom, [messages]);

const handleSendMessage = async (e: React.FormEvent) => {
    e.preventDefault();
    if (!input.trim() || isLoading || !chat) return;

    const userMessage: Message = { role: 'user', text: input };
    setMessages(prev => [...prev, userMessage]);
    setInput('');
    setIsLoading(true);

    try {
        let fullResponse = '';
        const stream = await chat.sendMessageStream({ message: input });
        for await (const chunk of stream) {
            fullResponse += chunk.text;
            setMessages(prev => {
                const last = prev[prev.length - 1];
                if (last.role === 'model') {
```

```
                    return [...prev.slice(0, -1), { role: 'model', text: fullResponse }];
                }
                return [...prev, { role: 'model', text: fullResponse }];
            });
        }
    } catch (error) {
        console.error("Chat error:", error);
        setMessages(prev => [...prev, { role: 'model', text: 'Sorry, I encountered an
        error.' }]);
    } finally {
        setIsLoading(false);
    }
};

return (
    <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8">
        <header className="mb-6">
            <h1 className="text-3xl font-bold text-slate-100 flex items-center">
                <CpuChipIcon />
                <span className="ml-3">AI Oracle</span>
            </h1>
            <p className="text-slate-400 mt-1">Chat with an AI that has context of your
            generated files.</p>
        </header>
        <div className="flex-grow flex flex-col bg-slate-800/50 rounded-lg border
        border-slate-700/50 overflow-hidden">
            <div className="flex-grow p-4 space-y-4 overflow-y-auto">
                {messages.map((msg, index) => (
                    <div key={index} className={`flex ${msg.role === 'user' ? 'justify-end' :
                    'justify-start'}`}>
                        <div className={`max-w-xl p-3 rounded-lg ${msg.role === 'user' ? 'bg-cyan-600' :
                        'bg-slate-700'}`}>
                            <MarkdownRenderer content={msg.text} />
                        </div>
                    </div>
                ))}
                {isLoading && messages[messages.length-1].role === 'user' && (
                     <div className="flex justify-start">
                        <div className="max-w-xl p-3 rounded-lg bg-slate-700">
                            <LoadingSpinner />
                        </div>
                    </div>
                )}
                <div ref={messagesEndRef} />
            </div>
            <form onSubmit={handleSendMessage} className="p-4 border-t border-slate-700/50
            flex gap-2">
                <input
                    type="text"
                    value={input}
                    onChange={e => setInput(e.target.value)}
                    placeholder="Ask the Oracle to refactor, explain, or add to the code..."
                    className="flex-grow px-4 py-2 rounded-md bg-slate-800 border border-slate-600"
                    disabled={isLoading}
                />
                <button type="submit" disabled={isLoading || !input.trim()} className="px-6 py-2
                bg-cyan-500 text-slate-900 font-bold rounded-md disabled:bg-slate-600">
                    Send
                </button>
            </form>
        </div>
    </div>
```

```
    );
};


// ===== AiContextProtocol.tsx =====

import React, { useState, useCallback } from 'react';
import { callMcpModel } from '../../services/geminiService.ts';
import { CpuChipIcon } from '../icons/FeatureIcons.tsx';
import { LoadingSpinner } from '../shared/LoadingSpinner.tsx';
import { MarkdownRenderer } from '../shared/MarkdownRenderer.tsx';

const exampleContext = `DOCUMENTATION FOR "PIXELATOR" LIBRARY v1.2

FUNCTIONS:
- pixelator.createCanvas(width: number, height: number): Canvas
  - Creates a new canvas object.
- pixelator.drawRect(canvas: Canvas, x: number, y: number, width: number,
height: number, color: string): void
  - Draws a rectangle. Color format is hex, e.g., "#FF0000".
- pixelator.addText(canvas: Canvas, x: number, y: number, text: string, font:
string, size: number): void
  - Adds text to the canvas. Font is a string like "Arial".
- pixelator.export(canvas: Canvas): string
  - Returns a base64 encoded PNG of the canvas.
`;

export const AiContextProtocol: React.FC = () => {
    const [context, setContext] = useState(exampleContext);
    const [prompt, setPrompt] = useState("Write code to create a 200x100 canvas,
    draw a blue rectangle at 10,10, and add the text 'Hello' in 16px Arial.");
    const [response, setResponse] = useState('');
    const [isLoading, setIsLoading] = useState(false);
    const [error, setError] = useState('');

    const handleSubmit = useCallback(async () => {
        if (!context.trim() || !prompt.trim()) {
            setError('Please provide both a context and a prompt.');
            return;
        }
        setIsLoading(true);
        setError('');
        setResponse('');
        try {
            const result = await callMcpModel(context, prompt);
            setResponse(result.text);
        } catch (err) {
            const errorMessage = err instanceof Error ? err.message : 'An unknown error
            occurred.';
            setError(`Failed to get response: ${errorMessage}`);
        } finally {
            setIsLoading(false);
        }
    }, [context, prompt]);

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8">
            <header className="mb-6">
                <h1 className="text-3xl font-bold text-slate-100 flex items-center">
                    <CpuChipIcon />
                    <span className="ml-3">AI Context Protocol (MCP)</span>
                </h1>
```

```
                <p className="text-slate-400 mt-1">Ground AI responses with a large context like
                API docs or a codebase.</p>
            </header>
            <div className="flex-grow grid grid-cols-1 lg:grid-cols-2 gap-6 h-full overflow-
            hidden">
                <div className="flex flex-col h-full">
                    <label htmlFor="context-input" className="text-sm font-medium text-slate-400
                    mb-2">Context (e.g., API Documentation)</label>
                    <textarea
                        id="context-input"
                        value={context}
                        onChange={(e) => setContext(e.target.value)}
                        className="flex-grow p-4 bg-slate-900 border border-slate-700 rounded-md resize-
                        none font-mono text-sm"
                    />
                     <div className="mt-4">
                        <label htmlFor="prompt-input" className="text-sm font-medium text-slate-400
                        mb-2">Prompt</label>
                        <textarea
                            id="prompt-input"
                            value={prompt}
                            onChange={(e) => setPrompt(e.target.value)}
                            className="w-full h-24 p-2 bg-slate-900 border border-slate-700 rounded-md
                            resize-none text-sm"
                        />
                    </div>
                    <button
                        onClick={handleSubmit}
                        disabled={isLoading}
                        className="mt-4 w-full flex items-center justify-center px-6 py-3 bg-cyan-500
                        text-slate-900 font-bold rounded-md hover:bg-cyan-400 disabled:bg-slate-600"
                    >
                        {isLoading ? <LoadingSpinner /> : 'Ask with Context'}
                    </button>
                </div>
                <div className="flex flex-col h-full">
                    <label className="text-sm font-medium text-slate-400 mb-2">AI Response</label>
                    <div className="flex-grow p-4 bg-slate-800/50 border border-slate-700/50
                    rounded-md overflow-y-auto">
                        {isLoading && <div className="flex items-center justify-center
                        h-full"><LoadingSpinner /></div>}
                        {error && <p className="text-red-400">{error}</p>}
                        {response && !isLoading && <MarkdownRenderer content={response} />}
                        {!isLoading && !response && !error && <div className="text-slate-500 h-full flex
                        items-center justify-center">The grounded response will appear here.</div>}
                    </div>
                </div>
            </div>
        </div>
    );
};


// ===== AiCodeExplainer.tsx =====

import React, { useState, useCallback, useEffect, useMemo, useRef } from
'react';
import { explainCodeStructured } from '../../services/index.ts';
import type { StructuredExplanation } from '../../types.ts';
import { CpuChipIcon } from '../icons.tsx';
import { MarkdownRenderer, LoadingSpinner } from '../shared/index.tsx';
const exampleCode = `const bubbleSort = (arr) => {
```

```
  for (let i = 0; i < arr.length; i++) {
    for (let j = 0; j < arr.length - i - 1; j++) {
      if (arr[j] > arr[j + 1]) {
        [arr[j], arr[j + 1]] = [arr[j + 1], arr[j]];
      }
    }
  }
  return arr;
};`;

type ExplanationTab = 'summary' | 'lineByLine' | 'complexity' | 'suggestions';

const simpleSyntaxHighlight = (code: string) => {
    const escapedCode = code
        .replace(/&/g, '&amp;')
        .replace(/</g, '&lt;')
        .replace(/>/g, '&gt;');

    return escapedCode
        .replace(/\b(const|let|var|function|return|if|for|=>|import|from|export|default)
        \b/g, '<span class="text-indigo-600 font-semibold">$1</span>')
        .replace(/(\`|'|")(.*?)(\`|'|")/g, '<span class="text-
        emerald-700">$1$2$3</span>')
        .replace(/(\/\/.*)/g, '<span class="text-gray-500 italic">$1</span>')
        .replace(/(\{|\}|\(|\)|\[|\])/g, '<span class="text-gray-600">$1</span>');
};

export const AiCodeExplainer: React.FC<{ initialCode?: string }> = ({
initialCode }) => {
    const [code, setCode] = useState<string>(initialCode || exampleCode);
    const [explanation, setExplanation] = useState<StructuredExplanation |
    null>(null);
    const [isLoading, setIsLoading] = useState<boolean>(false);
    const [error, setError] = useState<string>('');
    const [activeTab, setActiveTab] = useState<ExplanationTab>('summary');
    const textareaRef = useRef<HTMLTextAreaElement>(null);
    const preRef = useRef<HTMLPreElement>(null);

    const handleExplain = useCallback(async (codeToExplain: string) => {
        if (!codeToExplain.trim()) {
            setError('Please enter some code to explain.');
            return;
        }
        setIsLoading(true);
        setError('');
        setExplanation(null);
        setActiveTab('summary');
        try {
            const result = await explainCodeStructured(codeToExplain);
            setExplanation(result);
        } catch (err) {
            const errorMessage = err instanceof Error ? err.message : 'An unknown error
            occurred.';
            setError(`Failed to get explanation: ${errorMessage}`);
        } finally {
            setIsLoading(false);
        }
    }, []);

    useEffect(() => {
        if (initialCode) {
            setCode(initialCode);
```

```
        handleExplain(initialCode);
    }
}, [initialCode, handleExplain]);

const handleScroll = () => {
    if (preRef.current && textareaRef.current) {
        preRef.current.scrollTop = textareaRef.current.scrollTop;
        preRef.current.scrollLeft = textareaRef.current.scrollLeft;
    }
};

const highlightedCode = useMemo(() => simpleSyntaxHighlight(code), [code]);

const renderTabContent = () => {
    if (!explanation) return null;
    switch(activeTab) {
        case 'summary':
            return <MarkdownRenderer content={explanation.summary} />;
        case 'lineByLine':
            return (
                <div className="space-y-3">
                    {explanation.lineByLine.map((item, index) => (
                        <div key={index} className="p-3 bg-background rounded-md border border-border">
                            <p className="font-mono text-xs text-primary mb-1">Lines: {item.lines}</p>
                            <p className="text-sm">{item.explanation}</p>
                        </div>
                    ))}
                </div>
            );
        case 'complexity':
            return (
                <div>
                    <p><strong>Time Complexity:</strong> <span className="font-mono text-
                    amber-600">{explanation.complexity.time}</span></p>
                    <p><strong>Space Complexity:</strong> <span className="font-mono text-
                    amber-600">{explanation.complexity.space}</span></p>
                </div>
            );
        case 'suggestions':
            return (
                 <ul className="list-disc list-inside space-y-2">
                    {explanation.suggestions.map((item, index) => <li key={index}>{item}</li>)}
                </ul>
            );
    }
}

return (
    <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
        <header className="mb-6 flex-shrink-0">
            <h1 className="text-3xl font-bold flex items-center">
                <CpuChipIcon />
                <span className="ml-3">AI Code Explainer</span>
            </h1>
            <p className="text-text-secondary mt-1">Get a detailed, structured analysis of
            any code snippet.</p>
        </header>
        <div className="flex-grow flex flex-col lg:flex-row gap-6 min-h-0">

            {/* Left Column: Code Input */}
            <div className="flex flex-col min-h-0 lg:w-1/2">
                <label htmlFor="code-input" className="text-sm font-medium text-text-secondary
```

```
            mb-2">Your Code</label>
            <div className="relative flex-grow bg-surface border border-border rounded-md
            focus-within:ring-2 focus-within:ring-primary overflow-hidden">
                <textarea
                    ref={textareaRef}
                    id="code-input"
                    value={code}
                    onChange={(e) => setCode(e.target.value)}
                    onScroll={handleScroll}
                    placeholder="Paste your code here..."
                    spellCheck="false"
                    className="absolute inset-0 w-full h-full p-4 bg-transparent resize-none font-
                    mono text-sm text-transparent caret-primary outline-none z-10"
                />
                <pre
                    ref={preRef}
                    aria-hidden="true"
                    className="absolute inset-0 w-full h-full p-4 font-mono text-sm text-text-
                    primary pointer-events-none z-0 whitespace-pre-wrap overflow-auto no-scrollbar"
                    dangerouslySetInnerHTML={{ __html: highlightedCode + '\n' }}
                />
            </div>
            <div className="mt-4 flex-shrink-0">
                <button
                    onClick={() => handleExplain(code)}
                    disabled={isLoading}
                    className="btn-primary w-full flex items-center justify-center px-6 py-3"
                >
                    {isLoading ? <LoadingSpinner/> : 'Analyze Code'}
                </button>
            </div>
        </div>

        {/* Right Column: AI Analysis */}
        <div className="flex flex-col min-h-0 lg:w-1/2">
            <label className="text-sm font-medium text-text-secondary mb-2">AI
            Analysis</label>
            <div className="relative flex-grow flex flex-col bg-surface border border-border
            rounded-md overflow-hidden">
                <div className="flex-shrink-0 flex border-b border-border">
                   {(['summary', 'lineByLine', 'complexity', 'suggestions'] as
                   ExplanationTab[]).map(tab => (
                       <button key={tab} onClick={() => setActiveTab(tab)} disabled={!explanation}
                        className={`px-4 py-2 text-sm font-medium capitalize transition-colors
                        ${activeTab === tab ? 'bg-background text-primary font-semibold' : 'text-text-
                        secondary hover:bg-gray-100 disabled:text-gray-400'}`}>
                           {tab.replace(/([A-Z])/g, ' $1')}
                       </button>
                   ))}
                </div>
                <div className="p-4 flex-grow overflow-y-auto">
                    {isLoading && <div className="flex items-center justify-center
                    h-full"><LoadingSpinner /></div>}
                    {error && <p className="text-red-500">{error}</p>}
                    {explanation && !isLoading && renderTabContent()}
                    {!isLoading && !explanation && !error && <div className="text-text-secondary
                    h-full flex items-center justify-center">The analysis will appear here.</div>}
                </div>
            </div>
        </div>
    </div>
</div>
```

```
    );
};

// ===== AiCodeMigrator_2.tsx =====



import React, { useState, useCallback, useEffect } from 'react';
import { migrateCodeStream } from '../../services/index.ts';
import { ArrowPathIcon } from '../icons.tsx';
import { LoadingSpinner } from '../shared/index.tsx';
import { MarkdownRenderer } from '../shared/index.tsx';

const languages = ['SASS', 'CSS', 'JavaScript', 'TypeScript', 'Python', 'Go',
'React', 'Vue', 'Angular', 'Tailwind CSS'];

const exampleCode = `// SASS
$primary-color: #333;

body {
  color: $primary-color;
  font-family: sans-serif;
}`;

export const AiCodeMigrator: React.FC<{ inputCode?: string, fromLang?: string,
toLang?: string }> = ({ inputCode: initialCode, fromLang: initialFrom, toLang:
initialTo }) => {
    const [inputCode, setInputCode] = useState<string>(initialCode || exampleCode);
    const [outputCode, setOutputCode] = useState<string>('');
    const [fromLang, setFromLang] = useState(initialFrom || 'SASS');
    const [toLang, setToLang] = useState(initialTo || 'CSS');
    const [isLoading, setIsLoading] = useState<boolean>(false);
    const [error, setError] = useState<string>('');

    const handleMigrate = useCallback(async (code: string, from: string, to: string)
    => {
        if (!code.trim()) {
            setError('Please enter some code to migrate.');
            return;
        }
        setIsLoading(true);
        setError('');
        setOutputCode('');
        try {
            const stream = migrateCodeStream(code, from, to);
            let fullResponse = '';
            for await (const chunk of stream) {
                fullResponse += chunk;
                setOutputCode(fullResponse);
            }
        } catch (err) {
            const errorMessage = err instanceof Error ? err.message : 'An unknown error
            occurred.';
            setError(`Failed to migrate code: ${errorMessage}`);
        } finally {
            setIsLoading(false);
        }
    }, []);

    useEffect(() => {
        if (initialCode && initialFrom && initialTo) {
            setInputCode(initialCode);
```

```
        setFromLang(initialFrom);
        setToLang(initialTo);
        handleMigrate(initialCode, initialFrom, initialTo);
    }
}, [initialCode, initialFrom, initialTo, handleMigrate]);

const LanguageSelector: React.FC<{ value: string, onChange: (val: string) =>
void }> = ({ value, onChange }) => (
    <select value={value} onChange={e => onChange(e.target.value)} className="w-full
    px-3 py-2 rounded-md bg-surface border border-border">
        {languages.map(lang => <option key={lang} value={lang}>{lang}</option>)}
    </select>
);

return (
    <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
        <header className="mb-6">
            <h1 className="text-3xl font-bold flex items-center"><ArrowPathIcon /><span
            className="ml-3">AI Code Migrator</span></h1>
            <p className="text-text-secondary mt-1">Translate code between languages,
            frameworks, and syntax styles.</p>
        </header>
        <div className="flex-grow flex flex-col min-h-0">
            <div className="grid grid-cols-1 lg:grid-cols-2 gap-6 flex-grow min-h-0">
                <div className="flex flex-col h-full">
                    <div className="mb-2">
                        <label className="text-sm font-medium text-text-secondary">From:</label>
                        <LanguageSelector value={fromLang} onChange={setFromLang} />
                    </div>
                    <textarea
                        value={inputCode}
                        onChange={(e) => setInputCode(e.target.value)}
                        placeholder="Paste your source code here..."
                        className="flex-grow p-4 bg-surface border border-border rounded-md resize-none
                        font-mono text-sm"
                    />
                </div>
                <div className="flex flex-col h-full">
                    <div className="mb-2">
                        <label className="text-sm font-medium text-text-secondary">To:</label>
                        <LanguageSelector value={toLang} onChange={setToLang} />
                    </div>
                    <div className="flex-grow p-1 bg-background border border-border rounded-md
                    overflow-y-auto">
                       {isLoading && <div className="flex items-center justify-center
                       h-full"><LoadingSpinner /></div>}
                        {error && <p className="p-4 text-red-500">{error}</p>}
                        {outputCode && !isLoading && <MarkdownRenderer content={outputCode} />}
                        {!isLoading && !outputCode && !error && <div className="text-text-secondary
                        h-full flex items-center justify-center">Migrated code will appear here.</div>}
                    </div>
                </div>
            </div>
             <button
                onClick={() => handleMigrate(inputCode, fromLang, toLang)}
                disabled={isLoading}
                className="btn-primary mt-4 w-full max-w-sm mx-auto flex items-center justify-
                center px-6 py-3"
            >
                {isLoading ? <LoadingSpinner /> : 'Migrate Code'}
            </button>
        </div>
```

```
        </div>
    );
};

// ===== AiCodingChallenge.tsx =====

import React, { useState, useCallback, useEffect } from 'react';
import { generateCodingChallengeStream } from '../../services/index.ts';
import { BeakerIcon } from '../icons.tsx';
import { LoadingSpinner } from '../shared/index.tsx';
import { MarkdownRenderer } from '../shared/index.tsx';

export const AiCodingChallenge: React.FC = () => {
    const [challenge, setChallenge] = useState<string>('');
    const [isLoading, setIsLoading] = useState<boolean>(false);
    const [error, setError] = useState<string>('');

    const handleGenerate = useCallback(async () => {
        setIsLoading(true);
        setError('');
        setChallenge('');
        try {
            const stream = generateCodingChallengeStream(null);
            let fullResponse = '';
            for await (const chunk of stream) {
                fullResponse += chunk;
                setChallenge(fullResponse);
            }
        } catch (err) {
            const errorMessage = err instanceof Error ? err.message : 'An unknown error
            occurred.';
            setError(`Failed to generate challenge: ${errorMessage}`);
        } finally {
            setIsLoading(false);
        }
    }, []);

    useEffect(() => {
        // Generate a challenge on initial load for a better user experience
        handleGenerate();
    // eslint-disable-next-line react-hooks/exhaustive-deps
    }, []);

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
            <header className="mb-6 flex justify-between items-center">
                <div>
                    <h1 className="text-3xl font-bold flex items-center">
                        <BeakerIcon />
                        <span className="ml-3">AI Coding Challenge Generator</span>
                    </h1>
                    <p className="text-text-secondary mt-1">Generate a unique coding problem to test
                    your skills.</p>
                </div>
                <button
                    onClick={handleGenerate}
                    disabled={isLoading}
                    className="btn-primary flex items-center justify-center px-6 py-3"
                >
                    {isLoading ? <LoadingSpinner /> : 'Generate New Challenge'}
                </button>
            </header>
```

```
            <div className="flex-grow p-4 bg-surface border border-border rounded-md
            overflow-y-auto">
                {isLoading && (
                     <div className="flex items-center justify-center h-full">
                        <LoadingSpinner />
                     </div>
                )}
                {error && <p className="text-red-500">{error}</p>}
                {challenge && !isLoading && (
                    <MarkdownRenderer content={challenge} />
                )}
                 {!isLoading && !challenge && !error && (
                    <div className="text-text-secondary h-full flex items-center justify-center">
                        Click "Generate New Challenge" to start.
                    </div>
                )}
            </div>
        </div>
    );
};

// ===== AiCommandCenter.tsx =====

import React, { useState, useCallback } from 'react';
import { Type, FunctionDeclaration } from "@google/genai";
import { getInferenceFunction, CommandResponse, FEATURE_TAXONOMY, logError }
from '../../services/index.ts';
import { useGlobalState } from '../../contexts/GlobalStateContext.tsx';
import { CommandLineIcon } from '../icons.tsx';
import { LoadingSpinner } from '../shared/index.tsx';
import { ALL_FEATURES } from './index.ts';

const functionDeclarations: FunctionDeclaration[] = [
    {
        name: 'navigateTo',
        description: 'Navigates to a specific feature page.',
        parameters: {
            type: Type.OBJECT,
            properties: {
                featureId: {
                    type: Type.STRING,
                    description: 'The ID of the feature to navigate to.',
                    enum: ALL_FEATURES.map(f => f.id)
                },
            },
            required: ['featureId'],
        },
    },
    {
        name: 'runFeatureWithInput',
        description: 'Navigates to a feature and passes initial data to it.',
        parameters: {
            type: Type.OBJECT,
            properties: {
                 featureId: {
                    type: Type.STRING,
                    description: 'The ID of the feature to run.',
                    enum: ALL_FEATURES.map(f => f.id)
                },
                props: {
                    type: Type.OBJECT,
                    description: 'An object containing the initial properties for the feature, based
```

```
                    on its required inputs.',
                    properties: {
                        initialCode: { type: Type.STRING },
                        initialPrompt: { type: Type.STRING },
                        beforeCode: { type: Type.STRING },
                        afterCode: { type: Type.STRING },
                        logInput: { type: Type.STRING },
                        diff: { type: Type.STRING },
                        codeInput: { type: Type.STRING },
                        jsonInput: { type: Type.STRING },
                    }
                }
            },
            required: ['featureId', 'props']
        }
    }
];

const knowledgeBase = FEATURE_TAXONOMY.map(f => `- ${f.name} (${f.id}):
${f.description} Inputs: ${f.inputs}`).join('\n');

const ExamplePromptButton: React.FC<{ text: string, onClick: (text: string) =>
void }> = ({ text, onClick }) => (
    <button
        onClick={() => onClick(text)}
        className="px-3 py-1.5 bg-surface border border-border rounded-full text-xs
        hover:bg-gray-100 transition-colors"
    >
        {text}
    </button>
)

export const AiCommandCenter: React.FC = () => {
    const { dispatch } = useGlobalState();
    const [prompt, setPrompt] = useState('');
    const [isLoading, setIsLoading] = useState(false);
    const [lastResponse, setLastResponse] = useState('');

    const handleCommand = useCallback(async () => {
        if (!prompt.trim()) return;

        setIsLoading(true);
        setLastResponse('');

        try {
            const response: CommandResponse = await getInferenceFunction(prompt,
            functionDeclarations, knowledgeBase);

            if (response.functionCalls && response.functionCalls.length > 0) {
                const call = response.functionCalls[0];
                const { name, args } = call;

                setLastResponse(`Understood! Executing command: ${name}`);

                switch (name) {
                    case 'navigateTo':
                        dispatch({ type: 'SET_VIEW', payload: { view: args.featureId }});
                        break;
                    case 'runFeatureWithInput':
                         dispatch({ type: 'SET_VIEW', payload: { view: args.featureId, props: args.props
                         } });
                        break;
```

```
                default:
                    setLastResponse(`Unknown command: ${name}`);
            }
             setPrompt('');
        } else {
             setLastResponse(response.text);
        }

    } catch (err) {
        logError(err as Error, { prompt });
        setLastResponse(err instanceof Error ? err.message : 'An unknown error
        occurred.');
    } finally {
        setIsLoading(false);
    }
}, [prompt, dispatch]);

const handleKeyDown = (e: React.KeyboardEvent) => {
    if (e.key === 'Enter' && !e.shiftKey) {
        e.preventDefault();
        handleCommand();
    }
};

const handleExampleClick = (text: string) => {
    setPrompt(text);
}

return (
    <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
        <header className="mb-6 text-center">
            <h1 className="text-4xl font-extrabold tracking-tight flex items-center justify-
            center">
                <CommandLineIcon />
                <span className="ml-3">AI Command Center</span>
            </h1>
            <p className="mt-2 text-lg text-text-secondary">What would you like to do?</p>
        </header>

        <div className="flex-grow flex flex-col justify-end max-w-3xl w-full mx-auto">
            {lastResponse && (
                <div className="mb-4 p-4 bg-surface rounded-lg text-text-primary border border-
                border">
                    <p><strong>AI:</strong> {lastResponse}</p>
                </div>
            )}
             <div className="relative">
                <textarea
                    value={prompt}
                    onChange={e => setPrompt(e.target.value)}
                    onKeyDown={handleKeyDown}
                    disabled={isLoading}
                    placeholder='Try "explain this code: const a = 1;" or "open the theme designer"'
                    className="w-full p-4 pr-28 rounded-lg bg-surface border border-border
                    focus:ring-2 focus:ring-primary focus:outline-none resize-none shadow-sm"
                    rows={2}
                />
                <button
                    onClick={handleCommand}
                    disabled={isLoading}
                    className="btn-primary absolute right-3 top-1/2 -translate-y-1/2 px-4 py-2"
                >
```

```
                        {isLoading ? <LoadingSpinner/> : 'Send'}
                    </button>
                </div>
                 <div className="flex flex-wrap items-center justify-center gap-2 mt-4">
                    <ExamplePromptButton text="Open Theme Designer" onClick={handleExampleClick} />
                    <ExamplePromptButton text="Generate a commit for a bug fix"
                    onClick={handleExampleClick} />
                    <ExamplePromptButton text="Create a regex for email validation"
                    onClick={handleExampleClick} />
                </div>
                 <p className="text-xs text-text-secondary text-center mt-2">Press Enter to send,
                 Shift+Enter for new line.</p>
            </div>
        </div>
    );
};

// ===== AiCommitGenerator.tsx =====

import React, { useState, useCallback, useEffect } from 'react';
import { generateCommitMessageStream, downloadFile } from
'../../services/index.ts';
import { GitBranchIcon, ArrowDownTrayIcon } from '../icons.tsx';
import { LoadingSpinner } from '../shared/index.tsx';

const exampleDiff = `diff --git a/src/components/Button.tsx
b/src/components/Button.tsx
index 1b2c3d4..5e6f7g8 100644
--- a/src/components/Button.tsx
+++ b/src/components/Button.tsx
@@ -1,7 +1,7 @@
 import React from 'react';

 interface ButtonProps {
-  text: string;
+  label: string;
   onClick: () => void;
 }
`;

export const AiCommitGenerator: React.FC<{ diff?: string }> = ({ diff:
initialDiff }) => {
    const [diff, setDiff] = useState<string>(initialDiff || exampleDiff);
    const [message, setMessage] = useState<string>('');
    const [isLoading, setIsLoading] = useState<boolean>(false);
    const [error, setError] = useState<string>('');

    const handleGenerate = useCallback(async (diffToAnalyze: string) => {
        if (!diffToAnalyze.trim()) {
            setError('Please paste a diff to generate a message.');
            return;
        }
        setIsLoading(true);
        setError('');
        setMessage('');
        try {
            const stream = generateCommitMessageStream(diffToAnalyze);
            let fullResponse = '';
            for await (const chunk of stream) {
                fullResponse += chunk;
                setMessage(fullResponse);
            }
```

```
    } catch (err) {
        const errorMessage = err instanceof Error ? err.message : 'An unknown error
        occurred.';
        setError(`Failed to generate message: ${errorMessage}`);
    } finally {
        setIsLoading(false);
    }
}, []);

useEffect(() => {
    if (initialDiff) {
        setDiff(initialDiff);
        handleGenerate(initialDiff);
    }
}, [initialDiff, handleGenerate]);

const handleCopy = () => {
    navigator.clipboard.writeText(message);
};

const handleDownload = () => {
    downloadFile(message, 'commit_message.txt', 'text/plain');
};

return (
    <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
        <header className="mb-6">
            <h1 className="text-3xl font-bold flex items-center">
                <GitBranchIcon />
                <span className="ml-3">AI Commit Message Generator</span>
            </h1>
            <p className="text-text-secondary mt-1">Paste your diff and let Gemini craft the
            perfect commit message.</p>
        </header>
        <div className="flex-grow flex flex-col gap-4 min-h-0">
            <div className="flex flex-col flex-1 min-h-0">
                <label htmlFor="diff-input" className="text-sm font-medium text-text-secondary
                mb-2">Git Diff</label>
                <textarea
                    id="diff-input"
                    value={diff}
                    onChange={(e) => setDiff(e.target.value)}
                    placeholder="Paste your git diff here..."
                    className="flex-grow p-4 bg-surface border border-border rounded-md resize-none
                    font-mono text-sm text-text-primary focus:ring-2 focus:ring-primary
                    focus:outline-none"
                />
            </div>
            <div className="flex-shrink-0">
                <button
                    onClick={() => handleGenerate(diff)}
                    disabled={isLoading}
                    className="btn-primary w-full max-w-xs mx-auto flex items-center justify-center
                    px-6 py-3"
                >
                    {isLoading ? <LoadingSpinner /> : 'Generate Commit Message'}
                </button>
            </div>
            <div className="flex flex-col flex-1 min-h-0">
                <div className="flex justify-between items-center mb-2">
                    <label className="text-sm font-medium text-text-secondary">Generated
                    Message</label>
```

```
                        {message && !isLoading && (
                            <div className="flex items-center gap-2">
                                <button onClick={handleCopy} className="px-3 py-1 bg-gray-100 text-xs rounded-md
                                hover:bg-gray-200">Copy</button>
                                <button onClick={handleDownload} className="flex items-center gap-1 px-3 py-1
                                bg-gray-100 text-xs rounded-md hover:bg-gray-200">
                                    <ArrowDownTrayIcon className="w-4 h-4" /> Download
                                </button>
                            </div>
                        )}
                    </div>
                    <div className="relative flex-grow p-4 bg-surface border border-border rounded-
                    md overflow-y-auto">
                        {isLoading && (
                             <div className="flex items-center justify-center h-full">
                                <LoadingSpinner />
                             </div>
                        )}
                        {error && <p className="text-red-500">{error}</p>}
                        {message && !isLoading && (
                           <pre className="whitespace-pre-wrap font-sans text-text-primary">{message}</pre>
                        )}
                         {!isLoading && !message && !error && (
                            <div className="text-text-secondary h-full flex items-center justify-center">
                                The commit message will appear here.
                            </div>
                        )}
                    </div>
                </div>
            </div>
        </div>
    );
};

// ===== AiFeatureBuilder_2.tsx =====

import React, { useState, useCallback, useEffect } from 'react';
import type { GeneratedFile } from '../../types.ts';
import { generateFeature, generateUnitTestsStream, generateCommitMessageStream,
saveFile, getAllFiles, clearAllFiles } from '../../services/index.ts';
import { CpuChipIcon, DocumentTextIcon, BeakerIcon, GitBranchIcon } from
'../icons.tsx';
import { LoadingSpinner, MarkdownRenderer } from '../shared/index.tsx';

type ActiveTab = 'CODE' | 'TESTS' | 'COMMIT';

export const AiFeatureBuilder: React.FC = () => {
    const [prompt, setPrompt] = useState<string>('A simple "Hello World" React
    component with a button that shows an alert.');
    const [generatedFiles, setGeneratedFiles] = useState<GeneratedFile[]>([]);
    const [unitTests, setUnitTests] = useState<string>('');
    const [commitMessage, setCommitMessage] = useState<string>('');
    const [selectedFile, setSelectedFile] = useState<GeneratedFile | null>(null);
    const [isLoading, setIsLoading] = useState<boolean>(false);
    const [error, setError] = useState<string>('');
    const [activeTab, setActiveTab] = useState<ActiveTab>('CODE');

    useEffect(() => {
        const loadFiles = async () => {
            const files = await getAllFiles();
            setGeneratedFiles(files);
            if (files.length > 0) setSelectedFile(files[0]);
```

```
    };
    loadFiles();
}, []);

const handleGenerate = useCallback(async () => {
    if (!prompt.trim()) { setError('Please enter a feature description.'); return; }
    setIsLoading(true);
    setError('');
    await clearAllFiles(); // Start fresh for each generation
    setGeneratedFiles([]);
    setUnitTests('');
    setCommitMessage('');
    setSelectedFile(null);
    setActiveTab('CODE');

    try {
        const resultFiles = await generateFeature(prompt);
        for (const file of resultFiles) { await saveFile(file); }
        setGeneratedFiles(resultFiles);

        if (resultFiles.length > 0) {
            const componentFile = resultFiles.find(f => f.filePath.endsWith('.tsx') ||
            f.filePath.endsWith('.jsx'));
            setSelectedFile(componentFile || resultFiles[0]);

            if (componentFile) {
                const testStream = generateUnitTestsStream(componentFile.content);
                let tests = '';
                for await (const chunk of testStream) { tests += chunk; setUnitTests(tests); }
            }

            const diffContext = resultFiles.map(f => `File:
            ${f.filePath}\n\n${f.content}`).join('\n---\n');
            const commitStream = generateCommitMessageStream(diffContext);
            let commit = '';
            for await (const chunk of commitStream) { commit += chunk;
            setCommitMessage(commit); }
        }
    } catch (err) {
        const errorMessage = err instanceof Error ? err.message : 'An unknown error
        occurred.';
        setError(`Failed to generate feature: ${errorMessage}`);
    } finally {
        setIsLoading(false);
    }
}, [prompt]);

const renderContent = () => {
    switch (activeTab) {
        case 'TESTS': return <MarkdownRenderer content={unitTests} />;
        case 'COMMIT': return <pre className="w-full h-full p-4 whitespace-pre-wrap
        font-sans text-sm text-text-primary">{commitMessage}</pre>;
        case 'CODE':
        default:
             return selectedFile ? <MarkdownRenderer
             content={`\`\`\`tsx\n${selectedFile.content}\n\`\`\``} /> : <div className="flex
             items-center justify-center h-full text-text-secondary">Select a file to view
             its content.</div>;
    }
}

return (
```

```
        <div className="h-full flex flex-col text-text-primary bg-surface">
            <header className="p-4 border-b border-border flex-shrink-0">
                <h1 className="text-xl font-bold flex items-center"><CpuChipIcon /><span
                className="ml-3">AI Feature Builder</span></h1>
            </header>

            <div className="flex-grow flex min-h-0">
                <aside className="w-64 bg-surface border-r border-border p-4 flex flex-col">
                    <h2 className="text-sm font-semibold text-text-secondary mb-2">Generated
                    Files</h2>
                    <div className="overflow-y-auto space-y-1">
                        {generatedFiles.map(file => (
                            <div key={file.filePath} onClick={() => { setSelectedFile(file);
                            setActiveTab('CODE'); }} className={`flex items-center space-x-2 p-2 rounded-md
                            cursor-pointer text-sm ${selectedFile?.filePath === file.filePath && activeTab
                            === 'CODE' ? 'bg-primary/10 text-primary' : 'hover:bg-gray-100'}`}>
                                <DocumentTextIcon /><span>{file.filePath.split('/').pop()}</span>
                            </div>
                        ))}
                    </div>
                </aside>

                <main className="flex-1 flex flex-col min-w-0">
                    <div className="flex-grow flex flex-col bg-background">
                         <div className="border-b border-border flex items-center bg-surface">
                            <button onClick={() => setActiveTab('CODE')} className={`flex items-center gap-2
                            px-4 py-2 text-sm ${activeTab === 'CODE' ? 'bg-background border-b-2 border-
                            primary text-text-primary' : 'text-text-secondary hover:bg-
                            gray-50'}`}><DocumentTextIcon /> Code</button>
                            {unitTests && <button onClick={() => setActiveTab('TESTS')} className={`flex
                            items-center gap-2 px-4 py-2 text-sm ${activeTab === 'TESTS' ? 'bg-background
                            border-b-2 border-primary text-text-primary' : 'text-text-secondary hover:bg-
                            gray-50'}`}><BeakerIcon /> Tests</button>}
                            {commitMessage && <button onClick={() => setActiveTab('COMMIT')}
                            className={`flex items-center gap-2 px-4 py-2 text-sm ${activeTab === 'COMMIT' ?
                            'bg-background border-b-2 border-primary text-text-primary' : 'text-text-
                            secondary hover:bg-gray-50'}`}><GitBranchIcon /> Commit</button>}
                        </div>
                        <div className="flex-grow p-2 overflow-auto">
                            {isLoading && !generatedFiles.length ? <div className="flex justify-center
                            items-center h-full"><LoadingSpinner/></div> : renderContent()}
                        </div>
                    </div>

                    <div className="flex-shrink-0 p-4 border-t border-border bg-surface">
                        <textarea value={prompt} onChange={(e) => setPrompt(e.target.value)}
                        placeholder="e.g., A user profile card with an avatar, name, and bio."
                        className="w-full p-2 bg-background border border-border rounded-md resize-none
                        text-sm h-20"/>
                         <button onClick={handleGenerate} disabled={isLoading} className="btn-primary
                         mt-2 w-full flex items-center justify-center gap-2 px-4 py-2">
                            {isLoading ? <><LoadingSpinner /> Generating...</> : 'Generate Feature'}
                        </button>
                         {error && <p className="text-red-600 text-xs mt-2 text-center">{error}</p>}
                    </div>
                </main>
            </div>
        </div>
    );
};

// ===== AiImageGenerator.tsx =====
```

```
import React, { useState, useCallback } from 'react';
import { generateImage } from '../../services/index.ts';
import { ImageGeneratorIcon, SparklesIcon } from '../icons.tsx';
import { LoadingSpinner } from '../shared/index.tsx';
import { ArrowDownTrayIcon } from '../icons.tsx';

const surprisePrompts = [
    'A majestic lion wearing a crown, painted in the style of Van Gogh.',
    'A futuristic cityscape on another planet with two moons in the sky.',
    'A cozy, magical library inside a giant tree.',
    'A surreal image of a ship sailing on a sea of clouds.',
    'An astronaut riding a space-themed bicycle on the moon.',
];

export const AiImageGenerator: React.FC = () => {
    const [prompt, setPrompt] = useState<string>('A photorealistic image of a
    futuristic city at sunset, with flying cars.');
    const [imageUrl, setImageUrl] = useState<string | null>(null);
    const [isLoading, setIsLoading] = useState<boolean>(false);
    const [error, setError] = useState<string>('');

    const handleGenerate = useCallback(async () => {
        if (!prompt.trim()) {
            setError('Please enter a prompt to generate an image.');
            return;
        }
        setIsLoading(true);
        setError('');
        setImageUrl(null);
        try {
            const resultUrl = await generateImage(prompt);
            setImageUrl(resultUrl);
        } catch (err) {
            const errorMessage = err instanceof Error ? err.message : 'An unknown error
            occurred.';
            setError(`Failed to generate image: ${errorMessage}`);
        } finally {
            setIsLoading(false);
        }
    }, [prompt]);

    const handleSurpriseMe = () => {
        const randomPrompt = surprisePrompts[Math.floor(Math.random() *
        surprisePrompts.length)];
        setPrompt(randomPrompt);
    };

    const handleDownload = () => {
        if (!imageUrl) return;
        const link = document.createElement('a');
        link.href = imageUrl;
        link.download = `${prompt.slice(0, 30).replace(/\s/g, '_')}.png`;
        document.body.appendChild(link);
        link.click();
        document.body.removeChild(link);
    }

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
            <header className="mb-6">
                <h1 className="text-3xl font-bold flex items-center">
                    <ImageGeneratorIcon />
```

```
        <span className="ml-3">AI Image Generator</span>
    </h1>
    <p className="text-text-secondary mt-1">Generate stunning, high-quality images
    from text prompts using Imagen 3.</p>
</header>

<div className="flex flex-col gap-4 mb-4">
    <label htmlFor="prompt-input" className="text-sm font-medium text-text-
    secondary">Your Prompt</label>
    <textarea
        id="prompt-input"
        value={prompt}
        onChange={(e) => setPrompt(e.target.value)}
        placeholder="e.g., A cute cat wearing a wizard hat"
        className="w-full p-3 rounded-md bg-surface border border-border focus:ring-2
        focus:ring-primary focus:outline-none resize-y"
        rows={3}
    />
    <div className="flex gap-2">
        <button
            onClick={handleGenerate}
            disabled={isLoading}
            className="btn-primary w-full flex items-center justify-center px-6 py-3"
        >
            {isLoading ? <LoadingSpinner /> : 'Generate Image'}
        </button>
        <button
            onClick={handleSurpriseMe}
            disabled={isLoading}
            className="px-4 py-3 bg-surface border border-border rounded-md hover:bg-
            gray-100 transition-colors"
            title="Surprise Me!"
        >
            <SparklesIcon />
        </button>
    </div>
</div>

<div className="flex-grow flex items-center justify-center bg-background
border-2 border-dashed border-border rounded-lg p-4 relative overflow-auto">
    {isLoading && <LoadingSpinner />}
    {error && <p className="text-red-500 text-center">{error}</p>}
    {imageUrl && !isLoading && (
        <>
            <img src={imageUrl} alt={prompt || "Generated by AI"} className="max-w-full max-
            h-full object-contain rounded-md shadow-lg" />
            <button
              onClick={handleDownload}
              className="absolute top-4 right-4 p-2 bg-black/30 text-white rounded-full
              hover:bg-black/50 backdrop-blur-sm"
              title="Download Image"
            >
                <ArrowDownTrayIcon />
            </button>
        </>
    )}
    {!isLoading && !imageUrl && !error && (
        <div className="text-center text-text-secondary">
            <p>Your generated image will appear here.</p>
        </div>
    )}
</div>
```

```
        </div>
    );
};

// ===== AiPullRequestAssistant.tsx =====

import React, { useState, useMemo, useCallback } from 'react';
import * as Diff from 'diff';
import { generatePrSummaryStructured } from '../../services/index.ts';
import type { StructuredPrSummary } from '../../types.ts';
import { AiPullRequestAssistantIcon } from '../icons.tsx';
import { LoadingSpinner } from '../shared/index.tsx';

const exampleBefore = `function Greeter(props) {
  return <h1>Hello, {props.name}!</h1>;
}`;
const exampleAfter = `function Greeter({ name, enthusiasmLevel = 1 }) {
  const punctuation = '!'.repeat(enthusiasmLevel);
  return <h1>Hello, {name}{punctuation}</h1>;
}`;

export const AiPullRequestAssistant: React.FC = () => {
    const [beforeCode, setBeforeCode] = useState<string>(exampleBefore);
    const [afterCode, setAfterCode] = useState<string>(exampleAfter);
    const [isLoading, setIsLoading] = useState<boolean>(false);
    const [error, setError] = useState<string>('');

    // Form state
    const [title, setTitle] = useState('');
    const [description, setDescription] = useState('');
    const [fromBranch, setFromBranch] = useState('feature/new-logic');
    const [toBranch, setToBranch] = useState('main');
    const [changeType, setChangeType] = useState('feat');
    const [relatedIssue, setRelatedIssue] = useState('');
    const [testingSteps, setTestingSteps] = useState('1. \n2. \n3.');

    const handleGenerateSummary = useCallback(async () => {
        if (!beforeCode.trim() && !afterCode.trim()) {
            setError('Please provide code to generate a summary.');
            return;
        }
        setIsLoading(true);
        setError('');

        try {
            const diff = Diff.createPatch('component.tsx', beforeCode, afterCode);
            const result: StructuredPrSummary = await generatePrSummaryStructured(diff);
            setTitle(result.title);
            setDescription(`${result.summary}\n\n**Key Changes:**\n${result.changes.map(c =>
            `- ${c}`).join('\n')}`);
        } catch (err) {
            const errorMessage = err instanceof Error ? err.message : 'An unknown error
            occurred.';
            setError(`Failed to generate summary: ${errorMessage}`);
        } finally {
            setIsLoading(false);
        }
    }, [beforeCode, afterCode]);

    const markdownPreview = useMemo(() => {
        return `
# ${changeType}: ${title}
```

```
${relatedIssue ? `\n**Closes:** ${relatedIssue}\n` : ''}
**Branch:** \`${fromBranch}\` -> \`${toBranch}\`

## Description
${description}

## Testing Steps
${testingSteps}
        `.trim();
    }, [title, description, fromBranch, toBranch, changeType, relatedIssue,
    testingSteps]);

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
            <header className="mb-6">
                <h1 className="text-3xl font-bold flex items-center">
                    <AiPullRequestAssistantIcon />
                    <span className="ml-3">AI Pull Request Assistant</span>
                </h1>
                <p className="text-text-secondary mt-1">Generate a PR summary from code changes
                and populate a full template.</p>
            </header>
            <div className="flex-grow grid grid-cols-1 lg:grid-cols-2 gap-6 min-h-0">
                {/* Left side: Inputs and Generator */}
                <div className="flex flex-col gap-4 min-h-0">
                    <div className="flex flex-col flex-1 min-h-0">
                        <label htmlFor="before-code" className="text-sm font-medium text-text-secondary
                        mb-2">Before</label>
                        <textarea id="before-code" value={beforeCode} onChange={e =>
                        setBeforeCode(e.target.value)} className="flex-grow p-4 bg-surface border
                        border-border rounded-md resize-none font-mono text-sm" />
                    </div>
                    <div className="flex flex-col flex-1 min-h-0">
                        <label htmlFor="after-code" className="text-sm font-medium text-text-secondary
                        mb-2">After</label>
                        <textarea id="after-code" value={afterCode} onChange={e =>
                        setAfterCode(e.target.value)} className="flex-grow p-4 bg-surface border border-
                        border rounded-md resize-none font-mono text-sm" />
                    </div>
                    <button onClick={handleGenerateSummary} disabled={isLoading} className="btn-
                    primary w-full flex items-center justify-center px-6 py-3">
                        {isLoading ? <LoadingSpinner /> : 'Generate Title & Description'}
                    </button>
                    {error && <p className="text-red-500 text-xs text-center">{error}</p>}
                </div>

                {/* Right side: Form and Preview */}
                <div className="flex flex-col gap-4 min-h-0">
                    <form className="flex flex-col gap-2 overflow-y-auto pr-2 bg-surface border
                    border-border p-4 rounded-lg h-1/2">
                        <div className="flex gap-2">
                            <div className="w-1/4"><label className="block text-xs">Type</label><select
                            value={changeType} onChange={e => setChangeType(e.target.value)}
                            className="w-full mt-1 p-1 rounded bg-background border border-border text-sm"><
                            option>feat</option><option>fix</option><option>chore</option><option>docs</opti
                            on><option>refactor</option></select></div>
                            <div className="w-3/4"><label className="block text-xs">Title</label><input
                            type="text" value={title} onChange={e => setTitle(e.target.value)}
                            className="w-full mt-1 p-1 rounded bg-background border border-border text-
                            sm"/></div>
                        </div>
                        <div><label className="block text-xs">Description</label><textarea
```

```
                        value={description} onChange={e => setDescription(e.target.value)}
                        className="w-full mt-1 p-1 rounded bg-background border border-border resize-y
                        h-24 text-sm"/></div>
                        <div><label className="block text-xs">Testing Steps</label><textarea
                        value={testingSteps} onChange={e => setTestingSteps(e.target.value)}
                        className="w-full mt-1 p-1 rounded bg-background border border-border resize-y
                        h-16 text-sm"/></div>
                    </form>
                    <div className="flex flex-col h-1/2">
                        <div className="flex justify-between items-center mb-2">
                            <label className="text-sm font-medium text-text-secondary">Markdown
                            Preview</label>
                            <button onClick={() => navigator.clipboard.writeText(markdownPreview)}
                            className="px-3 py-1 bg-gray-100 text-xs rounded-md hover:bg-gray-200">Copy
                            Markdown</button>
                        </div>
                        <div className="relative flex-grow"><pre className="w-full h-full bg-background
                        border border-border p-4 rounded-md text-sm overflow-auto whitespace-pre-
                        wrap">{markdownPreview}</pre></div>
                    </div>
                </div>
            </div>
        </div>
    );
};

// ===== AiStyleTransfer_2.tsx =====

import React, { useState, useCallback } from 'react';
import { transferCodeStyleStream } from '../../services/index.ts';
import { SparklesIcon } from '../icons.tsx';
import { LoadingSpinner } from '../shared/index.tsx';
import { MarkdownRenderer } from '../shared/index.tsx';

const exampleCode = `function my_func(x,y){return x+y;}`;
const exampleStyleGuide = `- Use camelCase for function names.
- Add a space after commas in argument lists.
- Use semicolons at the end of statements.`;

export const AiStyleTransfer: React.FC = () => {
    const [inputCode, setInputCode] = useState<string>(exampleCode);
    const [styleGuide, setStyleGuide] = useState<string>(exampleStyleGuide);
    const [outputCode, setOutputCode] = useState<string>('');
    const [isLoading, setIsLoading] = useState<boolean>(false);
    const [error, setError] = useState<string>('');

    const handleGenerate = useCallback(async () => {
        if (!inputCode.trim() || !styleGuide.trim()) {
            setError('Please provide both code and a style guide.');
            return;
        }
        setIsLoading(true);
        setError('');
        setOutputCode('');
        try {
            const stream = transferCodeStyleStream({ code: inputCode, styleGuide });
            let fullResponse = '';
            for await (const chunk of stream) {
                fullResponse += chunk;
                setOutputCode(fullResponse);
            }
        } catch (err) {
```

```
        const errorMessage = err instanceof Error ? err.message : 'An unknown error
        occurred.';
        setError(`Failed to transfer style: ${errorMessage}`);
    } finally {
        setIsLoading(false);
    }
}, [inputCode, styleGuide]);

return (
    <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
        <header className="mb-6">
            <h1 className="text-3xl font-bold flex items-center">
                <SparklesIcon />
                <span className="ml-3">AI Code Style Transfer</span>
            </h1>
            <p className="text-text-secondary mt-1">Rewrite code to match a specific style
            guide using AI.</p>
        </header>
        <div className="flex-grow flex flex-col gap-4 min-h-0">
            <div className="flex flex-col flex-1 min-h-0">
                <label htmlFor="input-code" className="text-sm font-medium text-text-secondary
                mb-2">Original Code</label>
                <textarea
                    id="input-code"
                    value={inputCode}
                    onChange={(e) => setInputCode(e.target.value)}
                    className="flex-grow p-4 bg-surface border border-border rounded-md resize-y
                    font-mono text-sm"
                />
            </div>
             <div className="flex flex-col flex-1 min-h-0">
                <label htmlFor="style-guide" className="text-sm font-medium text-text-secondary
                mb-2">Style Guide</label>
                <textarea
                    id="style-guide"
                    value={styleGuide}
                    onChange={(e) => setStyleGuide(e.target.value)}
                    className="flex-grow p-4 bg-surface border border-border rounded-md resize-y
                    font-mono text-sm"
                />
            </div>
             <div className="flex-shrink-0">
                <button
                    onClick={handleGenerate}
                    disabled={isLoading}
                    className="btn-primary w-full max-w-xs mx-auto flex items-center justify-center
                    px-6 py-3"
                >
                    {isLoading ? <LoadingSpinner /> : 'Rewrite Code'}
                </button>
            </div>
            <div className="flex flex-col flex-1 min-h-0">
                <label className="text-sm font-medium text-text-secondary mb-2">Rewritten
                Code</label>
                <div className="flex-grow p-1 bg-background border border-border rounded-md
                overflow-y-auto">
                    {isLoading && !outputCode && <div className="flex items-center justify-center
                    h-full"><LoadingSpinner /></div>}
                    {error && <p className="p-4 text-red-500">{error}</p>}
                    {outputCode && <MarkdownRenderer content={outputCode} />}
                     {!isLoading && !outputCode && !error && <div className="text-text-secondary
                     h-full flex items-center justify-center">Rewritten code will appear here.</div>}
```

```
                    </div>
                </div>
            </div>
        </div>
    );
};

// ===== AiUnitTestGenerator_2.tsx =====

import React, { useState, useCallback } from 'react';
import { generateUnitTestsStream, downloadFile } from '../../services/index.ts';
import { BeakerIcon, ArrowDownTrayIcon } from '../icons.tsx';
import { LoadingSpinner } from '../shared/index.tsx';
import { MarkdownRenderer } from '../shared/index.tsx';

const exampleCode = `import React from 'react';

export const Greeting = ({ name }) => {
  if (!name) {
    return <div>Hello, Guest!</div>;
  }
  return <div>Hello, {name}!</div>;
};`;

export const AiUnitTestGenerator: React.FC = () => {
    const [code, setCode] = useState<string>(exampleCode);
    const [tests, setTests] = useState<string>('');
    const [isLoading, setIsLoading] = useState<boolean>(false);
    const [error, setError] = useState<string>('');

    const handleGenerate = useCallback(async () => {
        if (!code.trim()) {
            setError('Please enter some code to generate tests for.');
            return;
        }
        setIsLoading(true);
        setError('');
        setTests('');
        try {
            const stream = generateUnitTestsStream(code);
            let fullResponse = '';
            for await (const chunk of stream) {
                fullResponse += chunk;
                setTests(fullResponse);
            }
        } catch (err) {
            const errorMessage = err instanceof Error ? err.message : 'An unknown error
            occurred.';
            setError(`Failed to generate tests: ${errorMessage}`);
        } finally {
            setIsLoading(false);
        }
    }, [code]);

    const cleanCodeForDownload = (markdown: string) => {
        return markdown.replace(/^```(?:\w+\n)?/, '').replace(/```$/, '');
    }

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
            <header className="mb-6">
                <h1 className="text-3xl font-bold flex items-center">
```

```
        <BeakerIcon />
        <span className="ml-3">AI Unit Test Generator</span>
    </h1>
    <p className="text-text-secondary mt-1">Provide a function or component and let
    AI write the tests.</p>
</header>
<div className="flex-grow flex flex-col gap-4 min-h-0">
    <div className="flex flex-col flex-1 min-h-0">
        <label htmlFor="code-input" className="text-sm font-medium text-text-secondary
        mb-2">Source Code</label>
        <textarea
            id="code-input"
            value={code}
            onChange={(e) => setCode(e.target.value)}
            placeholder="Paste your source code here..."
            className="flex-grow p-4 bg-surface border border-border rounded-md resize-none
            font-mono text-sm focus:ring-2 focus:ring-primary focus:outline-none"
        />
    </div>
    <div className="flex-shrink-0">
        <button
            onClick={handleGenerate}
            disabled={isLoading}
            className="btn-primary w-full max-w-xs mx-auto flex items-center justify-center
            px-6 py-3"
        >
            {isLoading ? <LoadingSpinner /> : 'Generate Unit Tests'}
        </button>
    </div>
    <div className="flex flex-col flex-1 min-h-0">
        <div className="flex justify-between items-center mb-2">
            <label className="text-sm font-medium text-text-secondary">Generated
            Tests</label>
            {tests && !isLoading && (
                <div className="flex items-center gap-2">
                    <button onClick={() =>
                    navigator.clipboard.writeText(cleanCodeForDownload(tests))} className="px-3 py-1
                    bg-gray-100 text-xs rounded-md hover:bg-gray-200">Copy Code</button>
                    <button onClick={() => downloadFile(cleanCodeForDownload(tests), 'tests.tsx',
                    'text/typescript')} className="flex items-center gap-1 px-3 py-1 bg-gray-100
                    text-xs rounded-md hover:bg-gray-200">
                        <ArrowDownTrayIcon className="w-4 h-4" /> Download
                    </button>
                </div>
            )}
        </div>
        <div className="flex-grow p-1 bg-background border border-border rounded-md
        overflow-y-auto">
            {isLoading && !tests && (
                <div className="flex items-center justify-center h-full">
                    <LoadingSpinner />
                </div>
            )}
            {error && <p className="p-4 text-red-500">{error}</p>}
            {tests && <MarkdownRenderer content={tests} />}
            {!isLoading && !tests && !error && (
                <div className="text-text-secondary h-full flex items-center justify-center">
                    The generated tests will appear here.
                </div>
            )}
        </div>
    </div>
```

```
            </div>
        </div>
    );
};

// ===== AsyncCallTreeViewer_2.tsx =====



import React, { useState, useMemo } from 'react';
import { ChartBarIcon } from '../icons.tsx';

interface CallNode {
    name: string;
    duration: number;
    children?: CallNode[];
}

const exampleJson = `{
    "name": "startApp",
    "duration": 500,
    "children": [
        {
            "name": "fetchUserData",
            "duration": 300,
            "children": [
                { "name": "authenticate", "duration": 100 },
                { "name": "fetchProfile", "duration": 150 }
            ]
        },
        {
            "name": "loadInitialAssets",
            "duration": 450,
            "children": [
                { "name": "loadImage.png", "duration": 200 },
                { "name": "loadScript.js", "duration": 250 }
            ]
        }
    ]
}`;


const TreeNode: React.FC<{ node: CallNode, level: number, maxDuration: number }>
= ({ node, level, maxDuration }) => {
    const [isOpen, setIsOpen] = React.useState(true);
    const hasChildren = node.children && node.children.length > 0;

    return (
        <div className="my-1">
            <div
                className="flex items-center p-2 rounded-md hover:bg-gray-100"
                style={{ paddingLeft: `${level * 20 + 8}px` }}
            >
                {hasChildren && (
                    <button onClick={() => setIsOpen(!isOpen)} className={`mr-2 text-text-secondary
                    w-4 h-4 flex-shrink-0 transform transition-transform ${isOpen ? 'rotate-90' :
                    ''}`}>
                        ■
                    </button>
                )}
                 {!hasChildren && <div className="w-6 mr-2 flex-shrink-0" />}
                 <div className="flex-grow flex items-center justify-between gap-4">
```

```
                    <span className="truncate">{node.name}</span>
                    <div className="flex items-center gap-2 flex-shrink-0">
                         <div className="w-24 h-4 bg-gray-200 rounded-full overflow-hidden">
                            <div className="h-4 bg-primary" style={{ width: `${(node.duration / maxDuration)
                            * 100}%` }}/>
                         </div>
                        <span className="text-primary w-16 text-
                        right">{node.duration.toFixed(0)}ms</span>
                    </div>
                </div>
            </div>
            {isOpen && hasChildren && (
                <div>
                    {node.children!.map((child, index) => (
                        <TreeNode key={index} node={child} level={level + 1} maxDuration={maxDuration}
                        />
                    ))}
                </div>
            )}
        </div>
    );
};


export const AsyncCallTreeViewer: React.FC = () => {
    const [jsonInput, setJsonInput] = useState(exampleJson);
    const [error, setError] = useState('');

    const { treeData, maxDuration } = useMemo(() => {
        try {
            const data: CallNode = JSON.parse(jsonInput);
             let max = 0;
            const findMax = (node: CallNode) => {
                if (node.duration > max) max = node.duration;
                if (node.children) node.children.forEach(findMax);
            };
            findMax(data);
            setError('');
            return { treeData: data, maxDuration: max };
        } catch (e) {
            setError('Invalid JSON format.');
            return { treeData: null, maxDuration: 0 };
        }
    }, [jsonInput]);

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
            <header className="mb-6">
                <h1 className="text-3xl flex items-center">
                    <ChartBarIcon />
                    <span className="ml-3">Async Call Tree Viewer</span>
                </h1>
                <p className="text-text-secondary mt-1">Paste a JSON structure to visualize an
                asynchronous function call tree.</p>
            </header>
            <div className="flex-grow flex flex-col gap-4 min-h-0">
                <div className="flex flex-col h-2/5 min-h-[200px]">
                    <label htmlFor="json-input" className="text-sm font-medium text-text-secondary
                    mb-2">JSON Input</label>
                    <textarea
                        id="json-input"
                        value={jsonInput}
```

```
                        onChange={e => setJsonInput(e.target.value)}
                        className={`flex-grow p-4 bg-surface border ${error ? 'border-red-500' :
                        'border-border'} rounded-md resize-y font-mono text-sm`}
                        spellCheck="false"
                    />
                    {error && <p className="text-red-500 text-xs mt-1">{error}</p>}
                </div>
                <div className="flex flex-col flex-grow min-h-0">
                    <label className="text-sm font-medium text-text-secondary mb-2">Visual
                    Tree</label>
                    <div className="flex-grow bg-surface p-4 rounded-lg text-sm overflow-y-auto
                    border border-border">
                        {treeData ? <TreeNode node={treeData} level={0} maxDuration={maxDuration} /> :
                        <div className="text-text-secondary">{error || 'Enter valid JSON to see the
                        tree.'}</div>}
                    </div>
                </div>
            </div>
        </div>
    );
};

// ===== AudioToCode_2.tsx =====

import React, { useState, useRef, useCallback } from 'react';
import { transcribeAudioToCodeStream, blobToBase64 } from
'../../services/index.ts';
import { MicrophoneIcon } from '../icons.tsx';
import { LoadingSpinner } from '../shared/index.tsx';
import { MarkdownRenderer } from '../shared/index.tsx';

export const AudioToCode: React.FC = () => {
    const [isRecording, setIsRecording] = useState(false);
    const [isLoading, setIsLoading] = useState(false);
    const [code, setCode] = useState('');
    const [error, setError] = useState('');
    const mediaRecorderRef = useRef<MediaRecorder | null>(null);
    const audioChunksRef = useRef<Blob[]>([]);

    const handleStartRecording = async () => {
        setError('');
        setCode('');
        if (!navigator.mediaDevices || !navigator.mediaDevices.getUserMedia) {
            setError('Audio recording is not supported by your browser.');
            return;
        }
        try {
            const stream = await navigator.mediaDevices.getUserMedia({ audio: true });
            mediaRecorderRef.current = new MediaRecorder(stream);
            mediaRecorderRef.current.ondataavailable = event => {
                audioChunksRef.current.push(event.data);
            };
            mediaRecorderRef.current.onstop = handleTranscribe;
            mediaRecorderRef.current.start();
            setIsRecording(true);
        } catch (err) {
            setError('Microphone access was denied. Please enable it in your browser
            settings.');
        }
    };

    const handleStopRecording = () => {
```

```
    if (mediaRecorderRef.current && isRecording) {
        mediaRecorderRef.current.stop();
        mediaRecorderRef.current.stream.getTracks().forEach(track => track.stop());
        setIsRecording(false);
        setIsLoading(true);
    }
};

const handleTranscribe = useCallback(async () => {
    if (audioChunksRef.current.length === 0) {
        setIsLoading(false);
        return;
    }
    const audioBlob = new Blob(audioChunksRef.current, { type: 'audio/webm' });
    audioChunksRef.current = [];
    try {
        const base64Audio = await blobToBase64(audioBlob);
        const stream = transcribeAudioToCodeStream(base64Audio, 'audio/webm');
        let fullResponse = '';
        for await (const chunk of stream) {
            fullResponse += chunk;
            setCode(fullResponse);
        }
    } catch (err) {
        const errorMessage = err instanceof Error ? err.message : 'An unknown error
        occurred.';
        setError(`Failed to transcribe audio: ${errorMessage}`);
    } finally {
        setIsLoading(false);
    }
}, []);

return (
    <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
        <header className="mb-6 text-center">
            <h1 className="text-3xl font-bold flex items-center justify-center">
                <MicrophoneIcon />
                <span className="ml-3">AI Audio-to-Code</span>
            </h1>
            <p className="text-text-secondary mt-1">Speak your programming ideas and watch
            them turn into code.</p>
        </header>
        <div className="flex-grow flex flex-col items-center gap-6 min-h-0">
            <div className="flex flex-col items-center justify-center bg-surface p-6
            rounded-lg w-full max-w-lg border border-border">
                 <button
                    onClick={isRecording ? handleStopRecording : handleStartRecording}
                    className={`w-24 h-24 rounded-full flex items-center justify-center text-white
                    font-bold text-lg transition-all ${isRecording ? 'bg-red-500 animate-pulse' :
                    'bg-primary'}`}
                    disabled={isLoading}
                >
                    {isLoading ? <LoadingSpinner/> : isRecording ? 'Stop' : 'Record'}
                </button>
                <p className="mt-4 text-text-secondary">
                    {isLoading ? 'Transcribing...' : isRecording ? 'Recording in progress...' :
                    'Click to start recording'}
                </p>
            </div>
             <div className="flex flex-col h-full w-full max-w-3xl">
                <label className="text-sm font-medium text-text-secondary mb-2">Generated
                Code</label>
```

```
                    <div className="flex-grow p-1 bg-background border border-border rounded-md
                    overflow-y-auto min-h-[200px]">
                        {isLoading && !code && (
                            <div className="flex items-center justify-center h-full"><LoadingSpinner
                            /></div>
                        )}
                        {error && <p className="p-4 text-red-500">{error}</p>}
                        {code && <MarkdownRenderer content={code} />}
                        {!isLoading && !code && !error && (
                            <div className="text-text-secondary h-full flex items-center justify-
                            center">Code will appear here.</div>
                        )}
                    </div>
                </div>
            </div>
        </div>
    );
};

// ===== ChangelogGenerator_2.tsx =====

import React, { useState, useCallback } from 'react';
import { generateChangelogFromLogStream } from '../../services/index.ts';
import { GitBranchIcon } from '../icons.tsx';
import { LoadingSpinner } from '../shared/index.tsx';
import { MarkdownRenderer } from '../shared/index.tsx';

const exampleLog = `commit 3a4b5c...
Author: Dev One <dev.one@example.com>
Date:   Mon Jul 15 11:30:00 2024 -0400

    feat: add user login page

commit 1a2b3c...
Author: Dev Two <dev.two@example.com>
Date:   Mon Jul 15 10:00:00 2024 -0400

    fix: correct typo in header
`;

export const ChangelogGenerator: React.FC = () => {
    const [log, setLog] = useState(exampleLog);
    const [changelog, setChangelog] = useState('');
    const [isLoading, setIsLoading] = useState(false);
    const [error, setError] = useState('');

    const handleGenerate = useCallback(async () => {
        if (!log.trim()) {
            setError('Please paste your git log output.');
            return;
        }
        setIsLoading(true);
        setError('');
        setChangelog('');
        try {
            const stream = generateChangelogFromLogStream(log);
            let fullResponse = '';
            for await (const chunk of stream) {
                fullResponse += chunk;
                setChangelog(fullResponse);
            }
        } catch (err) {
```

```
            setError(err instanceof Error ? err.message : 'An unknown error occurred.');
        } finally {
            setIsLoading(false);
        }
    }, [log]);

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
            <header className="mb-6">
                <h1 className="text-3xl font-bold flex items-center">
                    <GitBranchIcon />
                    <span className="ml-3">AI Changelog Generator</span>
                </h1>
                <p className="text-text-secondary mt-1">Generate a markdown changelog from your
                raw git log.</p>
            </header>
            <div className="flex-grow flex flex-col gap-4 min-h-0">
                <div className="flex flex-col flex-1 min-h-0">
                    <label htmlFor="commit-input" className="text-sm font-medium text-text-secondary
                    mb-2">Raw Git Log</label>
                    <textarea
                        id="commit-input"
                        value={log}
                        onChange={(e) => setLog(e.target.value)}
                        className="flex-grow p-4 bg-surface border border-border rounded-md resize-none
                        font-mono text-sm"
                    />
                </div>
                <div className="flex-shrink-0">
                    <button onClick={handleGenerate} disabled={isLoading} className="btn-primary
                    w-full max-w-xs mx-auto flex items-center justify-center px-6 py-3">
                        {isLoading ? <LoadingSpinner /> : 'Generate Changelog'}
                    </button>
                </div>
                <div className="flex flex-col flex-1 min-h-0">
                    <label className="text-sm font-medium text-text-secondary mb-2">Generated
                    Changelog.md</label>
                    <div className="relative flex-grow p-4 bg-background border border-border
                    rounded-md overflow-y-auto">
                        {isLoading && !changelog && <div className="flex items-center justify-center
                        h-full"><LoadingSpinner /></div>}
                        {error && <p className="text-red-500">{error}</p>}
                        {changelog && <MarkdownRenderer content={changelog} />}
                        {!isLoading && changelog && <button onClick={() =>
                        navigator.clipboard.writeText(changelog)} className="absolute top-2 right-2 px-2
                        py-1 bg-gray-100 text-xs rounded-md hover:bg-gray-200">Copy</button>}
                    </div>
                </div>
            </div>
        </div>
    );
};

// ===== CodeDiffGhost_2.tsx =====

import React, { useState, useEffect, useRef } from 'react';
import { EyeIcon } from '../icons.tsx';

const initialOldCode = `function UserProfile({ user }) {
  return (
    <div className="profile">
      <h1>{user.name}</h1>
```

```
      <p>{user.email}</p>
    </div>
  );
}`;

const initialNewCode = `function UserProfile({ user }) {
  const { name, email, avatar } = user;
  return (
    <div className="profile-card">
      <img src={avatar} alt={name} />
      <h2>{name}</h2>
      <a href={\`mailto:\${email}\`}>{email}</a>
    </div>
  );
}`;

export const CodeDiffGhost: React.FC = () => {
    const [oldCode, setOldCode] = useState(initialOldCode);
    const [newCode, setNewCode] = useState(initialNewCode);
    const [typedCode, setTypedCode] = useState('');
    const [isRunning, setIsRunning] = useState(false);
    const intervalRef = useRef<number | null>(null);

    const startAnimation = () => {
        if (intervalRef.current) clearInterval(intervalRef.current);
        setIsRunning(true);
        setTypedCode('');

        intervalRef.current = window.setInterval(() => {
            setTypedCode(prev => {
                if (prev.length < newCode.length) {
                    return newCode.substring(0, prev.length + 1);
                }
                if (intervalRef.current) clearInterval(intervalRef.current);
                setIsRunning(false);
                return newCode;
            });
        }, 15);
    };

    useEffect(() => {
        return () => {
            if (intervalRef.current) clearInterval(intervalRef.current);
        };
    }, []);

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
            <header className="mb-6">
                <h1 className="text-3xl flex items-center">
                    <EyeIcon />
                    <span className="ml-3">Code Diff Ghost</span>
                </h1>
                <p className="text-text-secondary mt-1">Visualize code changes with a "ghost
                typing" effect.</p>
            </header>
            <div className="flex justify-center mb-4">
                <button
                    onClick={startAnimation}
                    disabled={isRunning}
                    className="btn-primary px-6 py-2"
                >
```

```
                    {isRunning ? 'Visualizing...' : 'Show Changes'}
                </button>
            </div>
            <div className="flex-grow grid grid-cols-1 lg:grid-cols-2 gap-6 h-full overflow-
            hidden font-mono text-sm">
                <div className="flex flex-col h-full">
                    <label htmlFor="before-code" className="text-sm font-medium text-text-secondary
                    mb-2">Before</label>
                    <textarea
                        id="before-code"
                        value={oldCode}
                        onChange={e => setOldCode(e.target.value)}
                        className="flex-grow p-4 bg-surface border border-border rounded-md text-red-600
                        whitespace-pre-wrap resize-none"
                        spellCheck="false"
                    />
                </div>
                 <div className="flex flex-col h-full">
                    <label htmlFor="after-code" className="text-sm font-medium text-text-secondary
                    mb-2">After</label>
                     <div className="relative flex-grow">
                        <textarea
                            id="after-code"
                            value={newCode}
                            onChange={e => setNewCode(e.target.value)}
                            className="absolute inset-0 w-full h-full p-4 bg-surface border border-border
                            rounded-md text-emerald-700 whitespace-pre-wrap resize-none z-0"
                            spellCheck="false"
                        />
                        {(isRunning || typedCode) && (
                            <pre className="absolute inset-0 w-full h-full p-4 bg-surface pointer-events-
                            none text-emerald-700 whitespace-pre-wrap z-10">
                                {typedCode}{isRunning && <span className="animate-pulse">|</span>}
                            </pre>
                        )}
                    </div>
                </div>
            </div>
        </div>
    );
};

// ===== CodeFormatter_2.tsx =====

import React, { useState, useCallback } from 'react';
import { formatCodeStream } from '../../services/index.ts';
import { CodeBracketSquareIcon } from '../icons.tsx';
import { LoadingSpinner } from '../shared/index.tsx';
import { MarkdownRenderer } from '../shared/index.tsx';

const exampleCode = `const MyComponent = (props) => {
  const {name, items}=props
    if(!items || items.length === 0){
  return <p>No items found for {name}</p>;
    }
  return <ul>{items.map(item=> <li key={item.id}>{item.name}</li>)}</ul>
}`;

export const CodeFormatter: React.FC = () => {
    const [inputCode, setInputCode] = useState<string>(exampleCode);
    const [formattedCode, setFormattedCode] = useState<string>('');
    const [isLoading, setIsLoading] = useState<boolean>(false);
```

```
const [error, setError] = useState<string>('');

const handleFormat = useCallback(async () => {
    if (!inputCode.trim()) {
        setError('Please enter some code to format.');
        return;
    }
    setIsLoading(true);
    setError('');
    setFormattedCode('');
    try {
        const stream = formatCodeStream(inputCode);
        let fullResponse = '';
        for await (const chunk of stream) {
            fullResponse += chunk;
            setFormattedCode(fullResponse);
        }
    } catch (err) {
        const errorMessage = err instanceof Error ? err.message : 'An unknown error
        occurred.';
        setError(`Failed to format code: ${errorMessage}`);
    } finally {
        setIsLoading(false);
    }
}, [inputCode]);

return (
    <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
        <header className="mb-6">
            <h1 className="text-3xl font-bold flex items-center">
                <CodeBracketSquareIcon />
                <span className="ml-3">AI Code Formatter</span>
            </h1>
            <p className="text-text-secondary mt-1">Clean up your code with AI-powered
            formatting, like a smart Prettier.</p>
        </header>
        <div className="flex-grow flex flex-col min-h-0">
            <div className="grid grid-cols-1 lg:grid-cols-2 gap-6 flex-grow min-h-0">
                <div className="flex flex-col h-full">
                    <label htmlFor="code-input" className="text-sm font-medium text-text-secondary
                    mb-2">Input</label>
                    <textarea
                        id="code-input"
                        value={inputCode}
                        onChange={(e) => setInputCode(e.target.value)}
                        placeholder="Paste your unformatted code here..."
                        className="flex-grow p-4 bg-surface border border-border rounded-md resize-none
                        font-mono text-sm"
                    />
                </div>
                <div className="flex flex-col h-full">
                    <label className="text-sm font-medium text-text-secondary mb-2">Output</label>
                    <div className="flex-grow p-1 bg-background border border-border rounded-md
                    overflow-y-auto">
                       {isLoading && !formattedCode && (
                            <div className="flex items-center justify-center h-full">
                                <LoadingSpinner />
                            </div>
                        )}
                        {error && <p className="p-4 text-red-500">{error}</p>}
                        {formattedCode && <MarkdownRenderer content={formattedCode} />}
                        {!isLoading && !formattedCode && !error && (
```

```
                                <div className="text-text-secondary h-full flex items-center justify-center">
                                    Formatted code will appear here.
                                </div>
                            )}
                        </div>
                    </div>
                </div>
                 <button
                    onClick={handleFormat}
                    disabled={isLoading}
                    className="btn-primary mt-4 w-full max-w-sm mx-auto flex items-center justify-
                    center px-6 py-3"
                >
                    {isLoading ? <LoadingSpinner /> : 'Format Code'}
                </button>
            </div>
        </div>
    );
};

// ===== CodeReviewBot_2.tsx =====

import React, { useState, useCallback } from 'react';
import { reviewCodeStream } from '../../services/index.ts';
import { CpuChipIcon } from '../icons.tsx';
import { LoadingSpinner } from '../shared/index.tsx';
import { MarkdownRenderer } from '../shared/index.tsx';

const exampleCode = `function UserList(users) {
  if (users.length = 0) {
    return "no users";
  } else {
    return (
      users.map(u => {
        return <li>{u.name}</li>
      })
    )
  }
}`;

export const CodeReviewBot: React.FC = () => {
    const [code, setCode] = useState<string>(exampleCode);
    const [review, setReview] = useState<string>('');
    const [isLoading, setIsLoading] = useState<boolean>(false);
    const [error, setError] = useState<string>('');

    const handleGenerate = useCallback(async () => {
        if (!code.trim()) {
            setError('Please enter some code to review.');
            return;
        }
        setIsLoading(true);
        setError('');
        setReview('');
        try {
            const stream = reviewCodeStream(code);
            let fullResponse = '';
            for await (const chunk of stream) {
                fullResponse += chunk;
                setReview(fullResponse);
            }
        } catch (err) {
```

```
            const errorMessage = err instanceof Error ? err.message : 'An unknown error
            occurred.';
            setError(`Failed to get review: ${errorMessage}`);
        } finally {
            setIsLoading(false);
        }
    }, [code]);

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
            <header className="mb-6">
                <h1 className="text-3xl font-bold flex items-center">
                    <CpuChipIcon />
                    <span className="ml-3">AI Code Review Bot</span>
                </h1>
                <p className="text-text-secondary mt-1">Get an automated code review from
                Gemini.</p>
            </header>
            <div className="flex-grow flex flex-col gap-4 min-h-0">
                <div className="flex flex-col flex-1 min-h-0">
                    <label htmlFor="code-input" className="text-sm font-medium text-text-secondary
                    mb-2">Code to Review</label>
                    <textarea
                        id="code-input"
                        value={code}
                        onChange={(e) => setCode(e.target.value)}
                        placeholder="Paste your code here..."
                        className="flex-grow p-4 bg-surface border border-border rounded-md resize-none
                        font-mono text-sm"
                    />
                </div>
                 <div className="flex-shrink-0">
                    <button
                        onClick={handleGenerate}
                        disabled={isLoading}
                        className="btn-primary w-full max-w-xs mx-auto flex items-center justify-center
                        px-6 py-3"
                    >
                        {isLoading ? <LoadingSpinner /> : 'Request Review'}
                    </button>
                </div>
                <div className="flex flex-col flex-1 min-h-0">
                    <label className="text-sm font-medium text-text-secondary mb-2">AI
                    Feedback</label>
                    <div className="flex-grow p-4 bg-background border border-border rounded-md
                    overflow-y-auto">
                        {isLoading && !review && <div className="flex items-center justify-center
                        h-full"><LoadingSpinner /></div>}
                        {error && <p className="text-red-500">{error}</p>}
                        {review && <MarkdownRenderer content={review} />}
                         {!isLoading && !review && !error && <div className="text-text-secondary h-full
                         flex items-center justify-center">Review will appear here.</div>}
                    </div>
                </div>
            </div>
        </div>
    );
};

// ===== CodeSpellChecker_2.tsx =====

import React, { useState, useMemo } from 'react';
```

```
import { BeakerIcon } from '../icons.tsx';

const commonTypos = [
    'funtion', 'functoin', 'funciton', 'contructor', 'cosntructor',
    'consle', 'conosle', 'cosnole', 'varable', 'varaible', 'vairable',
    'docment', 'docuemnt', 'docmunet', 'componnet', 'componenet', 'compnent',
    'retunr', 'retrun', 'asnyc', 'asycn', 'awai', 'awiat', 'promse',
    'resolv', 'rejct', 'catach', 'thne', 'lenght', 'lengt', 'prperty',
    'undefinded', 'nul', 'booleon', 'numbar', 'srtring', 'arrya', 'objcet',
    'elemnt', 'attriubte', 'eveent', 'listner', 'handeler', 'clieck',
    'submitt', 'resposne', 'requset', 'stauts', 'eror', 'sucess',
    'implemnt', 'overide', 'extned', 'pbulic', 'prvate', 'procted',
    'statci', 'abstact', 'interace', 'enmu', 'moduel', 'packge',
    'importt', 'exprot', 'defualt', 'namspace', 'tyep', 'clsas',
    'whiel', 'swich', 'cse', 'brek', 'contiune', 'thrwo', 'finnaly'
];

const typoRegex = new RegExp(`\\b(${commonTypos.join('|')})\\b`, 'gi');

const HighlightedText: React.FC<{ text: string }> = React.memo(({ text }) => {
    const parts = useMemo(() => {
        return text.split(typoRegex).map((part, i) => {
            if (typoRegex.test(part)) {
                return <span key={i} className="underline decoration-red-500 decoration-wavy"
                title={`Possible typo`}>{part}</span>;
            }
            return part;
        });
    }, [text]);

    return <>{parts}</>;
});

export const CodeSpellChecker: React.FC = () => {
    const [code, setCode] = useState('funtion myFunction() {\n  consle.log("Hello
    World");\n  const myVarable = docment.getElementById("root");\n  // This is a
    React componnet\n}');

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
            <header className="mb-6">
                <h1 className="text-3xl flex items-center">
                    <BeakerIcon />
                    <span className="ml-3">Code Spell Checker</span>
                </h1>
                <p className="text-text-secondary mt-1">A simple tool that finds and highlights
                common typos in code.</p>
            </header>
            <div className="relative flex-grow font-mono text-sm bg-surface border border-
            border rounded-lg p-4 overflow-auto">
                <textarea
                    value={code}
                    onChange={(e) => setCode(e.target.value)}
                    className="absolute inset-0 w-full h-full p-4 bg-transparent text-transparent
                    caret-primary resize-none z-10"
                    spellCheck="false"
                />
                <pre className="absolute inset-0 w-full h-full p-4 pointer-events-none
                whitespace-pre-wrap" aria-hidden="true">
                    <HighlightedText text={code} />
                </pre>
            </div>
```

```
            <p className="text-xs text-text-secondary mt-2 text-center">This checker uses a
            predefined list of common typos and does not use AI.</p>
        </div>
    );
};

// ===== ColorPaletteGenerator_2.tsx =====

import React, { useState, useCallback } from 'react';
import { HexColorPicker } from 'react-colorful';
import { generateColorPalette } from '../../services/index.ts';
import { SparklesIcon } from '../icons.tsx';
import { LoadingSpinner } from '../shared/index.tsx';

export const ColorPaletteGenerator: React.FC = () => {
    const [baseColor, setBaseColor] = useState("#0047AB");
    const [palette, setPalette] = useState<string[]>(['#0047AB', '#3366CC',
    '#6688D1', '#99AADD', '#CCD3E8', '#F0F2F5']);
    const [isLoading, setIsLoading] = useState<boolean>(false);
    const [error, setError] = useState<string>('');

    const handleGenerate = useCallback(async () => {
        setIsLoading(true);
        setError('');
        try {
            const result = await generateColorPalette(baseColor);
            setPalette(result.colors);
        } catch (err) {
            const errorMessage = err instanceof Error ? err.message : 'An unknown error
            occurred.';
            setError(`Failed to generate palette: ${errorMessage}`);
        } finally {
            setIsLoading(false);
        }
    }, [baseColor]);

    const handleCopy = (color: string) => {
        navigator.clipboard.writeText(color);
    };

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
            <header className="mb-6 text-center">
                <h1 className="text-3xl font-bold flex items-center justify-center">
                    <SparklesIcon />
                    <span className="ml-3">AI Color Palette Generator</span>
                </h1>
                <p className="text-text-secondary mt-1">Pick a base color and let Gemini design
                a beautiful palette for you.</p>
            </header>
            <div className="flex-grow flex flex-col md:flex-row items-center justify-center
            gap-8">
                <div className="flex flex-col items-center gap-4">
                     <HexColorPicker color={baseColor} onChange={setBaseColor} className="!w-64
                     !h-64"/>
                     <div className="p-2 bg-surface rounded-md font-mono text-lg border border-
                     border" style={{color: baseColor}}>{baseColor}</div>
                      <button
                        onClick={handleGenerate}
                        disabled={isLoading}
                        className="btn-primary w-full flex items-center justify-center px-6 py-3"
                    >
```

```
                        {isLoading ? <LoadingSpinner /> : 'Generate Palette'}
                    </button>
                    {error && <p className="text-red-500 text-sm mt-2">{error}</p>}
                </div>
                <div className="flex flex-col gap-2 w-full max-w-sm">
                    <label className="text-sm font-medium text-text-secondary mb-2">Generated
                    Palette:</label>
                    {isLoading ? (
                         <div className="flex items-center justify-center h-48"><LoadingSpinner /></div>
                    ) : (
                        palette.map((color) => (
                            <div key={color} className="group flex items-center justify-between p-4 rounded-
                            md shadow-md border border-border" style={{ backgroundColor: color }}>
                                <span className="font-mono font-bold text-black/70 mix-blend-
                                overlay">{color}</span>
                                <button
                                    onClick={() => handleCopy(color)}
                                    className="opacity-0 group-hover:opacity-100 transition-opacity bg-white/30
                                    hover:bg-white/50 px-3 py-1 rounded text-xs text-black font-semibold backdrop-
                                    blur-sm">
                                    Copy
                                </button>
                            </div>
                        ))
                    )}
                </div>
            </div>
        </div>
    );
};

// ===== CommandPaletteTrigger_2.tsx =====

import React from 'react';
import { CommandLineIcon } from '../icons.tsx';

export const CommandPaletteTrigger: React.FC = () => {
    return (
        <div className="flex flex-col items-center justify-center h-full p-8 text-center
        text-text-secondary">
            <div className="text-6xl mb-4 text-primary" aria-hidden="true">
                <CommandLineIcon />
            </div>
            <h1 className="text-3xl font-bold text-text-primary mb-2">
                Command Palette
            </h1>
            <p className="text-lg mb-4 max-w-md">
                The Command Palette provides quick keyboard access to all features and commands.
            </p>
            <div className="bg-surface text-primary border border-border rounded-lg px-6
            py-4 animate-pulse shadow-sm">
                <p className="font-semibold text-text-primary">Press <kbd className="mx-1 font-
                sans px-2 py-1.5 text-xs font-semibold text-gray-800 bg-gray-100 border border-
                gray-200 rounded-lg">Ctrl</kbd> + <kbd className="mx-1 font-sans px-2 py-1.5
                text-xs font-semibold text-gray-800 bg-gray-100 border border-gray-200 rounded-
                lg">K</kbd> to open.</p>
            </div>
        </div>
    );
};

// ===== Connections.tsx =====
```

```
import React, { useState, useEffect } from 'react';
import { useGlobalState } from '../../contexts/GlobalStateContext.tsx';
import { GithubIcon } from '../icons.tsx';
import { getRepos, deleteRepo } from '../../services/index.ts';
import type { Repo } from '../../types.ts';
import { LoadingSpinner } from '../shared/index.tsx';


const ConnectionCard: React.FC<{
    name: string;
    icon: React.ReactNode;
    isConnected: boolean;
    userLogin?: string;
    onDisconnect: () => void;
}> = ({ name, icon, isConnected, userLogin, onDisconnect }) => {
    return (
        <div className="bg-surface border border-border rounded-lg p-6 flex items-center
        justify-between">
            <div className="flex items-center gap-4">
                <div className="w-10 h-10">{icon}</div>
                <div>
                    <h3 className="text-lg font-bold text-text-primary">{name}</h3>
                    <p className={`text-sm ${isConnected ? 'text-green-600' : 'text-text-
                    secondary'}`}>
                        {isConnected ? `Connected as ${userLogin}` : 'Not Connected'}
                    </p>
                </div>
            </div>
            {isConnected && (
                <button onClick={onDisconnect} className="px-4 py-2 bg-red-100 text-red-700
                font-semibold rounded-md hover:bg-red-200">Disconnect</button>
            )}
        </div>
    );
};

const RepoSelector: React.FC = () => {
    const { state, dispatch } = useGlobalState();
    const [repos, setRepos] = useState<Repo[]>([]);
    const [isLoading, setIsLoading] = useState(false);

    useEffect(() => {
        if (state.isGithubConnected) {
            setIsLoading(true);
            getRepos().then(setRepos).finally(() => setIsLoading(false));
        }
    }, [state.isGithubConnected]);

    const handleSelectRepo = (fullName: string) => {
        if (!fullName) {
            dispatch({ type: 'SET_SELECTED_REPO', payload: null });
            dispatch({ type: 'LOAD_PROJECT_FILES', payload: null });
            return;
        }
        const [owner, repo] = fullName.split('/');
        dispatch({ type: 'SET_SELECTED_REPO', payload: { owner, repo } });
    };

    const handleDeleteRepo = async () => {
        if (!state.selectedRepo) return;
        const { owner, repo } = state.selectedRepo;
        if (window.confirm(`Are you sure you want to PERMANENTLY DELETE the repository
```

```
        "${owner}/${repo}"? This action cannot be undone.`)) {
            try {
                await deleteRepo(owner, repo);
                dispatch({ type: 'SET_SELECTED_REPO', payload: null });
                dispatch({ type: 'LOAD_PROJECT_FILES', payload: null });
                setIsLoading(true);
                getRepos().then(setRepos).finally(() => setIsLoading(false));
            } catch (e) {
                alert("Failed to delete repository.");
            }
        }
    };

    return (
        <div className="bg-surface border border-border rounded-lg p-6">
            <h3 className="text-lg font-bold text-text-primary mb-2">Active Repository</h3>
            <p className="text-sm text-text-secondary mb-4">Select a repository for the AI
            to interact with in the Project Explorer and other tools.</p>
            {isLoading ? <LoadingSpinner /> : (
                <div className="flex items-center gap-2">
                    <select
                        value={state.selectedRepo ?
                        `${state.selectedRepo.owner}/${state.selectedRepo.repo}` : ''}
                        onChange={e => handleSelectRepo(e.target.value)}
                        className="flex-grow p-2 rounded bg-background border border-border"
                    >
                        <option value="">-- Select a Repository --</option>
                        {repos.map(r => <option key={r.id} value={r.full_name}>{r.full_name}</option>)}
                    </select>
                    {state.selectedRepo && (
                         <button onClick={handleDeleteRepo} className="px-4 py-2 bg-red-100 text-red-700
                         font-semibold rounded-md hover:bg-red-200">Delete</button>
                    )}
                </div>
            )}
        </div>
    );
}

export const Connections: React.FC = () => {
    const { state, dispatch } = useGlobalState();
    const { isGithubConnected, user } = state;

    const handleDisconnectGitHub = () => {
        dispatch({ type: 'LOGOUT' });
    };

    return (
        <div className="h-full p-4 sm:p-6 lg:p-8">
            <div className="max-w-4xl mx-auto">
                <header className="mb-8">
                    <h1 className="text-4xl font-extrabold tracking-tight">Connections</h1>
                    <p className="mt-2 text-lg text-text-secondary">Manage your GitHub connection to
                    unlock powerful workflows.</p>
                </header>

                <div className="space-y-6">
                    <ConnectionCard
                        name="GitHub"
                        icon={<GithubIcon />}
                        isConnected={isGithubConnected}
                        userLogin={user?.login}
```

```
                        onDisconnect={handleDisconnectGitHub}
                    />
                    {isGithubConnected && <RepoSelector />}
                </div>
            </div>
        </div>
    );
};

// ===== CronJobBuilder_2.tsx =====

import React, { useState, useMemo, useCallback, useEffect } from 'react';
import { CommandLineIcon, SparklesIcon } from '../icons.tsx';
import { generateCronFromDescription, CronParts } from
'../../services/index.ts';
import { LoadingSpinner } from '../shared/index.tsx';

const CronPartSelector: React.FC<{ label: string, value: string, onChange:
(value: string) => void, options: (string|number)[] }> = ({ label, value,
onChange, options }) => {
    return (
        <div>
            <label className="block text-sm font-medium text-text-secondary">{label}</label>
            <select value={value} onChange={e => onChange(e.target.value)} className="w-full
            mt-1 px-3 py-2 rounded-md bg-surface border border-border">
                <option value="*">* (every)</option>
                {options.map(o => <option key={o} value={o}>{o}</option>)}
            </select>
        </div>
    );
};

export const CronJobBuilder: React.FC<{ initialPrompt?: string }> = ({
initialPrompt }) => {
    const [minute, setMinute] = useState('0');
    const [hour, setHour] = useState('17');
    const [dayOfMonth, setDayOfMonth] = useState('*');
    const [month, setMonth] = useState('*');
    const [dayOfWeek, setDayOfWeek] = useState('1-5');
    const [aiPrompt, setAiPrompt] = useState(initialPrompt || 'every weekday at
    5pm');
    const [isLoading, setIsLoading] = useState(false);

    const cronExpression = useMemo(() => {
        return `${minute} ${hour} ${dayOfMonth} ${month} ${dayOfWeek}`;
    }, [minute, hour, dayOfMonth, month, dayOfWeek]);

    const handleAiGenerate = useCallback(async (p: string) => {
        if (!p) return;
        setIsLoading(true);
        try {
            const result: CronParts = await generateCronFromDescription(p);
            setMinute(result.minute);
            setHour(result.hour);
            setDayOfMonth(result.dayOfMonth);
            setMonth(result.month);
            setDayOfWeek(result.dayOfWeek);
        } catch (e) {
            console.error(e);
        } finally {
            setIsLoading(false);
        }
```

```
    }, []);

    useEffect(() => {
        if (initialPrompt) {
            setAiPrompt(initialPrompt);
            handleAiGenerate(initialPrompt);
        }
    }, [initialPrompt, handleAiGenerate]);

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
            <header className="mb-6">
                <h1 className="text-3xl font-bold flex items-center">
                    <CommandLineIcon />
                    <span className="ml-3">AI Cron Job Builder</span>
                </h1>
                <p className="text-text-secondary mt-1">Visually construct a cron expression or
                describe it in plain English.</p>
            </header>
             <div className="flex gap-2 mb-6">
                <input type="text" value={aiPrompt} onChange={e => setAiPrompt(e.target.value)}
                placeholder="Describe a schedule..." className="flex-grow px-3 py-1.5 rounded-md
                bg-surface border border-border text-sm"/>
                <button onClick={() => handleAiGenerate(aiPrompt)} disabled={isLoading}
                className="btn-primary px-4 py-1.5 flex items-center gap-2">
                    {isLoading ? <LoadingSpinner /> : <SparklesIcon />} AI Generate
                </button>
            </div>
            <div className="grid grid-cols-2 md:grid-cols-5 gap-4 mb-6">
                <CronPartSelector label="Minute" value={minute} onChange={setMinute}
                options={Array.from({length: 60}, (_, i) => i)} />
                <CronPartSelector label="Hour" value={hour} onChange={setHour}
                options={Array.from({length: 24}, (_, i) => i)} />
                <CronPartSelector label="Day (Month)" value={dayOfMonth}
                onChange={setDayOfMonth} options={Array.from({length: 31}, (_, i) => i + 1)} />
                <CronPartSelector label="Month" value={month} onChange={setMonth}
                options={Array.from({length: 12}, (_, i) => i + 1)} />
                <CronPartSelector label="Day (Week)" value={dayOfWeek} onChange={setDayOfWeek}
                options={Array.from({length: 7}, (_, i) => i)} />
            </div>
            <div className="bg-surface p-4 rounded-lg text-center border border-border">
                <p className="text-text-secondary text-sm">Generated Expression</p>
                <p className="font-mono text-primary text-2xl mt-1">{cronExpression}</p>
                 <button onClick={() => navigator.clipboard.writeText(cronExpression)}
                 className="mt-4 px-3 py-1 bg-gray-100 hover:bg-gray-200 rounded-md text-
                 xs">Copy</button>
            </div>
        </div>
    );
};

// ===== CssGridEditor_2.tsx =====

import React, { useState, useMemo } from 'react';
import { CodeBracketSquareIcon, ArrowDownTrayIcon } from '../icons.tsx';
import { downloadFile } from '../../services/index.ts';

const initialSettings = { rows: 3, cols: 4, rowGap: 1, colGap: 1 };

export const CssGridEditor: React.FC = () => {
    const [rows, setRows] = useState(initialSettings.rows);
    const [cols, setCols] = useState(initialSettings.cols);
```

```
    const [rowGap, setRowGap] = useState(initialSettings.rowGap);
    const [colGap, setColGap] = useState(initialSettings.colGap);

    const gridStyle = {
        display: 'grid',
        gridTemplateColumns: `repeat(${cols}, 1fr)`,
        gridTemplateRows: `repeat(${rows}, 1fr)`,
        gap: `${rowGap}rem ${colGap}rem`,
        height: '100%',
        width: '100%'
    };

    const cssCode = useMemo(() => {
        return `.grid-container {
  display: grid;
  grid-template-columns: repeat(${cols}, 1fr);
  grid-template-rows: repeat(${rows}, 1fr);
  gap: ${rowGap}rem ${colGap}rem;
}`;
    }, [rows, cols, rowGap, colGap]);

    const handleCopy = () => {
        navigator.clipboard.writeText(cssCode);
    };

    const handleDownload = () => {
        downloadFile(cssCode, 'grid.css', 'text/css');
    };

    const handleReset = () => {
        setRows(initialSettings.rows);
        setCols(initialSettings.cols);
        setRowGap(initialSettings.rowGap);
        setColGap(initialSettings.colGap);
    };

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
            <header className="mb-6">
                <h1 className="text-3xl font-bold flex items-center">
                    <CodeBracketSquareIcon />
                    <span className="ml-3">CSS Grid Visual Editor</span>
                </h1>
                <p className="text-text-secondary mt-1">Configure your grid layout and copy the
                generated CSS.</p>
            </header>
            <div className="flex-grow grid grid-cols-1 lg:grid-cols-3 gap-6 min-h-0">
                <div className="lg:col-span-1 flex flex-col gap-4 bg-surface border border-
                border p-6 rounded-lg overflow-y-auto">
                    <div className="flex justify-between items-center">
                        <h3 className="text-xl font-bold">Controls</h3>
                        <button onClick={handleReset} className="text-xs px-3 py-1 bg-gray-100 hover:bg-
                        gray-200 rounded-md">Reset</button>
                    </div>
                    <div className="space-y-4">
                        <div>
                            <label htmlFor="rows" className="block text-sm font-medium text-text-
                            secondary">Rows ({rows})</label>
                            <input id="rows" type="range" min="1" max="12" value={rows} onChange={e =>
                            setRows(Number(e.target.value))} className="w-full h-2 bg-gray-200 rounded-lg
                            appearance-none cursor-pointer" />
                        </div>
```

```
                        <div>
                            <label htmlFor="cols" className="block text-sm font-medium text-text-
                            secondary">Columns ({cols})</label>
                            <input id="cols" type="range" min="1" max="12" value={cols} onChange={e =>
                            setCols(Number(e.target.value))} className="w-full h-2 bg-gray-200 rounded-lg
                            appearance-none cursor-pointer" />
                        </div>
                         <div>
                            <label htmlFor="rowGap" className="block text-sm font-medium text-text-
                            secondary">Row Gap ({rowGap}rem)</label>
                            <input id="rowGap" type="range" min="0" max="8" step="0.25" value={rowGap}
                            onChange={e => setRowGap(Number(e.target.value))} className="w-full h-2 bg-
                            gray-200 rounded-lg appearance-none cursor-pointer" />
                        </div>
                         <div>
                            <label htmlFor="colGap" className="block text-sm font-medium text-text-
                            secondary">Column Gap ({colGap}rem)</label>
                            <input id="colGap" type="range" min="0" max="8" step="0.25" value={colGap}
                            onChange={e => setColGap(Number(e.target.value))} className="w-full h-2 bg-
                            gray-200 rounded-lg appearance-none cursor-pointer" />
                        </div>
                    </div>
                     <div className="flex-grow mt-4 flex flex-col min-h-[150px]">
                        <div className="flex justify-between items-center mb-2">
                            <label className="text-sm font-medium text-text-secondary">Generated CSS</label>
                            <div className="flex gap-2">
                                <button onClick={handleCopy} className="px-2 py-1 bg-gray-100 hover:bg-gray-200
                                rounded-md text-xs">Copy</button>
                                <button onClick={handleDownload} className="flex items-center gap-1 px-2 py-1
                                bg-gray-100 hover:bg-gray-200 rounded-md text-xs"><ArrowDownTrayIcon
                                className="w-4 h-4"/> Download</button>
                            </div>
                        </div>
                        <div className="relative flex-grow">
                            <pre className="bg-background p-4 rounded-md text-primary text-sm overflow-auto
                            h-full w-full absolute">{cssCode}</pre>
                        </div>
                    </div>
                </div>
                <div className="lg:col-span-2 bg-background rounded-lg p-4 border-2 border-
                dashed border-border">
                    <div style={gridStyle}>
                        {Array.from({ length: rows * cols }).map((_, i) => (
                            <div key={i} className="bg-primary/10 rounded-lg border-2 border-dashed border-
                            primary/50 flex items-center justify-center text-primary">
                                <span className="text-xs opacity-70">{i + 1}</span>
                            </div>
                        ))}
                    </div>
                </div>
            </div>
        </div>
    );
};

// ===== DevNotesStickyPanel_2.tsx =====

import React, { useState, useCallback } from 'react';
import { FileCodeIcon, SparklesIcon } from '../icons.tsx';
import { useLocalStorage } from '../../hooks/useLocalStorage.ts';
import { summarizeNotesStream } from '../../services/index.ts';
import { LoadingSpinner } from '../shared/index.tsx';
```

```
import { MarkdownRenderer } from '../shared/index.tsx';

interface Note {
    id: number;
    text: string;
    x: number;
    y: number;
    color: string;
}

const colors = ['bg-yellow-400 text-yellow-900', 'bg-green-400 text-green-900',
'bg-blue-400 text-blue-900', 'bg-pink-400 text-pink-900', 'bg-purple-400 text-
purple-900'];

export const DevNotesStickyPanel: React.FC = () => {
    const [notes, setNotes] = useLocalStorage<Note[]>('devcore_notes', []);
    const [dragging, setDragging] = useState<{ id: number; offsetX: number; offsetY:
    number } | null>(null);
    const [isSummarizing, setIsSummarizing] = useState(false);
    const [summary, setSummary] = useState('');

    const handleSummarize = useCallback(async () => {
        if (notes.length === 0) return;
        setIsSummarizing(true);
        setSummary('');
        try {
            const allNotesText = notes.map((n: Note) => `- ${n.text}`).join('\n');
            const stream = summarizeNotesStream(allNotesText);
            let fullResponse = '';
            for await (const chunk of stream) {
                fullResponse += chunk;
                setSummary(fullResponse);
            }
        } catch (error) {
            console.error(error);
            setSummary('Sorry, an error occurred while summarizing.');
        } finally {
            setIsSummarizing(false);
        }
    }, [notes]);


    const addNote = () => {
        const newNote: Note = {
            id: Date.now(),
            text: 'New note...',
            x: 50 + (notes.length % 10) * 20,
            y: 50 + (notes.length % 10) * 20,
            color: colors[notes.length % colors.length],
        };
        setNotes([...notes, newNote]);
    };

    const updateText = (id: number, text: string) => {
        setNotes(notes.map((n: Note) => n.id === id ? { ...n, text } : n));
    };

    const deleteNote = (id: number, e: React.MouseEvent) => {
        e.stopPropagation();
        setNotes(notes.filter((n: Note) => n.id !== id));
    };
    const onMouseDown = (e: React.MouseEvent<HTMLDivElement>, id: number) => {
```

```
    if((e.target as HTMLElement).tagName === 'TEXTAREA' || (e.target as
    HTMLElement).tagName === 'BUTTON') return;
    const noteElement = e.currentTarget;
    const rect = noteElement.getBoundingClientRect();
    setDragging({ id, offsetX: e.clientX - rect.left, offsetY: e.clientY - rect.top
    });
};

const onMouseMove = (e: React.MouseEvent<HTMLDivElement>) => {
    if (!dragging) return;
    const boardRect = e.currentTarget.getBoundingClientRect();
    setNotes(
        notes.map((n: Note) =>
            n.id === dragging.id
                ? { ...n, x: e.clientX - dragging.offsetX - boardRect.left, y: e.clientY -
                dragging.offsetY - boardRect.top }
                : n
        )
    );
};

const onMouseUp = () => setDragging(null);

return (
    <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
        <header className="mb-6 flex justify-between items-center">
             <div>
                <h1 className="text-3xl font-bold flex items-center"><FileCodeIcon /><span
                className="ml-3">Dev Notes Sticky Panel</span></h1>
                <p className="text-text-secondary mt-1">A place for your thoughts, todos, and
                random ideas.</p>
            </div>
            <div className="flex gap-2">
                <button onClick={handleSummarize} disabled={isSummarizing || notes.length === 0}
                className="btn-primary flex items-center gap-2 px-4 py-2">
                    <SparklesIcon/> {isSummarizing ? 'Summarizing...' : 'AI Summarize'}
                </button>
                <button onClick={addNote} className="btn-primary px-6 py-2">Add Note</button>
            </div>
        </header>
        <div
            className="relative flex-grow bg-background border-2 border-dashed border-border
            rounded-lg overflow-hidden"
            onMouseMove={onMouseMove}
            onMouseUp={onMouseUp}
            onMouseLeave={onMouseUp}
        >
            {notes.map((note: Note) => (
                <div
                    key={note.id}
                    className={`group absolute w-48 h-48 p-2 flex flex-col shadow-lg cursor-grab
                    active:cursor-grabbing rounded-md transition-transform duration-100 border
                    border-black/40 ${note.color}`}
                    style={{ top: note.y, left: note.x, transform: dragging?.id === note.id ?
                    'scale(1.05)' : 'scale(1)' }}
                    onMouseDown={e => onMouseDown(e, note.id)}
                >
                     <button onClick={(e) => deleteNote(note.id, e)} className="absolute -top-2
                     -right-2 w-6 h-6 rounded-full bg-gray-700 text-white font-bold text-xs flex
                     items-center justify-center opacity-0 group-hover:opacity-100 hover:bg-red-500
                     transition-all">&times;</button>
                    <textarea
```

```
                            value={note.text}
                            onChange={(e) => updateText(note.id, e.target.value)}
                            className="w-full h-full bg-transparent resize-none focus:outline-none font-
                            medium p-1 placeholder:text-inherit/50"
                        />
                    </div>
                ))}
            </div>
            {(isSummarizing || summary) && (
                 <div className="fixed inset-0 bg-gray-900/80 backdrop-blur-sm z-50 flex items-
                 center justify-center" onClick={() => setSummary('')}>
                    <div className="w-full max-w-2xl bg-surface border border-border rounded-lg
                    shadow-2xl p-6" onClick={e => e.stopPropagation()}>
                        <h2 className="text-xl font-bold mb-4">AI Summary of Notes</h2>
                        {isSummarizing && !summary ? <LoadingSpinner /> : <MarkdownRenderer
                        content={summary} />}
                    </div>
                </div>
            )}
        </div>
    );
};

// ===== DigitalWhiteboard.tsx =====

import React, { useState, useCallback } from 'react';
import { SparklesIcon, DigitalWhiteboardIcon } from '../icons.tsx';
import { useLocalStorage } from '../../hooks/useLocalStorage.ts';
import { summarizeNotesStream } from '../../services/index.ts';
import { LoadingSpinner } from '../shared/index.tsx';
import { MarkdownRenderer } from '../shared/index.tsx';

interface Note {
    id: number;
    text: string;
    x: number;
    y: number;
    color: string;
}

const colors = ['bg-yellow-400', 'bg-green-400', 'bg-blue-400', 'bg-pink-400',
'bg-purple-400', 'bg-orange-400'];
const textColors = ['text-yellow-900', 'text-green-900', 'text-blue-900', 'text-
pink-900', 'text-purple-900', 'text-orange-900'];

export const DigitalWhiteboard: React.FC = () => {
    const [notes, setNotes] = useLocalStorage<Note[]>('devcore_whiteboard_notes',
    []);
    const [dragging, setDragging] = useState<{ id: number; offsetX: number; offsetY:
    number } | null>(null);
    const [isSummarizing, setIsSummarizing] = useState(false);
    const [summary, setSummary] = useState('');

    const handleSummarize = useCallback(async () => {
        if (notes.length === 0) return;
        setIsSummarizing(true);
        setSummary('');
        try {
            const allNotesText = notes.map((n: Note) => `- ${n.text}`).join('\n');
            const stream = summarizeNotesStream(allNotesText);
            let fullResponse = '';
            for await (const chunk of stream) {
```

```
            fullResponse += chunk;
            setSummary(fullResponse);
        }
    } catch (error) {
        console.error(error);
        setSummary('Sorry, an error occurred while summarizing.');
    } finally {
        setIsSummarizing(false);
    }
}, [notes]);

const addNote = () => {
    const newNote: Note = {
        id: Date.now(),
        text: 'New idea...',
        x: 50,
        y: 50,
        color: colors[notes.length % colors.length],
    };
    setNotes([...notes, newNote]);
};

const deleteNote = (id: number, e: React.MouseEvent) => {
    e.stopPropagation();
    setNotes(notes.filter((n) => n.id !== id));
};

const updateNote = (id: number, updates: Partial<Note>) => {
    setNotes(notes.map((n) => n.id === id ? { ...n, ...updates } : n));
};

const onMouseDown = (e: React.MouseEvent<HTMLDivElement>, id: number) => {
    const target = e.target as HTMLElement;
    if (target.tagName === 'TEXTAREA' || target.dataset.role === 'button') return;

    const noteElement = e.currentTarget;
    const rect = noteElement.getBoundingClientRect();
    setDragging({ id, offsetX: e.clientX - rect.left, offsetY: e.clientY - rect.top
    });
};

const onMouseMove = (e: React.MouseEvent<HTMLDivElement>) => {
    if (!dragging) return;
    const boardRect = e.currentTarget.getBoundingClientRect();
    updateNote(dragging.id, {
        x: e.clientX - dragging.offsetX - boardRect.left,
        y: e.clientY - dragging.offsetY - boardRect.top
    });
};

const onMouseUp = () => setDragging(null);

return (
    <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
        <header className="mb-6 flex justify-between items-center">
             <div>
                <h1 className="text-3xl font-bold flex items-center"><DigitalWhiteboardIcon
                /><span className="ml-3">Digital Whiteboard</span></h1>
                <p className="text-text-secondary mt-1">Organize your ideas with interactive
                sticky notes and AI summaries.</p>
            </div>
            <div className="flex gap-2">
```

```
                    <button onClick={handleSummarize} disabled={isSummarizing || notes.length === 0}
                    className="btn-primary flex items-center gap-2 px-4 py-2">
                        <SparklesIcon/> {isSummarizing ? 'Summarizing...' : 'AI Summarize'}
                    </button>
                    <button onClick={addNote} className="btn-primary px-6 py-2">Add Note</button>
                </div>
            </header>
            <div
                className="relative flex-grow bg-background border-2 border-dashed border-border
                rounded-lg overflow-hidden"
                onMouseMove={onMouseMove} onMouseUp={onMouseUp} onMouseLeave={onMouseUp}
            >
                {notes.map((note) => (
                    <div
                        key={note.id}
                        className={`group absolute w-56 h-56 p-2 flex flex-col shadow-lg cursor-grab
                        active:cursor-grabbing rounded-md transition-transform duration-100 border
                        border-black/40 ${note.color} ${textColors[colors.indexOf(note.color)]}`}
                        style={{ top: note.y, left: note.x, transform: dragging?.id === note.id ?
                        'scale(1.05)' : 'scale(1)' }}
                        onMouseDown={e => onMouseDown(e, note.id)}
                    >
                        <button data-role="button" onClick={(e) => deleteNote(note.id, e)}
                        className="absolute -top-2 -right-2 w-6 h-6 rounded-full bg-gray-700 text-white
                        font-bold text-xs flex items-center justify-center opacity-0 group-
                        hover:opacity-100 hover:bg-red-500 transition-all">&times;</button>
                        <textarea
                            value={note.text}
                            onChange={(e) => updateNote(note.id, { text: e.target.value })}
                            className="w-full h-full bg-transparent resize-none focus:outline-none font-
                            medium p-1"
                        />
                        <div data-role="button" className="flex-shrink-0 flex justify-center gap-1 p-1
                        opacity-0 group-hover:opacity-100 transition-opacity">
                            {colors.map((c, i) => <button key={c} onClick={() => updateNote(note.id, {
                            color: c })} className={`w-4 h-4 rounded-full ${c} border border-black/20
                            ${note.color === c ? 'ring-2 ring-offset-1 ring-black/50' : ''}`}/>)}
                        </div>
                    </div>
                ))}
            </div>
             {(isSummarizing || summary) && (
                 <div className="fixed inset-0 bg-gray-900/80 backdrop-blur-sm z-50 flex items-
                 center justify-center" onClick={() => setSummary('')}>
                    <div className="w-full max-w-2xl bg-surface border border-border rounded-lg
                    shadow-2xl p-6" onClick={e => e.stopPropagation()}>
                        <h2 className="text-xl font-bold mb-4">AI Summary of Notes</h2>
                        {isSummarizing && !summary ? <LoadingSpinner /> : <MarkdownRenderer
                        content={summary} />}
                    </div>
                </div>
            )}
        </div>
    );
};

// ===== FontPairingTool_2.tsx =====

import React, { useState, useEffect } from 'react';
import { EyeIcon } from '../icons.tsx';

const popularFonts = [
```

```
    'Roboto', 'Open Sans', 'Lato', 'Montserrat', 'Oswald', 'Source Sans Pro',
    'Raleway', 'Poppins', 'Nunito', 'Merriweather',
    'Playfair Display', 'Lora', 'Noto Sans', 'Ubuntu', 'PT Sans', 'Slabo 27px'
];

export const FontPairingTool: React.FC = () => {
    const [headingFont, setHeadingFont] = useState('Oswald');
    const [bodyFont, setBodyFont] = useState('Roboto');

    useEffect(() => {
        const fontsToLoad = [headingFont, bodyFont].filter(Boolean).join('|');
        if (fontsToLoad) {
            const linkId = 'font-pairing-stylesheet';
            let link = document.getElementById(linkId) as HTMLLinkElement;
            if (!link) {
                link = document.createElement('link');
                link.id = linkId;
                link.rel = 'stylesheet';
                document.head.appendChild(link);
            }
            link.href = `https://fonts.googleapis.com/css?family=${fontsToLoad.replace(/ /g,
            '+')}:400,700&display=swap`;
        }
    }, [headingFont, bodyFont]);

    const FontSelector: React.FC<{ label: string, value: string, onChange: (font:
    string) => void }> = ({ label, value, onChange }) => (
        <div>
            <label className="block text-sm font-medium text-text-secondary">{label}</label>
            <select value={value} onChange={e => onChange(e.target.value)} className="w-full
            mt-1 px-3 py-2 rounded-md bg-surface border border-border">
                {popularFonts.map(font => <option key={font} value={font}>{font}</option>)}
            </select>
        </div>
    );

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
            <header className="mb-6">
                <h1 className="text-3xl font-bold flex items-center">
                    <EyeIcon />
                    <span className="ml-3">Font Pairing Tool</span>
                </h1>
                <p className="text-text-secondary mt-1">Preview Google Font combinations for
                your projects.</p>
            </header>
            <div className="flex-grow grid grid-cols-1 lg:grid-cols-3 gap-6 min-h-0">
                <div className="lg:col-span-1 flex flex-col gap-4 bg-surface border border-
                border p-6 rounded-lg">
                    <h3 className="text-xl font-bold">Controls</h3>
                    <FontSelector label="Heading Font" value={headingFont} onChange={setHeadingFont}
                    />
                    <FontSelector label="Body Font" value={bodyFont} onChange={setBodyFont} />
                </div>
                <div className="lg:col-span-2 bg-background border border-border rounded-lg p-8
                overflow-y-auto">
                    <h2 className="text-4xl font-bold mb-4" style={{ fontFamily: `'${headingFont}',
                    sans-serif` }}>
                        The Quick Brown Fox Jumps Over the Lazy Dog
                    </h2>
                    <p className="text-lg" style={{ fontFamily: `'${bodyFont}', sans-serif` }}>
                        Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed non risus.
```

```
                        Suspendisse lectus tortor, dignissim sit amet, adipiscing nec, ultricies sed,
                        dolor. Cras elementum ultrices diam. Maecenas ligula massa, varius a, semper
                        congue, euismod non, mi. Proin porttitor, orci nec nonummy molestie, enim est
                        eleifend mi, non fermentum diam nisl sit amet erat.
                    </p>
                </div>
            </div>
        </div>
    );
};

// ===== FontPreviewPicker_2.tsx =====

import React, { useState, useEffect } from 'react';
import { EyeIcon } from '../icons.tsx';

const popularFonts = [
    'Roboto', 'Open Sans', 'Lato', 'Montserrat', 'Oswald', 'Source Sans Pro',
    'Raleway', 'Poppins', 'Nunito', 'Merriweather',
    'Playfair Display', 'Lora', 'Noto Sans', 'Ubuntu', 'PT Sans', 'Slabo 27px'
];

export const FontPreviewPicker: React.FC = () => {
    const [selectedFont, setSelectedFont] = useState('Roboto');
    const [previewText, setPreviewText] = useState('The quick brown fox jumps over
    the lazy dog.');

    useEffect(() => {
        if (selectedFont) {
            const linkId = 'font-preview-stylesheet';
            let link = document.getElementById(linkId) as HTMLLinkElement;
            if (!link) {
                link = document.createElement('link');
                link.id = linkId;
                link.rel = 'stylesheet';
                document.head.appendChild(link);
            }
            link.href = `https://fonts.googleapis.com/css?family=${selectedFont.replace(/
            /g, '+')}&display=swap`;
        }
    }, [selectedFont]);

    const importRule = `@import
    url('https://fonts.googleapis.com/css?family=${selectedFont.replace(/ /g,
    '+')}&display=swap');`;
    const cssRule = `font-family: '${selectedFont}', sans-serif;`;

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
            <header className="mb-6">
                <h1 className="text-3xl font-bold flex items-center">
                    <EyeIcon />
                    <span className="ml-3">Font Preview & Picker</span>
                </h1>
                <p className="text-text-secondary mt-1">Preview Google Fonts and get the CSS
                import rule.</p>
            </header>
            <div className="flex-grow grid grid-cols-1 lg:grid-cols-3 gap-6 min-h-0">
                <div className="lg:col-span-1 flex flex-col gap-4 bg-surface border border-
                border p-6 rounded-lg">
                    <h3 className="text-xl font-bold">Controls</h3>
                    <div>
```

```
                        <label htmlFor="font-select" className="block text-sm font-medium text-text-
                        secondary">Select Font</label>
                        <select id="font-select" value={selectedFont} onChange={e =>
                        setSelectedFont(e.target.value)} className="w-full mt-1 px-3 py-2 rounded-md bg-
                        surface border border-border">
                            {popularFonts.map(font => <option key={font} value={font}>{font}</option>)}
                        </select>
                    </div>
                    <div>
                        <label htmlFor="preview-text" className="block text-sm font-medium text-text-
                        secondary">Preview Text</label>
                        <textarea id="preview-text" value={previewText} onChange={e =>
                        setPreviewText(e.target.value)} className="w-full mt-1 p-2 rounded-md bg-surface
                        border border-border h-24 resize-none"></textarea>
                    </div>
                    <div className="space-y-2">
                        <label className="block text-sm font-medium text-text-secondary">CSS
                        Rules</label>
                        <div className="relative"><pre className="bg-background p-2 rounded-md text-
                        primary text-xs overflow-x-auto">{importRule}</pre><button onClick={() =>
                        navigator.clipboard.writeText(importRule)} className="absolute top-1 right-1
                        px-2 py-0.5 bg-gray-100 hover:bg-gray-200 rounded-md text-
                        xs">Copy</button></div>
                        <div className="relative"><pre className="bg-background p-2 rounded-md text-
                        primary text-xs overflow-x-auto">{cssRule}</pre><button onClick={() =>
                        navigator.clipboard.writeText(cssRule)} className="absolute top-1 right-1 px-2
                        py-0.5 bg-gray-100 hover:bg-gray-200 rounded-md text-xs">Copy</button></div>
                    </div>
                </div>
                <div className="lg:col-span-2 bg-background border border-border rounded-lg p-8
                flex items-center justify-center">
                    <p className="text-4xl" style={{ fontFamily: `'${selectedFont}', sans-serif` }}>
                        {previewText}
                    </p>
                </div>
            </div>
        </div>
    );
};

// ===== JsonTreeNavigator_2.tsx =====

import React, { useState } from 'react';
import { FileCodeIcon } from '../icons.tsx';

interface JsonNodeProps {
    data: any;
    nodeKey: string;
    isRoot?: boolean;
}

const JsonNode: React.FC<JsonNodeProps> = ({ data, nodeKey, isRoot = false }) =>
{
    const [isOpen, setIsOpen] = useState(isRoot);
    const isObject = typeof data === 'object' && data !== null;

    const toggleOpen = () => setIsOpen(!isOpen);

    if (!isObject) {
        return (
            <div className="ml-4 pl-4 border-l border-border">
                <span className="text-purple-700">{nodeKey}: </span>
```

```
                <span className={typeof data === 'string' ? 'text-green-700' : 'text-
                orange-700'}>
                    {typeof data === 'string' ? `"${data}"` : String(data)}
                </span>
            </div>
        );
    }

    const entries = Object.entries(data);
    const bracket = Array.isArray(data) ? '[]' : '{}';

    return (
        <div className={`ml-4 ${!isRoot ? 'pl-4 border-l border-border' : ''}`}>
            <button onClick={toggleOpen} className="flex items-center cursor-pointer
            hover:bg-gray-100 rounded px-1">
                <span className={`transform transition-transform ${isOpen ? 'rotate-90' :
                'rotate-0'}`}>■</span>
                <span className="ml-1 text-purple-700">{nodeKey}:</span>
                <span className="ml-2 text-text-secondary">{bracket[0]}</span>
                {!isOpen && <span className="text-text-secondary">...{bracket[1]}</span>}
            </button>
            {isOpen && (
                <div>
                    {entries.map(([key, value]) => (
                        <JsonNode key={key} nodeKey={key} data={value} />
                    ))}
                    <div className="text-text-secondary ml-4">{bracket[1]}</div>
                </div>
            )}
        </div>
    );
};

export const JsonTreeNavigator: React.FC<{ initialData?: object }> = ({
initialData }) => {
    const defaultJson = '{\n  "id": "devcore-001",\n  "active": true,\n  "features":
    [\n    "ai-explainer",\n    "api-tester"\n  ],\n  "config": {\n    "theme":
    "dark",\n    "version": 1\n  }\n}';
    const [jsonInput, setJsonInput] = useState(initialData ?
    JSON.stringify(initialData, null, 2) : defaultJson);
    const [parsedData, setParsedData] = useState<any>(() => {
        try {
            return JSON.parse(jsonInput);
        } catch {
            return null;
        }
    });
    const [error, setError] = useState('');

    const parseJson = (input: string) => {
        try {
            const parsed = JSON.parse(input);
            setParsedData(parsed);
            setError('');
        } catch (e) {
            if (e instanceof Error) setError(e.message);
            setParsedData(null);
        }
    };

    const handleInputChange = (e: React.ChangeEvent<HTMLTextAreaElement>) => {
        setJsonInput(e.target.value);
```

```
        parseJson(e.target.value);
    }

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
            <header className="mb-6">
                <h1 className="text-3xl font-bold flex items-center">
                    <FileCodeIcon />
                    <span className="ml-3">JSON Tree Navigator</span>
                </h1>
                <p className="text-text-secondary mt-1">Paste your JSON data to visualize it as
                a collapsible tree.</p>
            </header>
            <div className="flex-grow flex flex-col gap-4 min-h-0">
                <div className="flex flex-col h-2/5 min-h-[200px]">
                    <label htmlFor="json-input" className="text-sm font-medium text-text-secondary
                    mb-2">JSON Input</label>
                    <textarea
                        id="json-input"
                        value={jsonInput}
                        onChange={handleInputChange}
                        className={`flex-grow p-4 bg-surface border ${error ? 'border-red-500' :
                        'border-border'} rounded-md resize-y font-mono text-sm focus:ring-2 focus:ring-
                        primary focus:outline-none`}
                    />
                    {error && <p className="text-red-500 text-xs mt-1">{error}</p>}
                </div>
                 <div className="flex flex-col flex-grow min-h-0">
                    <label className="text-sm font-medium text-text-secondary mb-2">Tree
                    View</label>
                    <div className="flex-grow p-4 bg-surface border border-border rounded-md
                    overflow-y-auto font-mono text-sm">
                        {parsedData ? <JsonNode data={parsedData} nodeKey="root" isRoot /> : <div
                        className="text-text-secondary">Enter valid JSON to view</div>}
                    </div>
                </div>
            </div>
        </div>
    );
};

// ===== LogicFlowBuilder_2.tsx =====

import React, { useState, useRef, useMemo, useCallback } from 'react';
import { ALL_FEATURES } from '../features/index.ts';
import type { Feature } from '../../types.ts';
import { MapIcon } from '../icons.tsx';

interface Node {
    id: number;
    featureId: string;
    x: number;
    y: number;
}

interface Link {
    from: number;
    to: number;
}

const featuresMap = new Map(ALL_FEATURES.map(f => [f.id, f]));
const FeaturePaletteItem: React.FC<{ feature: Feature, onDragStart: (e:
```

```
React.DragEvent, featureId: string) => void }> = ({ feature, onDragStart }) => (
    <div
        draggable
        onDragStart={e => onDragStart(e, feature.id)}
        className="p-3 rounded-md bg-gray-50 border border-border flex items-center
        gap-3 cursor-grab hover:bg-gray-100 transition-colors"
    >
        <div className="text-primary flex-shrink-0">{feature.icon}</div>
        <div>
            <h4 className="font-bold text-sm text-text-primary">{feature.name}</h4>
            <p className="text-xs text-text-secondary">{feature.category}</p>
        </div>
    </div>
);

const NodeComponent: React.FC<{
    node: Node;
    feature: Feature;
    onMouseDown: (e: React.MouseEvent, id: number) => void;
    onLinkStart: (e: React.MouseEvent, id: number) => void;
    onLinkEnd: (e: React.MouseEvent, id: number) => void;
}> = ({ node, feature, onMouseDown, onLinkStart, onLinkEnd }) => (
    <div
        className="absolute w-52 bg-surface rounded-lg shadow-md border-2 border-border
        cursor-grab active:cursor-grabbing flex flex-col"
        style={{ left: node.x, top: node.y, transform: 'translate(-50%, -50%)' }}
        onMouseDown={e => onMouseDown(e, node.id)}
        onMouseUp={e => onLinkEnd(e, node.id)}
    >
        <div className="p-2 flex items-center gap-2 border-b border-border">
            <div className="w-5 h-5 text-primary">{feature.icon}</div>
            <span className="text-sm font-semibold truncate text-text-
            primary">{feature.name}</span>
        </div>
        <div className="relative p-3 text-xs text-text-secondary min-h-[40px] flex
        items-center justify-center">
            Workflow Node
            <div
                onMouseDown={e => onLinkStart(e, node.id)}
                className="absolute right-[-9px] top-1/2 -translate-y-1/2 w-4 h-4 bg-primary
                rounded-full border-2 border-surface cursor-crosshair hover:scale-125
                transition-transform"
                title="Drag to connect"
            />
        </div>
    </div>
);

const SVGGrid: React.FC = React.memo(() => (
    <svg width="100%" height="100%" className="absolute inset-0">
        <defs>
            <pattern id="smallGrid" width="10" height="10" patternUnits="userSpaceOnUse">
                <path d="M 10 0 L 0 0 0 10" fill="none" stroke="rgba(0, 0, 0, 0.05)"
                strokeWidth="0.5"/>
            </pattern>
            <pattern id="grid" width="50" height="50" patternUnits="userSpaceOnUse">
                <rect width="50" height="50" fill="url(#smallGrid)"/>
                <path d="M 50 0 L 0 0 0 50" fill="none" stroke="rgba(0, 0, 0, 0.1)"
                strokeWidth="1"/>
            </pattern>
        </defs>
        <rect width="100%" height="100%" fill="url(#grid)" />
```

```
    </svg>
));

export const LogicFlowBuilder: React.FC = () => {
    const [nodes, setNodes] = useState<Node[]>([]);
    const [links, setLinks] = useState<Link[]>([]);
    const [draggingNode, setDraggingNode] = useState<{ id: number; offsetX: number;
    offsetY: number } | null>(null);
    const [linking, setLinking] = useState<{ from: number; fromPos: { x: number; y:
    number }; toPos: { x: number; y: number } } | null>(null);
    const canvasRef = useRef<HTMLDivElement>(null);

    const handleDragStart = (e: React.DragEvent, featureId: string) => {
        e.dataTransfer.setData('application/json', JSON.stringify({ featureId }));
    };

    const handleDrop = (e: React.DragEvent) => {
        e.preventDefault();
        if (!canvasRef.current) return;
        const { featureId } = JSON.parse(e.dataTransfer.getData('application/json'));
        const canvasRect = canvasRef.current.getBoundingClientRect();
        const newNode: Node = {
            id: Date.now(),
            featureId,
            x: e.clientX - canvasRect.left,
            y: e.clientY - canvasRect.top,
        };
        setNodes(prev => [...prev, newNode]);
    };

    const handleNodeMouseDown = (e: React.MouseEvent, id: number) => {
        const node = nodes.find(n => n.id === id);
        if (!node || (e.target as HTMLElement).title === 'Drag to connect') return;
        setDraggingNode({ id, offsetX: e.clientX - node.x, offsetY: e.clientY - node.y
        });
    };

    const handleCanvasMouseMove = (e: React.MouseEvent) => {
        if (draggingNode && canvasRef.current) {
            setNodes(nodes.map(n => n.id === draggingNode.id ? { ...n, x: e.clientX -
            draggingNode.offsetX, y: e.clientY - draggingNode.offsetY } : n));
        }
        if (linking && canvasRef.current) {
            const canvasRect = canvasRef.current.getBoundingClientRect();
            setLinking({ ...linking, toPos: { x: e.clientX - canvasRect.left, y: e.clientY -
            canvasRect.top } });
        }
    };

    const handleCanvasMouseUp = () => {
        setDraggingNode(null);
        setLinking(null);
    };

    const handleLinkStart = (e: React.MouseEvent, id: number) => {
        e.stopPropagation();
        const fromNode = nodes.find(n => n.id === id);
        if (!fromNode) return;
        setLinking({ from: id, fromPos: { x: fromNode.x, y: fromNode.y }, toPos: { x:
        fromNode.x, y: fromNode.y } });
    };
    const handleLinkEnd = (e: React.MouseEvent, id: number) => {
```

```
    e.stopPropagation();
    if (linking && linking.from !== id) {
        setLinks(prev => [...prev, { from: linking.from, to: id }]);
    }
    setLinking(null);
};

const nodePositions = useMemo(() => new Map(nodes.map(n => [n.id, { x: n.x, y:
n.y }])), [nodes]);

return (
    <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
        <header className="mb-6">
            <h1 className="text-3xl font-bold flex items-center"><MapIcon /><span
            className="ml-3">Logic Flow Builder</span></h1>
            <p className="text-text-secondary mt-1">Visually build application logic flows
            and development pipelines.</p>
        </header>
        <div className="flex-grow flex gap-6 min-h-0">
            <aside className="w-72 flex-shrink-0 bg-surface border border-border p-4
            rounded-lg flex flex-col">
                <h3 className="font-bold mb-3 text-lg">Features</h3>
                <div className="flex-grow overflow-y-auto space-y-3 pr-2">
                    {ALL_FEATURES.map(feature => <FeaturePaletteItem key={feature.id}
                    feature={feature} onDragStart={handleDragStart} />)}
                </div>
            </aside>
            <main
                ref={canvasRef}
                className="flex-grow relative bg-background border-2 border-dashed border-border
                rounded-lg overflow-hidden"
                onDrop={handleDrop}
                onDragOver={e => e.preventDefault()}
                onMouseMove={handleCanvasMouseMove}
                onMouseUp={handleCanvasMouseUp}
                onMouseLeave={handleCanvasMouseUp}
            >
                <SVGGrid />
                <svg width="100%" height="100%" className="absolute inset-0 pointer-events-
                none">
                    {links.map((link, i) => {
                        const fromNode = nodePositions.get(link.from);
                        const toNode = nodePositions.get(link.to);
                        if (!fromNode || !toNode) return null;
                        return <line key={i} x1={fromNode.x} y1={fromNode.y} x2={toNode.x} y2={toNode.y}
                        stroke="var(--color-primary)" strokeWidth="2" markerEnd="url(#arrow)" />;
                    })}
                    {linking && <line x1={linking.fromPos.x} y1={linking.fromPos.y}
                    x2={linking.toPos.x} y2={linking.toPos.y} stroke="var(--color-primary)"
                    strokeWidth="2" strokeDasharray="5,5" />}
                    <defs><marker id="arrow" viewBox="0 0 10 10" refX="8" refY="5" markerWidth="6"
                    markerHeight="6" orient="auto-start-reverse"><path d="M 0 0 L 10 5 L 0 10 z"
                    fill="var(--color-primary)" /></marker></defs>
                </svg>
                {nodes.map(node => {
                    const feature = featuresMap.get(node.featureId);
                    return feature ? <NodeComponent key={node.id} node={node} feature={feature}
                    onMouseDown={handleNodeMouseDown} onLinkStart={handleLinkStart}
                    onLinkEnd={handleLinkEnd} /> : null;
                })}
            </main>
        </div>
```

```
        </div>
    );
};

// ===== MarkdownSlides_2.tsx =====

import React, { useState, useMemo, useEffect, useRef, useCallback } from
'react';
import { marked } from 'marked';
import { PhotoIcon } from '../icons.tsx';

const exampleMarkdown = `# Slide 1: Welcome

This is a slide deck generated from Markdown.

- Use standard markdown syntax
- Like lists, headers, and **bold** text.

---

# Slide 2: Features

Navigate using the buttons below.

\`\`\`javascript
console.log("Code blocks work too!");
\`\`\`

---

# Slide 3: The End

Easy to create and present.
`;

export const MarkdownSlides: React.FC = () => {
    const [markdown, setMarkdown] = useState(exampleMarkdown);
    const [currentSlide, setCurrentSlide] = useState(0);
    const [slideHtml, setSlideHtml] = useState<string | TrustedHTML>('');
    const presentationRef = useRef<HTMLDivElement>(null);

    const slides = useMemo(() => markdown.split(/^-{3,}\s*$/m), [markdown]);

    useEffect(() => {
        const parse = async () => {
            const currentSlideContent = slides[currentSlide] || '';
            const html = await marked.parse(currentSlideContent);
            setSlideHtml(html);
        };
        parse();
    }, [slides, currentSlide]);

    const goToNext = useCallback(() => setCurrentSlide(s => Math.min(s + 1,
    slides.length - 1)), [slides.length]);
    const goToPrev = useCallback(() => setCurrentSlide(s => Math.max(s - 1, 0)),
    []);

    const handleFullscreen = () => {
        presentationRef.current?.requestFullscreen();
    };

    useEffect(() => {
```

```
        const handleKeyDown = (e: KeyboardEvent) => {
            if (document.fullscreenElement === presentationRef.current) {
                if (e.key === 'ArrowRight' || e.key === ' ') goToNext();
                if (e.key === 'ArrowLeft') goToPrev();
            }
        };
        document.addEventListener('keydown', handleKeyDown);
        return () => document.removeEventListener('keydown', handleKeyDown);
    }, [goToNext, goToPrev]);

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
            <header className="mb-6">
                <h1 className="text-3xl font-bold flex items-center"><PhotoIcon /><span
                className="ml-3">Markdown to Slides</span></h1>
                <p className="text-text-secondary mt-1">Write markdown, present it as a
                slideshow. Use '---' to separate slides.</p>
            </header>
            <div className="flex-grow grid grid-cols-1 lg:grid-cols-2 gap-6 h-full overflow-
            hidden">
                <div className="flex flex-col h-full">
                     <label htmlFor="md-input" className="text-sm font-medium text-text-secondary
                     mb-2">Markdown Editor</label>
                     <textarea id="md-input" value={markdown} onChange={e =>
                     setMarkdown(e.target.value)} className="flex-grow p-4 bg-surface border border-
                     border rounded-md resize-none font-mono text-sm focus:ring-2 focus:ring-primary
                     focus:outline-none"/>
                </div>
                 <div ref={presentationRef} className="flex flex-col h-full bg-surface
                 fullscreen:bg-background border border-border rounded-md">
                    <div className="flex-shrink-0 flex justify-end items-center p-2 border-b border-
                    border">
                        <button onClick={handleFullscreen} className="px-3 py-1 bg-gray-100 rounded-md
                        text-xs hover:bg-gray-200">Fullscreen</button>
                    </div>
                    <div className="relative flex-grow flex flex-col justify-center items-center p-8
                    overflow-y-auto">
                        <div className="prose prose-lg max-w-none w-full" dangerouslySetInnerHTML={{
                        __html: slideHtml }} />
                         <button onClick={goToPrev} disabled={currentSlide === 0} className="absolute
                         left-4 top-1/2 -translate-y-1/2 p-2 bg-gray-200/50 rounded-full
                         disabled:opacity-30 hover:bg-gray-300/50">■</button>
                         <button onClick={goToNext} disabled={currentSlide === slides.length - 1}
                         className="absolute right-4 top-1/2 -translate-y-1/2 p-2 bg-gray-200/50 rounded-
                         full disabled:opacity-30 hover:bg-gray-300/50">■</button>
                         <div className="absolute bottom-4 right-4 text-xs bg-black/50 px-2 py-1 rounded-
                         md text-white">
                            {currentSlide + 1} / {slides.length}
                        </div>
                    </div>
                </div>
            </div>
        </div>
    );
};

// ===== MetaTagEditor_2.tsx =====

import React, { useState, useMemo } from 'react';
import { CodeBracketSquareIcon } from '../icons.tsx';

interface MetaData {
```

```
    title: string;
    description: string;
    image: string;
    url: string;
}

const SocialCardPreview: React.FC<{ meta: MetaData }> = ({ meta }) => (
    <div className="w-full max-w-md mx-auto bg-surface border border-border
    rounded-2xl overflow-hidden shadow-lg">
        <div className="h-52 bg-gray-100 flex items-center justify-center">
            {meta.image ? <img src={meta.image} alt="Preview" className="w-full h-full
            object-cover" onError={(e) => e.currentTarget.style.display='none'}/> : <span
            className="text-text-secondary">Image Preview</span>}
        </div>
        <div className="p-4">
            <p className="text-xs text-text-secondary truncate">{new URL(meta.url ||
            'https://example.com').hostname}</p>
            <h3 className="font-bold text-text-primary truncate mt-1">{meta.title || 'Your
            Title Here'}</h3>
            <p className="text-sm text-text-secondary mt-1 line-clamp-2">{meta.description
            || 'A concise description of your content will appear here.'}</p>
        </div>
    </div>
);

export const MetaTagEditor: React.FC = () => {
    const [meta, setMeta] = useState<MetaData>({
        title: 'DevCore AI Toolkit', description: 'The ultimate toolkit for modern
        developers, powered by Gemini.',
        image: 'https://storage.googleapis.com/maker-studio-project-images-
        prod/programming_power_on_a_laptop_3a8f0bb1_39a9_4c2b_81f0_a74551480f2c.png',
        url: 'https://devcore.example.com'
    });

    const handleChange = (e: React.ChangeEvent<HTMLInputElement>) => {
        setMeta({ ...meta, [e.target.name]: e.target.value });
    };

    const generatedHtml = useMemo(() => {
        return `<!-- Primary Meta Tags -->
<title>${meta.title}</title>
<meta name="title" content="${meta.title}" />
<meta name="description" content="${meta.description}" />
<!-- Open Graph / Facebook -->
<meta property="og:type" content="website" />
<meta property="og:url" content="${meta.url}" />
<meta property="og:title" content="${meta.title}" />
<meta property="og:description" content="${meta.description}" />
<meta property="og:image" content="${meta.image}" />
<!-- Twitter -->
<meta property="twitter:card" content="summary_large_image" />
<meta property="twitter:url" content="${meta.url}" />
<meta property="twitter:title" content="${meta.title}" />
<meta property="twitter:description" content="${meta.description}" />
<meta property="twitter:image" content="${meta.image}" />`;
    }, [meta]);

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
            <header className="mb-6"><h1 className="text-3xl font-bold flex items-
            center"><CodeBracketSquareIcon /><span className="ml-3">Meta Tag
            Editor</span></h1><p className="text-text-secondary mt-1">Generate SEO & social
```

```
            media meta tags with a live preview.</p></header>
            <div className="flex-grow grid grid-cols-1 lg:grid-cols-2 xl:grid-cols-3 gap-6
            min-h-0">
                <div className="xl:col-span-1 flex flex-col gap-4 bg-surface border border-
                border p-6 rounded-lg overflow-y-auto">
                    <h3 className="text-xl font-bold">Metadata</h3>
                    <div><label className="block text-sm">Title</label><input type="text"
                    name="title" value={meta.title} onChange={handleChange} className="w-full mt-1
                    p-2 rounded bg-background border border-border"/></div>
                    <div><label className="block text-sm">Description</label><input type="text"
                    name="description" value={meta.description} onChange={handleChange}
                    className="w-full mt-1 p-2 rounded bg-background border border-border"/></div>
                    <div><label className="block text-sm">Canonical URL</label><input type="text"
                    name="url" value={meta.url} onChange={handleChange} className="w-full mt-1 p-2
                    rounded bg-background border border-border"/></div>
                    <div><label className="block text-sm">Social Image URL</label><input type="text"
                    name="image" value={meta.image} onChange={handleChange} className="w-full mt-1
                    p-2 rounded bg-background border border-border"/></div>
                </div>
                <div className="xl:col-span-1 flex flex-col">
                     <label className="text-sm font-medium text-text-secondary mb-2">Generated
                     HTML</label>
                     <div className="relative flex-grow"><pre className="w-full h-full bg-background
                     p-4 rounded-md text-primary text-sm overflow-auto">{generatedHtml}</pre><button
                     onClick={() => navigator.clipboard.writeText(generatedHtml)} className="absolute
                     top-2 right-2 px-2 py-1 bg-gray-100 hover:bg-gray-200 rounded-md text-
                     xs">Copy</button></div>
                </div>
                 <div className="hidden xl:flex flex-col items-center justify-center">
                    <label className="text-sm font-medium text-text-secondary mb-2">Live
                    Preview</label>
                    <SocialCardPreview meta={meta} />
                </div>
            </div>
        </div>
    );
};

// ===== NetworkVisualizer_2.tsx =====

import React, { useState, useEffect, useMemo } from 'react';
import { ChartBarIcon } from '../icons.tsx';

type SortKey = 'name' | 'initiatorType' | 'transferSize' | 'duration';
type SortDirection = 'asc' | 'desc';

const SummaryCard: React.FC<{ title: string, value: string | number }> = ({
title, value }) => (
    <div className="bg-surface border border-border p-3 rounded-lg text-center">
        <p className="text-xs text-text-secondary">{title}</p>
        <p className="text-xl font-bold text-text-primary">{value}</p>
    </div>
);

export const NetworkVisualizer: React.FC = () => {
    const [requests, setRequests] = useState<PerformanceResourceTiming[]>([]);
    const [sortKey, setSortKey] = useState<SortKey>('duration');
    const [sortDirection, setSortDirection] = useState<SortDirection>('desc');

    useEffect(() => {
        const entries = performance.getEntriesByType("resource") as
        PerformanceResourceTiming[];
```

```
    setRequests(entries);
}, []);

const sortedRequests = useMemo(() => {
    return [...requests].sort((a, b) => {
        const valA = a[sortKey];
        const valB = b[sortKey];
        if (valA < valB) return sortDirection === 'asc' ? -1 : 1;
        if (valA > valB) return sortDirection === 'asc' ? 1 : -1;
        return 0;
    });
}, [requests, sortKey, sortDirection]);

const { totalSize, totalDuration, maxDuration } = useMemo(() => {
    const totalSize = requests.reduce((acc, req) => acc + req.transferSize, 0);
    const maxFinish = Math.max(...requests.map(r => r.startTime + r.duration), 0);
    return { totalSize, totalDuration: maxFinish, maxDuration:
    Math.max(...requests.map(r => r.duration), 0) };
}, [requests]);

const handleSort = (key: SortKey) => {
    setSortDirection(sortKey === key && sortDirection === 'desc' ? 'asc' : 'desc');
    setSortKey(key);
};

const formatBytes = (bytes: number) => {
    if (bytes === 0) return '0 B';
    const k = 1024; const sizes = ['B', 'KB', 'MB'];
    const i = Math.floor(Math.log(bytes) / Math.log(k));
    return parseFloat((bytes / Math.pow(k, i)).toFixed(1)) + ' ' + sizes[i];
};

const SortableHeader: React.FC<{ skey: SortKey, label: string; className?:
string }> = ({ skey, label, className }) => (
    <th onClick={() => handleSort(skey)} className={`p-2 text-left cursor-pointer
    hover:bg-gray-100 ${className}`}>
        {label} {sortKey === skey && (sortDirection === 'asc' ? '▲' : '▼')}
    </th>
);

return (
    <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
        <header className="mb-6"><h1 className="text-3xl font-bold flex items-
        center"><ChartBarIcon /><span className="ml-3">Network Visualizer</span></h1><p
        className="text-text-secondary mt-1">Inspect network resources with a summary
        and waterfall chart.</p></header>
        <div className="grid grid-cols-2 md:grid-cols-4 gap-4 mb-4">
            <SummaryCard title="Total Requests" value={requests.length} />
            <SummaryCard title="Total Transferred" value={formatBytes(totalSize)} />
            <SummaryCard title="Finish Time" value={`${totalDuration.toFixed(0)}ms`} />
            <SummaryCard title="Longest Request" value={`${maxDuration.toFixed(0)}ms`} />
        </div>
        <div className="flex-grow overflow-auto bg-surface rounded-lg border border-
        border">
            <table className="w-full text-sm text-left table-fixed">
                <thead className="sticky top-0 bg-surface z-10"><tr className="border-b border-
                border">
                    <SortableHeader skey="name" label="Name" className="w-2/5"/>
                    <SortableHeader skey="initiatorType" label="Type" className="w-1/5" />
                    <SortableHeader skey="transferSize" label="Size" className="w-1/5"/>
                    <SortableHeader skey="duration" label="Time / Waterfall" className="w-1/5"/>
                </tr></thead>
```

```
                    <tbody>{sortedRequests.map((req, i) => (<tr key={i} className="border-b border-
                    border hover:bg-gray-50">
                        <td className="p-2 text-primary truncate"
                        title={req.name}>{req.name.split('/').pop()}</td>
                        <td className="p-2">{req.initiatorType}</td>
                        <td className="p-2">{formatBytes(req.transferSize)}</td>
                        <td className="p-2"><div className="flex items-center">
                            <span className="w-12">{req.duration.toFixed(0)}ms</span>
                            <div className="flex-grow h-4 bg-gray-200 rounded overflow-hidden">
                                <div className="h-4 bg-primary rounded" style={{ marginLeft: `${(req.startTime /
                                totalDuration) * 100}%`, width: `${(req.duration / totalDuration) * 100}%` }}
                                title={`Start: ${req.startTime.toFixed(0)}ms`}></div>
                            </div>
                        </div></td>
                    </tr>))}</tbody>
                </table>
            </div>
        </div>
    );
};

// ===== PrGenerator_2.tsx =====

import React, { useState, useMemo } from 'react';
import { GitBranchIcon } from '../icons.tsx';

export const PrGenerator: React.FC = () => {
    const [title, setTitle] = useState('feat: Implement new login flow');
    const [description, setDescription] = useState('This PR introduces a new, more
    secure login flow using OAuth.');
    const [fromBranch, setFromBranch] = useState('feature/new-login');
    const [toBranch, setToBranch] = useState('main');
    const [changeType, setChangeType] = useState('feat');
    const [relatedIssue, setRelatedIssue] = useState('#123');
    const [testingSteps, setTestingSteps] = useState('1. Navigate to /login\n2.
    Click "Login with OAuth"\n3. Verify you are redirected to the dashboard.');

    const markdownPreview = useMemo(() => {
        return `
# ${changeType}: ${title}
${relatedIssue ? `\n**Closes:** ${relatedIssue}\n` : ''}
**Branch:** \`${fromBranch}\` -> \`${toBranch}\`

## Description
${description}

## Testing Steps
${testingSteps}
        `.trim();
    }, [title, description, fromBranch, toBranch, changeType, relatedIssue,
    testingSteps]);

    const handleCopy = () => {
        navigator.clipboard.writeText(markdownPreview);
    };

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
            <header className="mb-6">
                <h1 className="text-3xl flex items-center"><GitBranchIcon /><span
                className="ml-3">Pull Request Generator</span></h1>
                <p className="text-text-secondary mt-1">Draft a professional pull request from a
```

```
                structured template.</p>
            </header>
            <div className="flex-grow grid grid-cols-1 lg:grid-cols-2 gap-6 min-h-0">
                <form className="flex flex-col gap-4 overflow-y-auto pr-2 bg-surface border
                border-border p-4 rounded-lg">
                    <div className="bg-background p-4 rounded-lg flex items-center gap-4 border
                    border-border">
                        <input type="text" value={fromBranch} onChange={e =>
                        setFromBranch(e.target.value)} className="w-full px-3 py-1.5 rounded-md bg-
                        surface text-sm font-mono border border-border"/>
                        <span className="font-bold text-text-secondary">→</span>
                         <input type="text" value={toBranch} onChange={e => setToBranch(e.target.value)}
                         className="w-full px-3 py-1.5 rounded-md bg-surface text-sm font-mono border
                         border-border"/>
                    </div>
                    <div className="flex gap-2">
                        <div className="w-1/4"><label className="block text-sm">Type</label><select
                        value={changeType} onChange={e => setChangeType(e.target.value)}
                        className="w-full mt-1 p-2 rounded bg-surface border border-border"><option>feat
                        </option><option>fix</option><option>chore</option><option>docs</option><option>
                        refactor</option></select></div>
                        <div className="w-3/4"><label className="block text-sm">Title</label><input
                        type="text" value={title} onChange={e => setTitle(e.target.value)}
                        className="w-full mt-1 p-2 rounded bg-surface border border-border"/></div>
                    </div>
                    <div><label className="block text-sm">Related Issue (optional)</label><input
                    type="text" value={relatedIssue} onChange={e => setRelatedIssue(e.target.value)}
                    className="w-full mt-1 p-2 rounded bg-surface border border-border"/></div>
                    <div><label className="block text-sm">Description</label><textarea
                    value={description} onChange={e => setDescription(e.target.value)}
                    className="w-full mt-1 p-2 rounded bg-surface border border-border resize-y
                    h-24"/></div>
                    <div><label className="block text-sm">Testing Steps</label><textarea
                    value={testingSteps} onChange={e => setTestingSteps(e.target.value)}
                    className="w-full mt-1 p-2 rounded bg-surface border border-border resize-y
                    h-24"/></div>
                </form>
                <div className="flex flex-col">
                    <div className="flex justify-between items-center mb-2">
                        <label className="text-sm font-medium text-text-secondary">Markdown
                        Preview</label>
                        <button onClick={handleCopy} className="px-3 py-1 bg-gray-100 text-xs rounded-md
                        hover:bg-gray-200">Copy Markdown</button>
                    </div>
                    <div className="relative flex-grow"><pre className="w-full h-full bg-background
                    border border-border p-4 rounded-md text-sm overflow-auto whitespace-pre-
                    wrap">{markdownPreview}</pre></div>
                </div>
            </div>
        </div>
    );
};

// ===== PrSummaryGenerator_2.tsx =====

import React, { useState, useCallback, useEffect } from 'react';
import * as Diff from 'diff';
import { generatePrSummaryStructured, downloadFile } from
'../../services/index.ts';
import type { StructuredPrSummary } from '../../types.ts';
import { GitBranchIcon, ArrowDownTrayIcon } from '../icons.tsx';
import { LoadingSpinner } from '../shared/index.tsx';
```

```
const exampleBefore = `function Greeter(props) {
  return <h1>Hello, {props.name}!</h1>;
}`;
const exampleAfter = `function Greeter({ name, enthusiasmLevel = 1 }) {
  const punctuation = '!'.repeat(enthusiasmLevel);
  return <h1>Hello, {name}{punctuation}</h1>;
}`;

export const PrSummaryGenerator: React.FC<{ beforeCode?: string, afterCode?:
string }> = ({ beforeCode: initialBefore, afterCode: initialAfter }) => {
    const [beforeCode, setBeforeCode] = useState<string>(initialBefore ||
    exampleBefore);
    const [afterCode, setAfterCode] = useState<string>(initialAfter ||
    exampleAfter);
    const [summary, setSummary] = useState<StructuredPrSummary | null>(null);
    const [isLoading, setIsLoading] = useState<boolean>(false);
    const [error, setError] = useState<string>('');

    const handleGenerate = useCallback(async (b: string, a: string) => {
        if (!b.trim() && !a.trim()) {
            setError('Please provide code to generate a summary.');
            return;
        }
        setIsLoading(true);
        setError('');
        setSummary(null);

        try {
            const diff = Diff.createPatch('component.tsx', b, a);
            const result = await generatePrSummaryStructured(diff);
            setSummary(result);
        } catch (err) {
            const errorMessage = err instanceof Error ? err.message : 'An unknown error
            occurred.';
            setError(`Failed to generate summary: ${errorMessage}`);
        } finally {
            setIsLoading(false);
        }
    }, []);

    const getMarkdownOutput = () => {
        if (!summary) return '';
        return `# ${summary.title}\n\n## Summary\n${summary.summary}\n\n## Key
        Changes\n${summary.changes.map(c => `- ${c}`).join('\n')}`;
    };

    useEffect(() => {
        if (initialBefore || initialAfter) {
            setBeforeCode(initialBefore || '');
            setAfterCode(initialAfter || '');
            handleGenerate(initialBefore || '', initialAfter || '');
        }
    }, [initialBefore, initialAfter, handleGenerate]);

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
            <header className="mb-6">
                <h1 className="text-3xl font-bold flex items-center">
                    <GitBranchIcon />
                    <span className="ml-3">AI PR Summary Generator</span>
                </h1>
                <p className="text-text-secondary mt-1">Paste 'before' and 'after' code snippets
```

```
        to generate a comprehensive PR summary.</p>
    </header>
    <div className="flex-grow grid grid-cols-1 lg:grid-cols-2 gap-6 min-h-0">
        <div className="flex flex-col gap-4 min-h-0">
            <div className="flex flex-col flex-1 min-h-0">
                <label htmlFor="before-code" className="text-sm font-medium text-text-secondary
                mb-2">Before</label>
                <textarea id="before-code" value={beforeCode} onChange={e =>
                setBeforeCode(e.target.value)} className="flex-grow p-4 bg-surface border
                border-border rounded-md resize-none font-mono text-sm" />
            </div>
             <div className="flex flex-col flex-1 min-h-0">
                <label htmlFor="after-code" className="text-sm font-medium text-text-secondary
                mb-2">After</label>
                <textarea id="after-code" value={afterCode} onChange={e =>
                setAfterCode(e.target.value)} className="flex-grow p-4 bg-surface border border-
                border rounded-md resize-none font-mono text-sm" />
            </div>
             <button onClick={() => handleGenerate(beforeCode, afterCode)}
             disabled={isLoading} className="btn-primary w-full flex items-center justify-
             center px-6 py-3">
                {isLoading ? <LoadingSpinner /> : 'Generate Summary'}
            </button>
        </div>
        <div className="flex flex-col min-h-0">
            <div className="flex justify-between items-center mb-2">
                <label className="text-sm font-medium text-text-secondary">Generated
                Summary</label>
                {summary && (
                    <div className="flex items-center gap-2">
                        <button onClick={() => navigator.clipboard.writeText(getMarkdownOutput())}
                        className="px-3 py-1 bg-gray-100 text-xs rounded-md hover:bg-gray-200">Copy
                        Markdown</button>
                        <button onClick={() => downloadFile(getMarkdownOutput(), 'PR_summary.md',
                        'text/markdown')} className="flex items-center gap-1 px-3 py-1 bg-gray-100 text-
                        xs rounded-md hover:bg-gray-200">
                            <ArrowDownTrayIcon className="w-4 h-4" /> Download
                        </button>
                    </div>
                )}
            </div>
            <div className="flex-grow p-4 bg-background border border-border rounded-md
            overflow-y-auto">
                {isLoading && <div className="flex items-center justify-center
                h-full"><LoadingSpinner /></div>}
                {error && <p className="text-red-500">{error}</p>}
                {summary && !isLoading && (
                    <div className="prose prose-sm max-w-none">
                        <h3 className="border-b border-border pb-2 mb-2">{summary.title}</h3>
                        <p>{summary.summary}</p>
                        <h4 className="mt-4">Key Changes:</h4>
                        <ul>
                            {summary.changes.map((change, i) => <li key={i}>{change}</li>)}
                        </ul>
                    </div>
                )}
                 {!isLoading && !summary && !error && <div className="text-text-secondary h-full
                 flex items-center justify-center">The summary will appear here.</div>}
            </div>
        </div>
    </div>
</div>
```

```
    );
};

// ===== ProjectExplorer.tsx =====

import React, { useState, useEffect, useCallback } from 'react';
import { useGlobalState } from '../../contexts/GlobalStateContext.tsx';
import type { FileNode } from '../../types.ts';
import { getRepoTree, getFileContent } from '../../services/index.ts';
import { LoadingSpinner } from '../shared/index.tsx';
import { FolderIcon, DocumentIcon } from '../icons.tsx';

// Recursive component to render the file tree
const FileTree: React.FC<{ node: FileNode, onFileSelect: (path: string) => void,
level?: number }> = ({ node, onFileSelect, level = 0 }) => {
    const [isOpen, setIsOpen] = useState(level < 2); // Auto-open first few levels
    const isFolder = node.type === 'folder';

    const handleToggle = () => {
        if (isFolder) {
            setIsOpen(!isOpen);
        }
    };

    const handleFileClick = (e: React.MouseEvent) => {
        e.stopPropagation();
        if (!isFolder) {
            onFileSelect(node.path);
        } else {
            handleToggle();
        }
    };

    return (
        <div className="text-sm">
            <div
                className="flex items-center p-1 rounded-md cursor-pointer hover:bg-gray-100"
                onClick={handleFileClick}
                style={{ paddingLeft: `${level * 16}px`}}
            >
                {isFolder ? <FolderIcon /> : <DocumentIcon />}
                <span className="ml-2 truncate">{node.name}</span>
            </div>
            {isFolder && isOpen && node.children && (
                <div>
                    {node.children.sort((a,b) => a.type === 'folder' ? -1 : 1).map(child => (
                        <FileTree key={child.path} node={child} onFileSelect={onFileSelect} level={level
                        + 1} />
                    ))}
                </div>
            )}
        </div>
    );
};

export const ProjectExplorer: React.FC = () => {
    const { state, dispatch } = useGlobalState();
    const { selectedRepo, projectFiles } = state;
    const [isLoading, setIsLoading] = useState(false);
    const [activeFileContent, setActiveFileContent] = useState<string | null>(null);
    const [activeFilePath, setActiveFilePath] = useState<string | null>(null);
    const [error, setError] = useState<string | null>(null);
```

```
useEffect(() => {
    if (selectedRepo && !projectFiles) {
        setIsLoading(true);
        setError(null);
        getRepoTree(selectedRepo.owner, selectedRepo.repo)
            .then(tree => dispatch({ type: 'LOAD_PROJECT_FILES', payload: tree }))
            .catch(err => {
                console.error(err);
                setError('Could not load repository file tree. It might be empty or you may not
                have access.');
            })
            .finally(() => setIsLoading(false));
    } else if (!selectedRepo && projectFiles) {
        // Clear files if repo is deselected
        dispatch({ type: 'LOAD_PROJECT_FILES', payload: null });
    }
}, [selectedRepo, projectFiles, dispatch]);

const handleFileSelect = useCallback(async (path: string) => {
    if (!selectedRepo) return;
    setIsLoading(true);
    setActiveFilePath(path);
    setActiveFileContent(null);
    setError(null);
    try {
        const content = await getFileContent(selectedRepo.owner, selectedRepo.repo,
        path);
        setActiveFileContent(content);
    } catch (e) {
        console.error(e);
        setError(`Could not load file: ${path}`);
        setActiveFileContent(null);
    } finally {
        setIsLoading(false);
    }
}, [selectedRepo]);

if (!selectedRepo) {
    return (
        <div className="h-full flex flex-col items-center justify-center text-center
        text-text-secondary p-4">
            <FolderIcon />
            <h2 className="text-lg font-semibold mt-2">No Repository Selected</h2>
            <p>Please select a repository in the 'Connections' tab to explore its files.</p>
        </div>
    );
}

if (isLoading && !projectFiles) {
    return (
        <div className="h-full flex items-center justify-center">
            <LoadingSpinner />
            <span className="ml-2 text-text-secondary">Loading repository tree...</span>
        </div>
    )
}

if (error && !projectFiles) {
     return (
        <div className="h-full flex flex-col items-center justify-center text-red-500
        p-4">
             <h2 className="text-lg font-semibold">Error Loading Repository</h2>
```

```
                <p>{error}</p>
            </div>
        );
    }

    return (
        <div className="h-full flex">
            <aside className="w-1/3 max-w-xs bg-surface border-r border-border p-2 overflow-
            y-auto">
                {projectFiles && <FileTree node={projectFiles} onFileSelect={handleFileSelect}
                />}
            </aside>
            <main className="flex-1 bg-background p-4 overflow-y-auto">
                 {isLoading && activeFilePath &&
                    <div className="h-full flex items-center justify-center">
                        <LoadingSpinner />
                        <span className="ml-2 text-text-secondary">Loading {activeFilePath}...</span>
                    </div>
                 }
                 {error && activeFilePath && <div className="p-4 text-red-500">{error}</div>}
                 {!isLoading && activeFileContent !== null && (
                    <div className="h-full flex flex-col">
                        <h3 className="font-mono text-sm mb-2 text-text-secondary flex-
                        shrink-0">{activeFilePath}</h3>
                        <div className="flex-grow bg-surface rounded-md border border-border overflow-
                        auto">
                            <pre className="p-4 text-sm whitespace-pre-wrap">
                                <code>{activeFileContent}</code>
                            </pre>
                        </div>
                    </div>
                 )}
                 {!isLoading && activeFileContent === null && !error && (
                     <div className="h-full flex items-center justify-center text-text-secondary">
                        <p>Select a file to view its content.</p>
                    </div>
                 )}
            </main>
        </div>
    );
};


// ===== ProjectMoodboard_2.tsx =====



import React, { useState } from 'react';
import { PhotoIcon } from '../icons.tsx';
import { useLocalStorage } from '../../hooks/useLocalStorage.ts';

interface Note {
    id: number;
    text: string;
    x: number;
    y: number;
    color: string;
}

const colors = ['bg-yellow-400', 'bg-green-400', 'bg-blue-400', 'bg-pink-400',
'bg-purple-400', 'bg-orange-400'];
const textColors = ['text-yellow-900', 'text-green-900', 'text-blue-900', 'text-
```

```
pink-900', 'text-purple-900', 'text-orange-900'];

export const ProjectMoodboard: React.FC = () => {
    const [notes, setNotes] = useLocalStorage<Note[]>('devcore_moodboard', []);
    const [dragging, setDragging] = useState<{ id: number; offsetX: number; offsetY:
    number } | null>(null);

    const addNote = () => {
        const newNote: Note = {
            id: Date.now(),
            text: 'New idea...',
            x: 50,
            y: 50,
            color: colors[notes.length % colors.length],
        };
        setNotes([...notes, newNote]);
    };

    const deleteNote = (id: number, e: React.MouseEvent) => {
        e.stopPropagation();
        setNotes(notes.filter((n) => n.id !== id));
    };

    const updateNote = (id: number, updates: Partial<Note>) => {
        setNotes(notes.map((n) => n.id === id ? { ...n, ...updates } : n));
    };

    const onMouseDown = (e: React.MouseEvent<HTMLDivElement>, id: number) => {
        const target = e.target as HTMLElement;
        if (target.tagName === 'TEXTAREA' || target.dataset.role === 'button') return;

        const noteElement = e.currentTarget;
        const rect = noteElement.getBoundingClientRect();
        setDragging({ id, offsetX: e.clientX - rect.left, offsetY: e.clientY - rect.top
        });
    };

    const onMouseMove = (e: React.MouseEvent<HTMLDivElement>) => {
        if (!dragging) return;
        const boardRect = e.currentTarget.getBoundingClientRect();
        updateNote(dragging.id, {
            x: e.clientX - dragging.offsetX - boardRect.left,
            y: e.clientY - dragging.offsetY - boardRect.top
        });
    };

    const onMouseUp = () => setDragging(null);

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
            <header className="mb-6 flex justify-between items-center">
                 <div>
                    <h1 className="text-3xl font-bold flex items-center"><PhotoIcon /><span
                    className="ml-3">Project Moodboard</span></h1>
                    <p className="text-text-secondary mt-1">Organize your ideas with interactive
                    sticky notes.</p>
                </div>
                <button onClick={addNote} className="btn-primary px-6 py-2">Add Note</button>
            </header>
            <div
                className="relative flex-grow bg-background border-2 border-dashed border-border
                rounded-lg overflow-hidden"
```

```
                onMouseMove={onMouseMove} onMouseUp={onMouseUp} onMouseLeave={onMouseUp}
            >
                {notes.map((note) => (
                    <div
                        key={note.id}
                        className={`group absolute w-56 h-56 p-2 flex flex-col shadow-lg cursor-grab
                        active:cursor-grabbing rounded-md transition-transform duration-100 border
                        border-black/40 ${note.color} ${textColors[colors.indexOf(note.color)]}`}
                        style={{ top: note.y, left: note.x, transform: dragging?.id === note.id ?
                        'scale(1.05)' : 'scale(1)' }}
                        onMouseDown={e => onMouseDown(e, note.id)}
                    >
                        <button data-role="button" onClick={(e) => deleteNote(note.id, e)}
                        className="absolute -top-2 -right-2 w-6 h-6 rounded-full bg-gray-700 text-white
                        font-bold text-xs flex items-center justify-center opacity-0 group-
                        hover:opacity-100 hover:bg-red-500 transition-all">&times;</button>
                        <textarea
                            value={note.text}
                            onChange={(e) => updateNote(note.id, { text: e.target.value })}
                            className="w-full h-full bg-transparent resize-none focus:outline-none font-
                            medium p-1"
                        />
                        <div data-role="button" className="flex-shrink-0 flex justify-center gap-1 p-1
                        opacity-0 group-hover:opacity-100 transition-opacity">
                            {colors.map((c, i) => <button key={c} onClick={() => updateNote(note.id, {
                            color: c })} className={`w-4 h-4 rounded-full ${c} border border-black/20
                            ${note.color === c ? 'ring-2 ring-offset-1 ring-black/50' : ''}`}/>)}
                        </div>
                    </div>
                ))}
            </div>
        </div>
    );
};

// ===== PromptCraftPad_2.tsx =====

import React, { useState, useEffect, useMemo } from 'react';
import { SparklesIcon } from '../icons.tsx';
import { useLocalStorage } from '../../hooks/useLocalStorage.ts';

interface Prompt {
    id: number;
    name: string;
    text: string;
}

export const PromptCraftPad: React.FC = () => {
    const [prompts, setPrompts] = useLocalStorage<Prompt[]>('devcore_prompts', [
        { id: 1, name: 'React Component Generator', text: 'Generate a React component
        named {name} that {description}. Style it with Tailwind CSS.'}
    ]);
    const [activePrompt, setActivePrompt] = useState<Prompt | null>(prompts[0] ||
    null);
    const [editingId, setEditingId] = useState<number | null>(null);
    const [tempName, setTempName] = useState('');
    const [variables, setVariables] = useState<Record<string, string>>({});

    const variableNames = useMemo(() => {
        if (!activePrompt) return [];
        return [...activePrompt.text.matchAll(/\{(\w+)\}/g)].map(match => match[1]);
    }, [activePrompt]);
```

```
const renderedPrompt = useMemo(() => {
    if (!activePrompt) return '';
    return variableNames.reduce((acc, varName) => {
        return acc.replace(new RegExp(`\\{${varName}\\}`, 'g'), variables[varName] ||
        `{${varName}}`);
    }, activePrompt.text);
}, [activePrompt, variables, variableNames]);

useEffect(() => {
    if(!activePrompt && prompts.length > 0) setActivePrompt(prompts[0]);
    if (activePrompt) setActivePrompt(prompts.find((p: Prompt) => p.id ===
    activePrompt.id) || null);
}, [prompts, activePrompt]);

const handleTextChange = (e: React.ChangeEvent<HTMLTextAreaElement>) => {
    if (!activePrompt) return;
    const updatedPrompt = { ...activePrompt, text: e.target.value };
    setPrompts(prompts.map((p: Prompt) => p.id === updatedPrompt.id ? updatedPrompt
    : p));
};

const handleNameUpdate = (id: number, newName: string) => {
    setPrompts(prompts.map((p: Prompt) => p.id === id ? {...p, name: newName} : p));
    setEditingId(null);
};

const handleAddNew = () => {
    const newPrompt = { id: Date.now(), name: 'New Untitled Prompt', text: '' };
    setPrompts([...prompts, newPrompt]);
    setActivePrompt(newPrompt);
};

const handleDelete = (id: number) => {
    setPrompts(prompts.filter((p: Prompt) => p.id !== id));
    if(activePrompt?.id === id) setActivePrompt(prompts.length > 1 ? prompts[0] :
    null);
}

return (
    <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
        <header className="mb-6"><h1 className="text-3xl font-bold flex items-
        center"><SparklesIcon /><span className="ml-3">Prompt Craft Pad</span></h1><p
        className="text-text-secondary mt-1">Create, save, and manage your favorite AI
        prompts.</p></header>
        <div className="flex-grow flex gap-6 min-h-0">
            <aside className="w-1/3 bg-surface border border-border p-4 rounded-lg flex
            flex-col">
                <h3 className="font-bold mb-2">My Prompts</h3>
                <ul className="space-y-2 flex-grow overflow-y-auto">{prompts.map((p: Prompt) =>
                (<li key={p.id} className="group flex items-center justify-between"><div
                className={`w-full text-left rounded-md ${activePrompt?.id === p.id ? 'bg-
                primary/10' : ''}`}><button onClick={() => setActivePrompt(p)} onDoubleClick={()
                => {setEditingId(p.id); setTempName(p.name);}} className={`w-full text-left px-3
                py-2 ${activePrompt?.id === p.id ? 'text-primary' : 'hover:bg-gray-100'}`}>
                {editingId === p.id ? <input autoFocus value={tempName} onChange={e =>
                setTempName(e.target.value)} onBlur={() => handleNameUpdate(p.id, tempName)}
                onKeyDown={e => e.key === 'Enter' && handleNameUpdate(p.id, tempName)}
                className="bg-gray-100 text-text-primary w-full"/> : p.name}
                </button></div><button onClick={() => handleDelete(p.id)} className="ml-2 p-1
                text-text-secondary hover:text-red-500 opacity-0 group-
                hover:opacity-100">&times;</button></li>))}</ul>
                <div className="mt-4 pt-4 border-t border-border"><button onClick={handleAddNew}
```

```
                    className="btn-primary w-full text-sm py-2">Add New Prompt</button></div>
                </aside>
                <main className="w-2/3 flex flex-col gap-4">
                    {activePrompt ? (<>
                        <textarea value={activePrompt.text} onChange={handleTextChange} className="flex-
                        grow p-4 bg-surface border border-border rounded-md resize-none font-mono text-
                        sm focus:ring-2 focus:ring-primary focus:outline-none"/>
                        {variableNames.length > 0 && <div className="flex-shrink-0 bg-surface border
                        border-border p-4 rounded-lg"><h4 className="font-bold mb-2">Test
                        Variables</h4><div className="grid grid-cols-2 gap-2">{variableNames.map(v =>
                        (<div key={v}><label className="text-xs">{v}</label><input type="text"
                        value={variables[v] || ''} onChange={e => setVariables({...variables, [v]:
                        e.target.value})} className="w-full bg-background border border-border px-2 py-1
                        rounded text-sm"/></div>))}</div><h4 className="font-bold mt-4 mb-2">Live
                        Preview</h4><p className="text-sm p-2 bg-background rounded border border-
                        border">{renderedPrompt}</p></div>}
                    </>) : (<div className="flex-grow flex items-center justify-center bg-background
                    rounded-lg text-text-secondary border border-border">Select a prompt or create a
                    new one.</div>)}
                </main>
            </div>
        </div>
    );
};

// ===== PwaManifestEditor_2.tsx =====

import React, { useState, useMemo } from 'react';
import { CodeBracketSquareIcon, ArrowDownTrayIcon } from '../icons.tsx';
import { downloadFile } from '../../services/fileUtils.ts';

interface ManifestData {
    name: string;
    short_name: string;
    start_url: string;
    scope: string;
    display: 'standalone' | 'fullscreen' | 'minimal-ui' | 'browser';
    orientation: 'any' | 'natural' | 'landscape' | 'portrait';
    background_color: string;
    theme_color: string;
}

const HomeScreenPreview: React.FC<{ manifest: ManifestData }> = ({ manifest })
=> (
    <div className="w-full max-w-xs mx-auto flex flex-col items-center">
        <div className="w-72 h-[550px] bg-gray-800 rounded-[40px] border-[10px] border-
        black shadow-2xl p-4 flex flex-col">
            <div className="flex-shrink-0 h-6 flex justify-between items-center px-4">
                <span className="text-xs font-bold" style={{color:
                manifest.theme_color}}>9:41</span>
                <div className="w-16 h-4 bg-black rounded-full" />
                <span className="text-xs font-bold" style={{color:
                manifest.theme_color}}>100%</span>
            </div>
            <div className="flex-grow grid grid-cols-4 gap-4 p-4">
                <div className="flex flex-col items-center gap-1">
                    <div className="w-14 h-14 bg-white rounded-xl flex items-center justify-center
                    text-3xl" style={{backgroundColor: manifest.background_color}}>
                        <span style={{color: manifest.theme_color}}>{manifest.short_name[0]}</span>
                    </div>
                    <p className="text-xs text-center text-white truncate
                    w-full">{manifest.short_name}</p>
```

```
                </div>
            </div>
        </div>
         <p className="text-xs text-text-secondary mt-2 text-center">Home Screen
         Preview</p>
    </div>
);


export const PwaManifestEditor: React.FC = () => {
    const [manifest, setManifest] = useState<ManifestData>({
        name: 'DevCore Progressive Web App', short_name: 'DevCore', start_url: '/',
        scope: '/',
        display: 'standalone', orientation: 'any', background_color: '#F5F7FA',
        theme_color: '#0047AB',
    });

    const handleChange = (e: React.ChangeEvent<HTMLInputElement |
    HTMLSelectElement>) => {
        setManifest({ ...manifest, [e.target.name]: e.target.value });
    };

    const generatedJson = useMemo(() => {
        const fullManifest = { ...manifest, icons: [{"src": "icon-192.png", "type":
        "image/png", "sizes": "192x192"}, {"src": "icon-512.png", "type": "image/png",
        "sizes": "512x512"}] };
        return JSON.stringify(fullManifest, null, 2);
    }, [manifest]);

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
            <header className="mb-6"><h1 className="text-3xl font-bold flex items-
            center"><CodeBracketSquareIcon /><span className="ml-3">PWA Manifest
            Editor</span></h1><p className="text-text-secondary mt-1">Configure and generate
            the `manifest.json` file for your PWA.</p></header>
            <div className="flex-grow grid grid-cols-1 lg:grid-cols-2 xl:grid-cols-3 gap-6
            min-h-0">
                <div className="xl:col-span-1 flex flex-col gap-4 bg-surface border border-
                border p-6 rounded-lg overflow-y-auto">
                    <h3 className="text-xl font-bold">Configuration</h3>
                    <div><label className="block text-sm">App Name</label><input type="text"
                    name="name" value={manifest.name} onChange={handleChange} className="w-full mt-1
                    p-2 rounded bg-background border border-border"/></div>
                    <div><label className="block text-sm">Short Name</label><input type="text"
                    name="short_name" value={manifest.short_name} onChange={handleChange}
                    className="w-full mt-1 p-2 rounded bg-background border border-border"/></div>
                    <div><label className="block text-sm">Start URL</label><input type="text"
                    name="start_url" value={manifest.start_url} onChange={handleChange}
                    className="w-full mt-1 p-2 rounded bg-background border border-border"/></div>
                    <div><label className="block text-sm">Scope</label><input type="text"
                    name="scope" value={manifest.scope} onChange={handleChange} className="w-full
                    mt-1 p-2 rounded bg-background border border-border"/></div>
                    <div><label className="block text-sm">Display Mode</label><select name="display"
                    value={manifest.display} onChange={handleChange} className="w-full mt-1 p-2
                    rounded bg-background border border-
                    border"><option>standalone</option><option>fullscreen</option><option>minimal-
                    ui</option><option>browser</option></select></div>
                    <div><label className="block text-sm">Orientation</label><select
                    name="orientation" value={manifest.orientation} onChange={handleChange}
                    className="w-full mt-1 p-2 rounded bg-background border border-border"><option>a
                    ny</option><option>natural</option><option>landscape</option><option>portrait</o
                    ption></select></div>
```

```
                    <div className="flex gap-4">
                       <div className="w-1/2"><label className="block text-sm">Background
                       Color</label><input type="color" name="background_color"
                       value={manifest.background_color} onChange={handleChange} className="w-full mt-1
                       h-10 rounded bg-background border border-border"/></div>
                       <div className="w-1/2"><label className="block text-sm">Theme
                       Color</label><input type="color" name="theme_color" value={manifest.theme_color}
                       onChange={handleChange} className="w-full mt-1 h-10 rounded bg-background border
                       border-border"/></div>
                    </div>
                </div>
                <div className="xl:col-span-1 flex flex-col">
                    <div className="flex justify-between items-center mb-2">
                        <label className="text-sm font-medium text-text-secondary">Generated
                        manifest.json</label>
                        <button onClick={() => downloadFile(generatedJson, 'manifest.json',
                        'application/json')} className="flex items-center gap-1 px-3 py-1 bg-gray-100
                        text-xs rounded-md hover:bg-gray-200">
                           <ArrowDownTrayIcon className="w-4 h-4"/> Download
                       </button>
                    </div>
                    <div className="relative flex-grow"><pre className="w-full h-full bg-background
                    p-4 rounded-md text-primary text-sm overflow-auto">{generatedJson}</pre></div>
                </div>
                <div className="hidden xl:flex flex-col items-center justify-center">
                    <label className="text-sm font-medium text-text-secondary mb-2">Live
                    Preview</label>
                    <HomeScreenPreview manifest={manifest} />
                </div>
            </div>
        </div>
    );
};

// ===== RegexSandbox_2.tsx =====

import React, { useState, useMemo, useCallback, useEffect } from 'react';
import { generateRegExStream } from '../../services/geminiService.ts';
import { BeakerIcon } from '../icons.tsx';
import { LoadingSpinner } from '../shared/index.tsx';

const commonPatterns = [
    { name: 'Email', pattern: '/[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\\.[a-zA-Z]{2,}/g'
    },
    { name: 'URL', pattern: '/https?:\\/\\/(www\\.)?[-a-zA-Z0-
    9@:%._\\+~#=]{1,256}\\.[a-zA-Z0-9()]{1,6}\\b([-a-zA-Z0-9()@:%_\\+.~#?&//=]*)/g'
    },
    { name: 'IPv4 Address', pattern:
    '/((25[0-5]|(2[0-4]|1\\d|[1-9]|)\\d)\\.?\\b){4}/g' },
    { name: 'Date (YYYY-MM-DD)', pattern: '/\\d{4}-\\d{2}-\\d{2}/g' },
];

const CheatSheet = () => (
    <div className="bg-surface border border-border p-4 rounded-lg">
        <h3 className="text-lg font-bold mb-2">Regex Cheat Sheet</h3>
        <div className="grid grid-cols-2 gap-x-4 gap-y-1 text-xs font-mono">
            <p><span className="text-primary">.</span> - Any character</p>
            <p><span className="text-primary">\d</span> - Any digit</p>
            <p><span className="text-primary">\w</span> - Word character</p>
            <p><span className="text-primary">\s</span> - Whitespace</p>
            <p><span className="text-primary">[abc]</span> - a, b, or c</p>
            <p><span className="text-primary">[^abc]</span> - Not a, b, or c</p>
```

```
            <p><span className="text-primary">*</span> - 0 or more</p>
            <p><span className="text-primary">+</span> - 1 or more</p>
            <p><span className="text-primary">?</span> - 0 or one</p>
            <p><span className="text-primary">^</span> - Start of string</p>
            <p><span className="text-primary">$</span> - End of string</p>
            <p><span className="text-primary">\b</span> - Word boundary</p>
        </div>
    </div>
);

export const RegexSandbox: React.FC<{ initialPrompt?: string }> = ({
initialPrompt }) => {
    const [pattern, setPattern] =
    useState<string>('/\\b([A-Z][a-z]+)\\s(\\w+)\\b/g');
    const [testString, setTestString] = useState<string>('The quick Brown Fox jumps
    over the Lazy Dog.');
    const [aiPrompt, setAiPrompt] = useState<string>(initialPrompt || 'find
    capitalized words and the word after');
    const [isAiLoading, setIsAiLoading] = useState<boolean>(false);

    const { matches, error } = useMemo(() => {
        try {
            const patternParts = pattern.match(/^\/(.*)\/([gimyus]*)$/);
            if (!patternParts) return { matches: null, error: 'Invalid regex literal. Use
            /pattern/flags.' };
            const [, regexBody, regexFlags] = patternParts;
            const regex = new RegExp(regexBody, regexFlags);
            return { matches: [...testString.matchAll(regex)], error: null };
        } catch (e) { return { matches: null, error: e instanceof Error ? e.message :
        'Unknown error.' }; }
    }, [pattern, testString]);

    const handleGenerateRegex = useCallback(async (p: string) => {
        if (!p) return;
        setIsAiLoading(true);
        try {
            const stream = generateRegExStream(p);
            let fullResponse = '';
            for await (const chunk of stream) { fullResponse += chunk; }
            setPattern(fullResponse.trim().replace(/^`+|`+$/g, ''));
        } finally { setIsAiLoading(false); }
    }, []);

    useEffect(() => { if (initialPrompt) handleGenerateRegex(initialPrompt); },
    [initialPrompt, handleGenerateRegex]);

    const highlightedString = useMemo(() => {
        if (!matches || matches.length === 0 || error) return testString;
        let lastIndex = 0;
        const parts: (string | JSX.Element)[] = [];
        matches.forEach((match, i) => {
            if (match.index === undefined) return;
            parts.push(testString.substring(lastIndex, match.index));
            parts.push(<mark key={i} className="bg-primary/20 text-primary rounded
            px-1">{match[0]}</mark>);
            lastIndex = match.index + match[0].length;
        });
        parts.push(testString.substring(lastIndex));
        return parts;
    }, [matches, testString, error]);

    return (
```

```
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
            <header className="mb-6"><h1 className="text-3xl font-bold flex items-
            center"><BeakerIcon /><span className="ml-3">RegEx Sandbox</span></h1><p
            className="text-text-secondary mt-1">Test your regular expressions and generate
            them with AI.</p></header>
            <div className="flex-grow grid grid-cols-1 lg:grid-cols-3 gap-6 min-h-0">
                <div className="lg:col-span-2 flex flex-col gap-4">
                    <div className="flex gap-2"><input type="text" value={aiPrompt} onChange={(e) =>
                    setAiPrompt(e.target.value)} placeholder="Describe the pattern to find..."
                    className="flex-grow px-3 py-1.5 rounded-md bg-surface border border-border
                    text-sm focus:ring-2 focus:ring-primary" /><button onClick={() =>
                    handleGenerateRegex(aiPrompt)} disabled={isAiLoading} className="btn-primary
                    px-4 py-1.5 flex items-center">{isAiLoading ? <LoadingSpinner/> :
                    'Generate'}</button></div>
                    <div><label htmlFor="regex-pattern" className="text-sm font-medium text-text-
                    secondary">Regular Expression</label><input id="regex-pattern" type="text"
                    value={pattern} onChange={(e) => setPattern(e.target.value)} className={`w-full
                    mt-1 px-3 py-2 rounded-md bg-surface border ${error ? 'border-red-500' :
                    'border-border'} font-mono text-sm focus:ring-2 focus:ring-primary`} />{error &&
                    <p className="text-red-500 text-xs mt-1">{error}</p>}</div>
                    <div className="flex flex-col flex-grow min-h-0"><label htmlFor="test-string"
                    className="text-sm font-medium text-text-secondary">Test String</label><textarea
                    id="test-string" value={testString} onChange={(e) =>
                    setTestString(e.target.value)} className="w-full mt-1 p-3 rounded-md bg-surface
                    border border-border font-mono text-sm resize-y h-32" /><div className="mt-2 p-3
                    bg-background rounded-md border border-border min-h-[50px] whitespace-pre-
                    wrap">{highlightedString}</div></div>
                    <div className="flex-shrink-0"><h3 className="text-lg font-bold">Match Groups
                    ({matches?.length || 0})</h3><div className="mt-2 p-2 bg-surface rounded-md
                    overflow-y-auto max-h-48 font-mono text-xs border border-border">{matches &&
                    matches.length > 0 ? (matches.map((match, i) => (<details key={i} className="p-2
                    border-b border-border"><summary className="cursor-pointer text-green-700">Match
                    {i + 1}: "{match[0]}"</summary><div className="pl-4
                    mt-1">{Array.from(match).map((group, gIndex) => <p key={gIndex} className="text-
                    text-secondary">Group {gIndex}: <span className="text-
                    amber-700">{String(group)}</span></p>)}</div></details>))) : (<p
                    className="text-text-secondary text-sm p-2">No matches found.</p>)}</div></div>
                </div>
                <div className="lg:col-span-1 space-y-4">
                    <CheatSheet />
                    <div className="bg-surface border border-border p-4 rounded-lg">
                        <h3 className="text-lg font-bold mb-2">Common Patterns</h3>
                        <div className="flex flex-col items-start gap-2">
                            {commonPatterns.map(p => (
                                <button key={p.name} onClick={() => setPattern(p.pattern)} className="text-left
                                text-sm text-primary hover:underline">
                                    {p.name}
                                </button>
                            ))}
                        </div>
                    </div>
                </div>
            </div>
        </div>
    );
};

// ===== ResponsiveTester_2.tsx =====

import React, { useState, useEffect } from 'react';
import { EyeIcon } from '../icons.tsx';
const devices = {
```

```
    'iPhone 12': { width: 390, height: 844 },
    'Pixel 5': { width: 393, height: 851 },
    'iPad Air': { width: 820, height: 1180 },
    'Surface Duo': { width: 540, height: 720 },
    'Laptop': { width: 1366, height: 768 },
    'Desktop': { width: 1920, height: 1080 },
    'Auto': { width: '100%', height: '100%' },
};

type DeviceName = keyof typeof devices;

export const ResponsiveTester: React.FC = () => {
    const [url, setUrl] = useState('https://react.dev');
    const [displayUrl, setDisplayUrl] = useState(url);
    const [size, setSize] = useState<{width: number | string, height: number |
    string}>(devices['Auto']);

    useEffect(() => {
        const handleResize = () => {
            if (size.width === '100%') {
                setSize({ width: '100%', height: '100%' });
            }
        };
        window.addEventListener('resize', handleResize);
        return () => window.removeEventListener('resize', handleResize);
    }, [size.width]);

    const handleUrlSubmit = (e: React.FormEvent) => {
        e.preventDefault();
        setDisplayUrl(url.startsWith('http') ? url : `https://${url}`);
    };

    const handleRotate = () => {
        if(typeof size.width === 'number' && typeof size.height === 'number') {
            setSize({ width: size.height, height: size.width });
        }
    };

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
            <header className="mb-6"><h1 className="text-3xl font-bold flex items-
            center"><EyeIcon /><span className="ml-3">Responsive Tester</span></h1><p
            className="text-text-secondary mt-1">Preview your web pages at different screen
            sizes.</p></header>
            <form onSubmit={handleUrlSubmit} className="flex items-center gap-2 mb-2">
                <input type="text" value={url} onChange={(e) => setUrl(e.target.value)}
                placeholder="https://example.com" className="flex-grow px-4 py-2 rounded-md bg-
                surface border border-border focus:ring-2 focus:ring-primary focus:outline-
                none"/>
                <button type="submit" className="btn-primary px-6 py-2">Load</button>
            </form>
            <div className="bg-surface p-2 rounded-lg flex flex-wrap justify-center items-
            center gap-2 mb-4 border border-border">
                {Object.keys(devices).map(name => (
                    <button key={name} onClick={() => setSize(devices[name as DeviceName])}
                    className={`px-3 py-1 rounded-md text-sm ${JSON.stringify(size) ===
                    JSON.stringify(devices[name as DeviceName]) ? 'bg-primary/10 text-primary font-
                    semibold' : 'hover:bg-gray-100'}`}>{name}</button>
                ))}
                <div className="flex items-center gap-1 ml-4">
                    <input type="number" value={typeof size.width === 'number' ? size.width : ''}
                    onChange={e => setSize({ ...size, width: Number(e.target.value) })}
```

```
                    className="w-20 px-2 py-1 bg-gray-100 border border-border rounded-md text-sm"/>
                    <span className="text-sm text-text-secondary">x</span>
                    <input type="number" value={typeof size.height === 'number' ? size.height : ''}
                    onChange={e => setSize({ ...size, height: Number(e.target.value) })}
                    className="w-20 px-2 py-1 bg-gray-100 border border-border rounded-md text-sm"/>
                </div>
                 <button onClick={handleRotate} className="px-3 py-1 rounded-md text-sm hover:bg-
                 gray-100" title="Rotate">■</button>
            </div>
            <div className="flex-grow bg-background rounded-lg p-4 overflow-auto border
            border-border">
                <iframe key={displayUrl} src={displayUrl} style={{ width: size.width, height:
                size.height }} className="bg-white border-4 border-gray-300 rounded-md
                transition-all duration-300 shadow-lg mx-auto" title="Responsive Preview"/>
            </div>
        </div>
    );
};

// ===== SassScssCompiler_2.tsx =====

import React, { useState, useMemo } from 'react';
import { CodeBracketSquareIcon } from '../icons.tsx';

const initialScss = `$primary-color: #0047AB;
$font-size: 16px;

.container {
  padding: 20px;
  background-color: #f0f0f0;

  .title {
    color: $primary-color;
    font-size: $font-size * 1.5;

    &:hover {
      text-decoration: underline;
    }
  }

  > p {
    margin-top: 10px;
  }
}`;

const escapeRegExp = (string: string): string => {
    // $& means the whole matched string
    return string.replace(/[.*+?^${}()|[\]\\]/g, '\\$&');
};

const compileScss = (scss: string): string => {
    try {
        let css = scss;
        css = css.replace(/\/\/.*$/gm, '');

        const variables: Record<string, string> = {};
        css = css.replace(/\$([\w-]+):\s*(.*?);/g, (_, name, value) => {
            variables[name] = value.trim(); return '';
        });

        for (let i = 0; i < 5; i++) {
            Object.entries(variables).forEach(([name, value]) => {
```

```
                css = css.replace(new RegExp(`\\$${escapeRegExp(name)}`, 'g'), value);
            });
        }

        css = css.replace(/([\d.]+)(px|rem|em|%)\s*([*\/])\s*([\d.]+)/g, (_, n1, unit,
        op, n2) => {
            const num1 = parseFloat(n1); const num2 = parseFloat(n2);
            const result = op === '*' ? num1 * num2 : num1 / num2;
            return `${result}${unit}`;
        });

        const processBlock = (block: string, parentSelector: string = ''): string => {
            let currentCss = '';
            let nestedCss = '';
            const properties = [];

            const regex =
            /((?:[\w-:.#&>+~*\s,]+|\([^)]*\))\s*\{[^{}]*\}|((?:[\w-]+\s*:[^;]+;))/g;
            const content = block.substring(block.indexOf('{') + 1, block.lastIndexOf('}'));
            let match;
            while ((match = regex.exec(content)) !== null) {
                if (match[1]) {
                    const nestedSelector = match[1].substring(0, match[1].indexOf('{')).trim();
                    const fullSelector = nestedSelector.includes('&') ? nestedSelector.replace(/&/g,
                    parentSelector) : `${parentSelector} ${nestedSelector}`.trim();
                    nestedCss += processBlock(match[1], fullSelector);
                } else if (match[2]) {
                    properties.push(`  ${match[2].trim()}`);
                }
            }

            if (properties.length > 0) {
                currentCss = `${parentSelector} {\n${properties.join('\n')}\n}\n`;
            }

            return currentCss + nestedCss;
        };

        let result = processBlock(`root{${css}}`, '').trim();
        return result.replace(/root\s*\{\s*\}/, '').trim();

    } catch(e) {
        console.error("SCSS Compilation Error:", e);
        return "/* Error compiling SCSS. Check console for details. */";
    }
};


export const SassScssCompiler: React.FC = () => {
    const [scss, setScss] = useState(initialScss);
    const compiledCss = useMemo(() => compileScss(scss), [scss]);

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
            <header className="mb-6">
                <h1 className="text-3xl flex items-center"><CodeBracketSquareIcon /><span
                className="ml-3">SASS/SCSS Compiler</span></h1>
                <p className="text-text-secondary mt-1">A real-time SASS/SCSS to CSS
                compiler.</p>
            </header>
            <div className="flex-grow flex flex-col gap-4 min-h-0">
                <div className="flex flex-col flex-1 min-h-0">
```

```
                    <label htmlFor="scss-input" className="text-sm font-medium text-text-secondary
                    mb-2">SASS/SCSS Input</label>
                    <textarea id="scss-input" value={scss} onChange={(e) => setScss(e.target.value)}
                    className="flex-grow p-4 bg-surface border border-border rounded-md resize-y
                    font-mono text-sm text-pink-600" spellCheck="false" />
                </div>
                <div className="flex flex-col flex-1 min-h-0">
                    <label className="text-sm font-medium text-text-secondary mb-2">Compiled CSS
                    Output</label>
                    <pre className="flex-grow p-4 bg-background border border-border rounded-md
                    overflow-y-auto text-blue-700 font-mono text-sm whitespace-pre-
                    wrap">{compiledCss}</pre>
                </div>
            </div>
        </div>
    );
};

// ===== SchemaDesigner_2.tsx =====

import React, { useState, useRef } from 'react';
import { MapIcon, ArrowDownTrayIcon } from '../icons.tsx';
import { downloadFile } from '../../services/fileUtils.ts';

interface Column { id: number; name: string; type: string; }
interface Table { id: number; name: string; columns: Column[]; x: number; y:
number; }

const exportToSQL = (tables: Table[]) => {
    return tables.map(table => {
        const columnsSQL = table.columns.map(col => `  "${col.name}"
        ${col.type.toUpperCase()}`).join(',\n');
        return `CREATE TABLE "${table.name}" (\n${columnsSQL}\n);`;
    }).join('\n\n');
};

export const SchemaDesigner: React.FC = () => {
    const [tables, setTables] = useState<Table[]>([
        { id: 1, name: 'users', columns: [{ id: 1, name: 'id', type: 'INTEGER PRIMARY
        KEY' }, {id: 2, name: 'username', type: 'VARCHAR(255)'}], x: 50, y: 50 },
        { id: 2, name: 'posts', columns: [{ id: 1, name: 'id', type: 'INTEGER PRIMARY
        KEY' }, {id: 2, name: 'user_id', type: 'INTEGER'}, {id: 3, name: 'content',
        type: 'TEXT'}], x: 300, y: 100 },
    ]);
    const [dragging, setDragging] = useState<{ id: number; offsetX: number; offsetY:
    number } | null>(null);
    const canvasRef = useRef<HTMLDivElement>(null);

    const onMouseDown = (e: React.MouseEvent<HTMLDivElement>, id: number) => {
        const tableElement = e.currentTarget;
        const rect = tableElement.getBoundingClientRect();
        setDragging({ id, offsetX: e.clientX - rect.left, offsetY: e.clientY - rect.top
        });
    };

    const onMouseMove = (e: React.MouseEvent<HTMLDivElement>) => {
        if (!dragging || !canvasRef.current) return;
        const canvasRect = canvasRef.current.getBoundingClientRect();
        setTables(tables.map(t => t.id === dragging.id ? { ...t, x: e.clientX -
        dragging.offsetX - canvasRect.left + canvasRef.current.scrollLeft, y: e.clientY
        - dragging.offsetY - canvasRect.top + canvasRef.current.scrollTop } : t));
    };
```

```
    const onMouseUp = () => setDragging(null);

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
            <header className="mb-6"><h1 className="text-3xl font-bold flex items-
            center"><MapIcon /><span className="ml-3">Schema Designer</span></h1><p
            className="text-text-secondary mt-1">Visually design your database schema with
            drag-and-drop.</p></header>
            <div className="flex-grow flex gap-6 min-h-0">
                <main ref={canvasRef} className="flex-grow relative bg-background rounded-lg
                border-2 border-dashed border-border overflow-auto" onMouseMove={onMouseMove}
                onMouseUp={onMouseUp} onMouseLeave={onMouseUp}>
                    {tables.map(table => (
                        <div key={table.id} className={`absolute w-64 bg-surface rounded-lg shadow-xl
                        border cursor-grab active:cursor-grabbing ${dragging?.id === table.id ? 'border-
                        primary' : 'border-border'}`} style={{ top: table.y, left: table.x }}
                        onMouseDown={e => onMouseDown(e, table.id)}>
                            <h3 className="font-bold text-primary text-lg p-2 bg-gray-50 rounded-t-lg
                            border-b border-border">{table.name}</h3>
                            <div className="p-2 space-y-1 font-mono text-xs">
                                {table.columns.map(col => (<div key={col.id} className="flex justify-between
                                items-center"><span className="text-text-primary">{col.name}</span><span
                                className="text-text-secondary">{col.type}</span></div>))}
                            </div>
                        </div>
                    ))}
                </main>
                <aside className="w-80 flex-shrink-0 flex flex-col gap-4">
                    <div className="flex flex-col gap-2">
                         <button onClick={() => downloadFile(JSON.stringify(tables, null, 2),
                         'schema.json', 'application/json')} className="flex-1 text-sm py-2 bg-gray-100
                         border border-border rounded-md flex items-center justify-center gap-2 hover:bg-
                         gray-200">
                            <ArrowDownTrayIcon className="w-4 h-4"/> Download JSON
                        </button>
                         <button onClick={() => downloadFile(exportToSQL(tables), 'schema.sql',
                         'application/sql')} className="btn-primary flex-1 text-sm py-2 flex items-center
                         justify-center gap-2">
                            <ArrowDownTrayIcon className="w-4 h-4"/> Download SQL
                         </button>
                    </div>
                    <div className="flex-grow bg-surface border border-border p-4 rounded-lg
                    overflow-y-auto">
                        <h3 className="font-bold mb-2">Editor</h3>
                        <p className="text-xs text-text-secondary">Schema editing coming soon!</p>
                    </div>
                </aside>
            </div>
        </div>
    );
};

// ===== ScreenshotToComponent_2.tsx =====

import React, { useState, useCallback, useRef } from 'react';
import { generateComponentFromImageStream } from
'../../services/geminiService.ts';
import { PhotoIcon, ArrowDownTrayIcon } from '../icons.tsx';
import { LoadingSpinner, MarkdownRenderer } from '../shared/index.tsx';
import { fileToBase64, blobToDataURL, downloadFile } from
'../../services/fileUtils.ts';
export const ScreenshotToComponent: React.FC = () => {
```

```
const [previewImage, setPreviewImage] = useState<string | null>(null);
const [rawCode, setRawCode] = useState('');
const [isLoading, setIsLoading] = useState(false);
const [error, setError] = useState('');
const fileInputRef = useRef<HTMLInputElement>(null);

const handleGenerate = async (base64Image: string) => {
    setIsLoading(true);
    setError('');
    setRawCode('');
    try {
        const stream = generateComponentFromImageStream(base64Image);
        let fullResponse = '';
        for await (const chunk of stream) {
            fullResponse += chunk;
            setRawCode(fullResponse.replace(/^```(?:\w+\n)?/, '').replace(/```$/, ''));
        }
    } catch (err) {
        setError(err instanceof Error ? err.message : 'An unknown error occurred.');
    } finally {
        setIsLoading(false);
    }
};

const processImageBlob = async (blob: Blob) => {
    try {
        const [dataUrl, base64Image] = await Promise.all([blobToDataURL(blob),
        fileToBase64(blob as File)]);
        setPreviewImage(dataUrl);
        handleGenerate(base64Image);
    } catch (e) {
        setError('Could not process the image.');
    }
};

const handlePaste = useCallback(async (event: React.ClipboardEvent) => {
    const items = event.clipboardData.items;
    for (const item of items) {
        if (item.type.indexOf('image') !== -1) {
            const blob = item.getAsFile();
            if (blob) await processImageBlob(blob);
            return;
        }
    }
}, []);

const handleFileChange = async (event: React.ChangeEvent<HTMLInputElement>) => {
    const file = event.target.files?.[0];
    if (file) await processImageBlob(file);
};

return (
    <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8">
        <header className="mb-6"><h1 className="text-3xl font-bold text-slate-100 flex
        items-center"><PhotoIcon /><span className="ml-3">AI Screenshot-to-
        Component</span></h1><p className="text-slate-400 mt-1">Paste or upload a
        screenshot of a UI element to generate React/Tailwind code.</p></header>
        <div className="flex-grow grid grid-cols-1 lg:grid-cols-2 gap-6 min-h-0">
            <div onPaste={handlePaste} className="flex flex-col items-center justify-center
            bg-slate-800/50 p-6 rounded-lg border-2 border-dashed border-slate-700
            focus:outline-none focus:border-cyan-500 overflow-y-auto" tabIndex={0}>
                {previewImage ? (<img src={previewImage} alt="Pasted content" className="max-w-
```

```
                    full max-h-full object-contain rounded-md shadow-lg" />) : (<div
                    className="text-center text-slate-400">
                            <h2 className="text-xl font-bold">Paste an image here</h2>
                            <p className="mb-2">(Cmd/Ctrl + V)</p>
                            <p className="text-sm">or</p>
                            <button onClick={() => fileInputRef.current?.click()} className="mt-2 btn-
                            primary px-4 py-2 text-sm">Upload File</button>
                            <input type="file" ref={fileInputRef} onChange={handleFileChange}
                            accept="image/*" className="hidden"/>
                        </div>)}
                </div>
                <div className="flex flex-col h-full">
                    <div className="flex justify-between items-center mb-2">
                        <label className="text-sm font-medium text-slate-400">Generated Code</label>
                        {rawCode && !isLoading && (
                            <div className="flex items-center gap-2">
                                <button onClick={() => navigator.clipboard.writeText(rawCode)} className="px-3
                                py-1 bg-slate-700 text-xs rounded-md hover:bg-slate-600">Copy Code</button>
                                <button onClick={() => downloadFile(rawCode, 'Component.tsx',
                                'text/typescript')} className="flex items-center gap-1 px-3 py-1 bg-slate-700
                                text-xs rounded-md hover:bg-slate-600">
                                    <ArrowDownTrayIcon className="w-4 h-4" /> Download
                                </button>
                            </div>
                        )}
                    </div>
                    <div className="flex-grow bg-slate-900 border border-slate-700 rounded-md
                    overflow-y-auto">
                        {isLoading && (<div className="flex items-center justify-center
                        h-full"><LoadingSpinner /></div>)}
                        {error && <p className="p-4 text-red-400">{error}</p>}
                        {rawCode && !isLoading && <MarkdownRenderer
                        content={`\`\`\`tsx\n${rawCode}\n\`\`\`} />}
                        {!isLoading && !rawCode && !error && (<div className="text-slate-500 h-full flex
                        items-center justify-center">Generated component code will appear here.</div>)}
                    </div>
                </div>
            </div>
        </div>
    );
};

// ===== SnippetVault_2.tsx =====

import React, { useState, useEffect, useMemo } from 'react';
import { LockClosedIcon, SparklesIcon, TrashIcon, ClipboardDocumentIcon,
ArrowDownTrayIcon } from '../icons.tsx';
import { useLocalStorage } from '../../hooks/useLocalStorage.ts';
import { enhanceSnippetStream } from '../../services/geminiService.ts';
import { LoadingSpinner } from '../shared/index.tsx';
import { downloadFile } from '../../services/fileUtils.ts';

interface Snippet {
    id: number; name: string; code: string; language: string; tags: string[];
}

const langToExt: Record<string, string> = {
    javascript: 'js',
    typescript: 'ts',
    python: 'py',
    css: 'css',
    html: 'html',
```

```
    json: 'json',
    markdown: 'md',
    plaintext: 'txt',
};

export const SnippetVault: React.FC = () => {
    const [snippets, setSnippets] = useLocalStorage<Snippet[]>('devcore_snippets',
    [{ id: 1, name: 'React Hook Boilerplate', language: 'javascript', code: `import
    { useState } from 'react';\n\nconst useCustomHook = () => {\n  const [value,
    setValue] = useState(null);\n  return { value, setValue };\n};`, tags: ['react',
    'hook'] }]);
    const [activeSnippet, setActiveSnippet] = useState<Snippet | null>(null);
    const [isEnhancing, setIsEnhancing] = useState(false);
    const [searchTerm, setSearchTerm] = useState('');
    const [isEditingName, setIsEditingName] = useState(false);

    const filteredSnippets = useMemo(() => {
        if (!searchTerm) return snippets;
        const lowerSearch = searchTerm.toLowerCase();
        return snippets.filter((s: Snippet) =>
            s.name.toLowerCase().includes(lowerSearch) ||
            s.code.toLowerCase().includes(lowerSearch) ||
            (s.tags && s.tags.some(t => t.toLowerCase().includes(lowerSearch)))
        );
    }, [snippets, searchTerm]);

    useEffect(() => {
        if (!activeSnippet && filteredSnippets.length > 0)
        setActiveSnippet(filteredSnippets[0]);
        if (activeSnippet) setActiveSnippet(snippets.find((s: Snippet) => s.id ===
        activeSnippet.id) || null);
    }, [snippets, activeSnippet, filteredSnippets]);

    const updateSnippet = (snippet: Snippet) => {
        setSnippets(snippets.map((s: Snippet) => s.id === snippet.id ? snippet : s));
        setActiveSnippet(snippet);
    };

    const handleEnhance = async () => {
        if (!activeSnippet) return;
        setIsEnhancing(true);
        try {
            const stream = enhanceSnippetStream(activeSnippet.code);
            let fullResponse = '';
            for await (const chunk of stream) {
                fullResponse += chunk;
                updateSnippet({ ...activeSnippet, code: fullResponse.replace(/^```(?:\w+\n)?/,
                '').replace(/```$/, '') });
            }
        } finally { setIsEnhancing(false); }
    };

    const handleAddNew = () => {
        const newSnippet: Snippet = { id: Date.now(), name: 'New Snippet', language:
        'plaintext', code: '', tags: [] };
        setSnippets([...snippets, newSnippet]);
        setActiveSnippet(newSnippet);
    };

    const handleDelete = (id: number) => {
        setSnippets(snippets.filter((s: Snippet) => s.id !== id));
        if(activeSnippet?.id === id) setActiveSnippet(filteredSnippets.length > 1 ?
```

```
    filteredSnippets[0] : null);
};

const handleDownload = () => {
    if(!activeSnippet) return;
    const extension = langToExt[activeSnippet.language] || 'txt';
    const filename = `${activeSnippet.name.replace(/\s/g, '_')}.${extension}`;
    downloadFile(activeSnippet.code, filename);
}

const handleNameChange = (e: React.ChangeEvent<HTMLInputElement>) => {
    if (activeSnippet) updateSnippet({...activeSnippet, name: e.target.value});
};

const handleTagsChange = (e: React.KeyboardEvent<HTMLInputElement>) => {
    if (e.key === 'Enter' && activeSnippet) {
        const newTag = e.currentTarget.value.trim();
        if (newTag && !activeSnippet.tags.includes(newTag)) {
            updateSnippet({...activeSnippet, tags: [...(activeSnippet.tags ?? []),
            newTag]});
        }
        e.currentTarget.value = '';
    }
};

return (
    <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
        <header className="mb-6"><h1 className="text-3xl font-bold flex items-
        center"><LockClosedIcon /><span className="ml-3">Snippet Vault</span></h1><p
        className="text-text-secondary mt-1">Store, search, tag, and enhance your
        reusable code snippets with AI.</p></header>
        <div className="flex-grow flex gap-6 min-h-0">
            <aside className="w-1/3 bg-surface border border-border p-4 rounded-lg flex
            flex-col">
                <input type="text" placeholder="Search snippets..." value={searchTerm}
                onChange={e => setSearchTerm(e.target.value)} className="w-full px-3 py-1.5 mb-3
                rounded-md bg-background border border-border text-sm"/>
                <ul className="space-y-2 flex-grow overflow-y-auto
                pr-2">{filteredSnippets.map((s: Snippet) => (<li key={s.id} className="group
                flex items-center justify-between"><button onClick={() => setActiveSnippet(s)}
                className={`w-full text-left px-3 py-2 rounded-md ${activeSnippet?.id === s.id ?
                'bg-primary/10 text-primary' : 'hover:bg-gray-100'}`}>{s.name}</button><div
                className="flex opacity-0 group-hover:opacity-100 transition-opacity"><button
                onClick={() => navigator.clipboard.writeText(s.code)} className="ml-2 p-1 text-
                text-secondary hover:text-primary" title="Copy"><ClipboardDocumentIcon
                /></button><button onClick={() => handleDelete(s.id)} className="ml-2 p-1 text-
                text-secondary hover:text-red-500"
                title="Delete"><TrashIcon/></button></div></li>))}</ul>
                <div className="mt-4 pt-4 border-t border-border"><button onClick={handleAddNew}
                className="btn-primary w-full text-sm py-2">Add New Snippet</button></div>
            </aside>
            <main className="w-2/3 flex flex-col">
                {activeSnippet ? (<>
                    <div className="flex justify-between items-center mb-2">
                        {isEditingName ? <input type="text" value={activeSnippet.name}
                        onChange={handleNameChange} onBlur={() => setIsEditingName(false)} autoFocus
                        className="text-lg font-bold bg-gray-100 rounded px-2"/> : <h3 onDoubleClick={()
                        => setIsEditingName(true)} className="text-lg font-bold cursor-
                        pointer">{activeSnippet.name}</h3>}
                        <div className="flex gap-2">
                            <button onClick={handleEnhance} disabled={isEnhancing} className="flex items-
                            center gap-2 px-3 py-1 bg-purple-500 text-white font-bold text-xs rounded-md
```

```
                                disabled:bg-gray-400"><SparklesIcon /> AI Enhance</button>
                                <button onClick={() => navigator.clipboard.writeText(activeSnippet.code)}
                                className="px-3 py-1 bg-gray-100 text-xs rounded-md">Copy</button>
                                <button onClick={handleDownload} className="flex items-center gap-1 px-3 py-1
                                bg-gray-100 text-xs rounded-md"><ArrowDownTrayIcon className="w-4 h-4"/>
                                Download</button>
                            </div>
                        </div>
                        <textarea value={activeSnippet.code} onChange={e =>
                        updateSnippet({...activeSnippet, code: e.target.value})} className="flex-grow
                        p-4 bg-surface border border-border rounded-md resize-none font-mono text-sm
                        focus:ring-2 focus:ring-primary focus:outline-none"/>
                        <div className="mt-2 text-xs text-text-secondary">
                           <div className="flex items-center gap-2 flex-wrap">
                             <span className="font-bold">Tags:</span> {(activeSnippet.tags ?? []).map(t =>
                             <span key={t} className="bg-gray-200 px-2 py-0.5 rounded-full">{t}</span>)}
                             <input type="text" placeholder="+ Add tag" onKeyDown={handleTagsChange}
                             className="bg-transparent border-b border-border focus:outline-none
                             focus:border-primary w-24 text-xs px-1"/>
                           </div>
                        </div>
                    </>) : (<div className="flex-grow flex items-center justify-center bg-background
                    border border-border rounded-lg text-text-secondary">Select a snippet or create
                    a new one.</div>)}
                </main>
            </div>
        </div>
    );
};

// ===== SvgPathEditor_2.tsx =====

import React, { useState, useRef } from 'react';
import { CodeBracketSquareIcon, ArrowDownTrayIcon } from '../icons.tsx';
import { downloadFile } from '../../services/fileUtils.ts';

const initialPath = "M 20 80 Q 100 20 180 80 T 340 80";

const parsePath = (d: string) => {
    const commands = d.match(/[a-df-z][^a-df-z]*/ig) || [];
    return commands.map((cmdStr, i) => {
        const command = cmdStr[0];
        const args = cmdStr.slice(1).trim().split(/[\s,]+/).map(parseFloat).filter(n =>
        !isNaN(n));
        const points = [];
        for (let j = 0; j < args.length; j += 2) {
            points.push({ x: args[j], y: args[j + 1] });
        }
        return { id: i, command, points };
    });
};
const buildPath = (parsed: any[]) => {
    return parsed.map(cmd => `${cmd.command} ${cmd.points.map((p:any) => `${p.x}
    ${p.y}`).join(' ')}`).join(' ');
};

export const SvgPathEditor: React.FC = () => {
    const [pathData, setPathData] = useState(initialPath);
    const svgRef = useRef<SVGSVGElement>(null);
    const [draggingPoint, setDraggingPoint] = useState<any>(null);
    const parsedPath = parsePath(pathData);
    const handleMouseDown = (e: React.MouseEvent, cmdIndex: number, pointIndex:
```

```
    number) => {
        e.stopPropagation();
        setDraggingPoint({ cmdIndex, pointIndex });
    };

    const handleMouseMove = (e: React.MouseEvent) => {
        if (!draggingPoint || !svgRef.current) return;
        const pt = new DOMPoint(e.clientX, e.clientY);
        const svgPoint = pt.matrixTransform(svgRef.current.getScreenCTM()?.inverse());

        const newParsedPath = parsedPath.map((cmd, cIdx) => {
            if (cIdx === draggingPoint.cmdIndex) {
                const newPoints = cmd.points.map((p, pIdx) => {
                    if (pIdx === draggingPoint.pointIndex) {
                        return { x: Math.round(svgPoint.x), y: Math.round(svgPoint.y) };
                    }
                    return p;
                });
                return { ...cmd, points: newPoints };
            }
            return cmd;
        });
        setPathData(buildPath(newParsedPath));
    };

    const handleMouseUp = () => setDraggingPoint(null);

    const handleDownload = () => {
        const svgContent = `<svg viewBox="0 0 400 160"
        xmlns="http://www.w3.org/2000/svg">
  <path d="${pathData}" stroke="black" fill="transparent" stroke-width="2"/>
</svg>`;
        downloadFile(svgContent, 'path.svg', 'image/svg+xml');
    };

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
            <header className="mb-6"><h1 className="text-3xl font-bold flex items-
            center"><CodeBracketSquareIcon /><span className="ml-3">SVG Path
            Editor</span></h1><p className="text-text-secondary mt-1">Visually create and
            manipulate SVG path data by dragging points.</p></header>
            <div className="flex-grow grid grid-cols-1 lg:grid-cols-2 gap-6 h-full overflow-
            hidden">
                <div className="flex flex-col h-full overflow-y-auto">
                    <div className="flex justify-between items-center mb-2">
                        <label htmlFor="path-input" className="text-sm font-medium text-text-
                        secondary">Path Data (d attribute)</label>
                         <button onClick={handleDownload} className="flex items-center gap-1 px-3 py-1
                         bg-gray-100 text-xs rounded-md hover:bg-gray-200">
                            <ArrowDownTrayIcon className="w-4 h-4"/> Download SVG
                        </button>
                    </div>
                    <textarea id="path-input" value={pathData} onChange={(e) =>
                    setPathData(e.target.value)} className="h-24 p-4 bg-surface border border-border
                    rounded-md resize-y font-mono text-sm text-primary" />
                     <div className="flex-grow mt-4 p-4 bg-surface border-2 border-dashed border-
                     border rounded-md overflow-hidden flex items-center justify-center
                     min-h-[200px]">
                        <svg ref={svgRef} viewBox="0 0 400 160" className="w-full h-full cursor-
                        crosshair" onMouseMove={handleMouseMove} onMouseUp={handleMouseUp}
                        onMouseLeave={handleMouseUp}>
                            <rect width="400" height="160" fill="var(--color-background)" />
```

```
                            <path d={pathData} stroke="var(--color-primary)" fill="transparent"
                            strokeWidth="2" />
                            {parsedPath.flatMap((cmd, cmdIndex) =>
                                cmd.points.map((p, pointIndex) => (
                                    <circle
                                        key={`${cmd.id}-${pointIndex}`}
                                        cx={p.x}
                                        cy={p.y}
                                        r="5"
                                        fill={cmd.command.toLowerCase() === 'c' || cmd.command.toLowerCase() === 'q' ||
                                        cmd.command.toLowerCase() === 's' || cmd.command.toLowerCase() === 't' ?
                                        '#fde047' : '#f87171'}
                                        stroke="var(--color-surface)"
                                        strokeWidth="2"
                                        className="cursor-move hover:stroke-white"
                                        onMouseDown={(e) => handleMouseDown(e, cmdIndex, pointIndex)}
                                    />
                                ))
                            )}
                        </svg>
                    </div>
                </div>
                <div className="flex flex-col h-full">
                    <label className="text-sm font-medium text-text-secondary mb-2">Parsed
                    Commands</label>
                    <div className="flex-grow p-2 bg-background border border-border rounded-md
                    overflow-y-auto font-mono text-xs space-y-2">
                        {parsedPath.map(cmd => (
                            <div key={cmd.id} className="p-2 bg-surface rounded">
                                <span className="font-bold text-amber-600">{cmd.command}</span>
                                <span className="text-text-secondary"> {cmd.points.map(p =>
                                `(${p.x},${p.y})`).join(' ')}</span>
                            </div>
                        ))}
                    </div>
                </div>
            </div>
        </div>
    );
};

// ===== ThemeDesigner_2.tsx =====

import React, { useState, useCallback, useEffect } from 'react';
import { SparklesIcon, ArrowDownTrayIcon } from '../icons.tsx';
import { generateThemeFromDescription } from '../../services/geminiService.ts';
import type { ColorTheme } from '../../types.ts';
import { LoadingSpinner } from '../shared/index.tsx';
import { HexColorPicker } from 'react-colorful';
import { downloadFile } from '../../services/fileUtils.ts';

const ColorInput: React.FC<{ label: string; color: string; onChange: (color:
string) => void; }> = ({ label, color, onChange }) => {
    const [isOpen, setIsOpen] = useState(false);
    return (
        <div className="relative">
            <div onClick={() => setIsOpen(!isOpen)} className="flex items-center justify-
            between p-3 bg-background rounded-md cursor-pointer border-2 border-border
            hover:border-gray-300">
                <div className="flex items-center gap-3">
                    <div className="w-6 h-6 rounded border border-border" style={{ backgroundColor:
                    color }} />
```

```
                    <span className="text-sm font-medium text-text-primary">{label}</span>
                </div>
                <span className="font-mono text-sm text-text-secondary">{color}</span>
            </div>
            {isOpen && (
                <div className="absolute z-10 top-full mt-2 right-0">
                    <div className="fixed inset-0" onClick={() => setIsOpen(false)} />
                    <HexColorPicker color={color} onChange={onChange} />
                </div>
            )}
        </div>
    );
};

const randomPrompts = [
    'A serene forest at dawn',
    'A retro 8-bit video game',
    'A cyberpunk cityscape at night',
    'A warm, cozy coffee shop',
    'An arctic expedition with icy blues',
    'A vibrant coral reef',
];

export const ThemeDesigner: React.FC<{ initialPrompt?: string }> = ({
initialPrompt }) => {
    const [theme, setTheme] = useState<ColorTheme>({
        primary: '#0047AB', background: '#F5F7FA', surface: '#FFFFFF', textPrimary:
        '#111827', textSecondary: '#6B7280',
    });
    const [prompt, setPrompt] = useState(initialPrompt || 'A professional and clean
    corporate blue theme.');
    const [isLoading, setIsLoading] = useState(false);
    const [error, setError] = useState('');

    const handleGenerate = useCallback(async (p: string) => {
        if (!p.trim()) { setError('Please enter a description.'); return; }
        setIsLoading(true); setError('');
        try {
            const newTheme = await generateThemeFromDescription(p);
            setTheme(newTheme);
        } catch (err) {
            setError(err instanceof Error ? err.message : "An unknown error occurred.");
        } finally {
            setIsLoading(false);
        }
    }, []);

    const handleRandom = () => {
        const randomPrompt = randomPrompts[Math.floor(Math.random() *
        randomPrompts.length)];
        setPrompt(randomPrompt);
        handleGenerate(randomPrompt);
    };

    const handleColorChange = (key: keyof ColorTheme, color: string) => {
        setTheme(prev => ({...prev, [key]: color}));
    };

    const getExportContent = (type: 'css' | 'tailwind') => {
        if (type === 'css') {
            return `:root {\n` +
                   `  --primary: ${theme.primary};\n` +
```

```
                    `  --background: ${theme.background};\n` +
                    `  --surface: ${theme.surface};\n` +
                    `  --text-primary: ${theme.textPrimary};\n` +
                    `  --text-secondary: ${theme.textSecondary};\n` +
                    `}`;
        }
        return `// tailwind.config.js
module.exports = {
  theme: {
    extend: {
      colors: {
        primary: '${theme.primary}',
        background: '${theme.background}',
        surface: '${theme.surface}',
        'text-primary': '${theme.textPrimary}',
        'text-secondary': '${theme.textSecondary}',
      },
    },
  },
};`;
    };

    useEffect(() => {
        if (initialPrompt) handleGenerate(initialPrompt);
    }, [initialPrompt, handleGenerate]);

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
            <header className="mb-6">
                <h1 className="text-3xl font-bold flex items-center"><SparklesIcon /><span
                className="ml-3">AI Theme Designer</span></h1>
                <p className="text-text-secondary mt-1">Describe, generate, fine-tune, and
                export a color theme for your application.</p>
            </header>
            <div className="flex-grow grid grid-cols-1 lg:grid-cols-3 gap-6 min-h-0">
                <div className="lg:col-span-1 flex flex-col gap-4 bg-surface border border-
                border p-6 rounded-lg overflow-y-auto">
                    <h3 className="text-xl font-bold">Describe your theme</h3>
                    <textarea value={prompt} onChange={e => setPrompt(e.target.value)}
                    className="p-2 bg-background border border-border rounded-md resize-none text-sm
                    h-24" placeholder="e.g., A light, airy theme for a blog" />
                     <div className="flex gap-2">
                         <button onClick={() => handleGenerate(prompt)} disabled={isLoading}
                         className="btn-primary w-full flex items-center justify-center gap-2 px-4 py-2">
                            {isLoading ? <LoadingSpinner /> : 'Generate Theme'}
                        </button>
                         <button onClick={handleRandom} disabled={isLoading} className="px-4 py-2 bg-
                         gray-100 border border-border rounded-md hover:bg-gray-200" title="Random
                         Theme">■</button>
                     </div>
                    {error && <p className="text-red-500 text-xs text-center">{error}</p>}

                    <div className="mt-4 border-t border-border pt-4 space-y-2 flex-grow overflow-y-
                    auto pr-2">
                         <h3 className="text-lg font-bold">Fine-Tune Palette</h3>
                         <ColorInput label="Primary" color={theme.primary} onChange={c =>
                         handleColorChange('primary', c)} />
                         <ColorInput label="Background" color={theme.background} onChange={c =>
                         handleColorChange('background', c)} />
                         <ColorInput label="Surface" color={theme.surface} onChange={c =>
                         handleColorChange('surface', c)} />
                         <ColorInput label="Text Primary" color={theme.textPrimary} onChange={c =>
```

```
                        handleColorChange('textPrimary', c)} />
                        <ColorInput label="Text Secondary" color={theme.textSecondary} onChange={c =>
                        handleColorChange('textSecondary', c)} />
                    </div>
                    <div className="flex-shrink-0 pt-4 border-t border-border flex flex-col gap-2">
                        <button onClick={() => downloadFile(getExportContent('css'), 'theme.css',
                        'text/css')} className="flex-1 text-sm py-2 bg-gray-100 border border-border
                        rounded-md flex items-center justify-center gap-2 hover:bg-gray-200">
                            <ArrowDownTrayIcon className="w-4 h-4"/> Download CSS
                        </button>
                        <button onClick={() => downloadFile(getExportContent('tailwind'),
                        'tailwind.config.js', 'application/javascript')} className="flex-1 text-sm py-2
                        bg-gray-100 border border-border rounded-md flex items-center justify-center
                        gap-2 hover:bg-gray-200">
                             <ArrowDownTrayIcon className="w-4 h-4"/> Download Tailwind Config
                        </button>
                    </div>
                </div>
                <div className="lg:col-span-2 rounded-lg p-8 overflow-y-auto border border-
                border" style={{ backgroundColor: theme.background, color: theme.textPrimary }}>
                     <h3 className="text-2xl font-bold mb-6" style={{ color: theme.textPrimary
                     }}>Live Preview</h3>
                     <div className="p-6 rounded-lg grid grid-cols-1 md:grid-cols-2 gap-6" style={{
                     backgroundColor: theme.surface }}>
                        <div className="space-y-4">
                            <h4 className="text-lg font-bold">Sample Card</h4>
                            <p className="text-sm" style={{color: theme.textSecondary}}>This is a sample
                            card to demonstrate the theme colors. It contains a primary button and some
                            secondary text.</p>
                            <button className="px-4 py-2 rounded-md font-bold transition-colors" style={{
                            backgroundColor: theme.primary, color: theme.textSecondary.includes('#F') ?
                            '#000' : '#FFF' }}>Primary Button</button>
                        </div>
                         <div className="space-y-4">
                            <input type="text" placeholder="Text input" className="w-full px-3 py-2 rounded-
                            md border" style={{backgroundColor: theme.background, borderColor:
                            theme.primary, color: theme.textPrimary}} />
                            <div className="p-3 border rounded" style={{borderColor: theme.primary, color:
                            theme.textSecondary}}>
                                <p>A bordered container.</p>
                            </div>
                         </div>
                     </div>
                     <div className="mt-6">
                        <h4 className="text-lg font-bold mb-2">Typography</h4>
                        <h1>Heading 1</h1>
                        <h2>Heading 2</h2>
                        <p style={{color: theme.textPrimary}}>This is a paragraph of primary text. Lorem
                        ipsum dolor sit amet, consectetur adipiscing elit.</p>
                        <p style={{color: theme.textSecondary}}>This is secondary text, for descriptions
                        or less important information.</p>
                     </div>
                </div>
            </div>
        </div>
    );
};

// ===== TypographyLab.tsx =====

import React, { useState, useEffect } from 'react';
import { TypographyLabIcon } from '../icons.tsx';
```

```
const popularFonts = [
    'Roboto', 'Open Sans', 'Lato', 'Montserrat', 'Oswald', 'Source Sans Pro',
    'Raleway', 'Poppins', 'Nunito', 'Merriweather',
    'Playfair Display', 'Lora', 'Noto Sans', 'Ubuntu', 'PT Sans', 'Slabo 27px'
];

export const TypographyLab: React.FC = () => {
    const [headingFont, setHeadingFont] = useState('Oswald');
    const [bodyFont, setBodyFont] = useState('Roboto');

    useEffect(() => {
        const fontsToLoad = [headingFont, bodyFont].filter(Boolean).join('|');
        if (fontsToLoad) {
            const linkId = 'font-pairing-stylesheet';
            let link = document.getElementById(linkId) as HTMLLinkElement;
            if (!link) {
                link = document.createElement('link');
                link.id = linkId;
                link.rel = 'stylesheet';
                document.head.appendChild(link);
            }
            link.href = `https://fonts.googleapis.com/css?family=${fontsToLoad.replace(/ /g,
            '+')}:400,700&display=swap`;
        }
    }, [headingFont, bodyFont]);

    const FontSelector: React.FC<{ label: string, value: string, onChange: (font:
    string) => void }> = ({ label, value, onChange }) => (
        <div>
            <label className="block text-sm font-medium text-text-secondary">{label}</label>
            <select value={value} onChange={e => onChange(e.target.value)} className="w-full
            mt-1 px-3 py-2 rounded-md bg-surface border border-border">
                {popularFonts.map(font => <option key={font} value={font}>{font}</option>)}
            </select>
        </div>
    );

    const headingImport = `@import
    url('https://fonts.googleapis.com/css?family=${headingFont.replace(/ /g,
    '+')}:700&display=swap');`;
    const bodyImport = `@import
    url('https://fonts.googleapis.com/css?family=${bodyFont.replace(/ /g,
    '+')}:400&display=swap');`;
    const headingRule = `font-family: '${headingFont}', sans-serif;`;
    const bodyRule = `font-family: '${bodyFont}', sans-serif;`;

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
            <header className="mb-6">
                <h1 className="text-3xl font-bold flex items-center">
                    <TypographyLabIcon />
                    <span className="ml-3">Typography Lab</span>
                </h1>
                <p className="text-text-secondary mt-1">Preview font pairings and get the
                necessary CSS rules.</p>
            </header>
            <div className="flex-grow grid grid-cols-1 lg:grid-cols-3 gap-6 min-h-0">
                <div className="lg:col-span-1 flex flex-col gap-4 bg-surface border border-
                border p-6 rounded-lg">
                    <h3 className="text-xl font-bold">Controls</h3>
                    <FontSelector label="Heading Font" value={headingFont} onChange={setHeadingFont}
                    />
```

```
                    <FontSelector label="Body Font" value={bodyFont} onChange={setBodyFont} />
                    <div className="space-y-2 mt-4 pt-4 border-t border-border">
                        <label className="block text-sm font-medium text-text-secondary">CSS
                        Rules</label>
                        <div className="relative"><pre className="bg-background p-2 rounded-md text-
                        primary text-xs overflow-x-auto">{headingImport}</pre><button onClick={() =>
                        navigator.clipboard.writeText(headingImport)} className="absolute top-1 right-1
                        px-2 py-0.5 bg-gray-100 hover:bg-gray-200 rounded-md text-
                        xs">Copy</button></div>
                        <div className="relative"><pre className="bg-background p-2 rounded-md text-
                        primary text-xs overflow-x-auto">{headingRule}</pre><button onClick={() =>
                        navigator.clipboard.writeText(headingRule)} className="absolute top-1 right-1
                        px-2 py-0.5 bg-gray-100 hover:bg-gray-200 rounded-md text-
                        xs">Copy</button></div>
                        <div className="relative"><pre className="bg-background p-2 rounded-md text-
                        primary text-xs overflow-x-auto">{bodyImport}</pre><button onClick={() =>
                        navigator.clipboard.writeText(bodyImport)} className="absolute top-1 right-1
                        px-2 py-0.5 bg-gray-100 hover:bg-gray-200 rounded-md text-
                        xs">Copy</button></div>
                        <div className="relative"><pre className="bg-background p-2 rounded-md text-
                        primary text-xs overflow-x-auto">{bodyRule}</pre><button onClick={() =>
                        navigator.clipboard.writeText(bodyRule)} className="absolute top-1 right-1 px-2
                        py-0.5 bg-gray-100 hover:bg-gray-200 rounded-md text-xs">Copy</button></div>
                    </div>
                </div>
                <div className="lg:col-span-2 bg-background border border-border rounded-lg p-8
                overflow-y-auto">
                    <h2 className="text-4xl font-bold mb-4" style={{ fontFamily: `'${headingFont}',
                    sans-serif` }}>
                        The Quick Brown Fox Jumps Over the Lazy Dog
                    </h2>
                    <p className="text-lg" style={{ fontFamily: `'${bodyFont}', sans-serif` }}>
                        Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed non risus.
                        Suspendisse lectus tortor, dignissim sit amet, adipiscing nec, ultricies sed,
                        dolor. Cras elementum ultrices diam. Maecenas ligula massa, varius a, semper
                        congue, euismod non, mi. Proin porttitor, orci nec nonummy molestie, enim est
                        eleifend mi, non fermentum diam nisl sit amet erat.
                    </p>
                </div>
            </div>
        </div>
    );
};

// ===== VisualGitTree_2.tsx =====

import React, { useState, useCallback, useEffect, useMemo } from 'react';
import { GitBranchIcon, ArrowDownTrayIcon } from '../icons.tsx';
import { generateChangelogFromLogStream } from
'../../services/geminiService.ts';
import { LoadingSpinner, MarkdownRenderer } from '../shared/index.tsx';
import { downloadFile } from '../../services/fileUtils.ts';

const exampleLog = `* commit 3a4b5c6d7e8f9g0h1i2j3k4l5m6n7o8p9q0r (HEAD -> main,
origin/main)
|\\  Merge: 1a2b3c4 2d3e4f5
| | Author: Dev One <dev.one@example.com>
| | Date:   Mon Jul 15 11:30:00 2024 -0400
| |
| |     feat: Implement collapsible sidebar navigation
| |
* | commit 2d3e4f5g6h7i8j9k0l1m2n3o4p5q6r7s8t9u (feature/new-sidebar)
```

```
| | Author: Dev Two <dev.two@example.com>
| | Date:   Mon Jul 15 10:00:00 2024 -0400
| |
| |     feat: Add icons to sidebar items
| |
* | commit 1a2b3c4d5e6f7g8h9i0j1k2l3m4n5o6p7q8r
|/  Author: Dev One <dev.one@example.com>
|   Date:   Fri Jul 12 16:45:00 2024 -0400
|
|       fix: Correct user authentication bug`;

const CommitGraph = ({ logInput }: { logInput: string }) => {
    const commits = useMemo(() => {
        const lines = logInput.split('\n');
        const parsedCommits: any[] = [];
        let currentCommit: any = null;

        lines.forEach(line => {
            const commitMatch = line.match(/^.?[\\|/ ]*\* commit (\w+)(.*)/);
            if (commitMatch) {
                if (currentCommit) parsedCommits.push(currentCommit);
                currentCommit = {
                    hash: commitMatch[1],
                    shortHash: commitMatch[1].substring(0, 7),
                    refs: commitMatch[2].trim(),
                    message: '',
                    author: '',
                };
            } else if (currentCommit) {
                 if (line.includes('Author:')) currentCommit.author =
                 line.split('Author:')[1].trim();
                 else if (line.trim().length > 0 && !line.match(/^[\\|/ ]*[\\|/ ]/)) {
                     currentCommit.message += line.trim() + ' ';
                 }
            }
        });
        if (currentCommit) parsedCommits.push(currentCommit);

        return parsedCommits.map((c, i) => ({ ...c, x: 50, y: 50 + i * 60 }));
    }, [logInput]);

    return (
         <svg width="100%" height={50 + commits.length * 60} className="min-h-[200px]">
            {commits.map((commit, i) => {
                const parent = commits[i + 1];
                return (
                    <g key={commit.hash}>
                        {parent && <line x1={commit.x} y1={commit.y} x2={parent.x} y2={parent.y}
                        stroke="var(--color-border)" strokeWidth="2" />}
                        <g className="group cursor-pointer">
                            <circle cx={commit.x} cy={commit.y} r="8" fill="var(--color-primary)"
                            stroke="var(--color-surface)" strokeWidth="3" />
                            <foreignObject x={commit.x + 20} y={commit.y - 25} width="350" height="50">
                                <div className="text-sm p-1">
                                    <p className="font-bold truncate text-text-primary">{commit.message}</p>
                                    <p className="text-xs text-text-secondary font-mono">{commit.shortHash} <span
                                    className="text-amber-600">{commit.refs}</span></p>
                                </div>
                            </foreignObject>
                            <title>{`Commit: ${commit.hash}\nAuthor:
                            ${commit.author}\n\n${commit.message}`}</title>
                        </g>
```

```
                    </g>
                );
            })}
        </svg>
    );
};

export const VisualGitTree: React.FC<{ logInput?: string }> = ({ logInput:
initialLogInput }) => {
    const [logInput, setLogInput] = useState(initialLogInput || exampleLog);
    const [analysis, setAnalysis] = useState('');
    const [isLoading, setIsLoading] = useState(false);
    const [error, setError] = useState('');

    const handleAnalyze = useCallback(async (logToAnalyze: string) => {
        if (!logToAnalyze.trim()) {
            setError('Please paste git log output.');
            return;
        }
        setIsLoading(true);
        setError('');
        setAnalysis('');
        try {
            const stream = generateChangelogFromLogStream(logToAnalyze);
            let fullResponse = '';
            for await (const chunk of stream) {
                fullResponse += chunk;
                setAnalysis(fullResponse);
            }
        } catch (err) {
            const errorMessage = err instanceof Error ? err.message : 'An unknown error
            occurred.';
            setError(`Failed to analyze log: ${errorMessage}`);
        } finally {
            setIsLoading(false);
        }
    }, []);

    useEffect(() => {
        if (initialLogInput) {
            setLogInput(initialLogInput);
            handleAnalyze(initialLogInput);
        }
    }, [initialLogInput, handleAnalyze]);

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
            <header className="mb-6">
                <h1 className="text-3xl font-bold flex items-center">
                    <GitBranchIcon />
                    <span className="ml-3">Visual Git Tree</span>
                </h1>
                <p className="text-text-secondary mt-1">Paste your `git log --graph` output to
                visualize the history and get an AI summary.</p>
            </header>
            <div className="flex-grow grid grid-cols-1 lg:grid-cols-2 gap-6 h-full overflow-
            hidden">
                <div className="flex flex-col h-full">
                    <label htmlFor="log-input" className="text-sm font-medium text-text-secondary
                    mb-2">Git Log Output</label>
                    <textarea
                        id="log-input"
```

```
                        value={logInput}
                        onChange={(e) => setLogInput(e.target.value)}
                        placeholder="Paste your git log output here..."
                        className="flex-grow p-4 bg-surface border border-border rounded-md resize-none
                        font-mono text-sm"
                    />
                    <button
                        onClick={() => handleAnalyze(logInput)}
                        disabled={isLoading}
                        className="btn-primary mt-4 w-full flex items-center justify-center px-6 py-3"
                    >
                        {isLoading ? <LoadingSpinner /> : 'Analyze & Summarize'}
                    </button>
                </div>
                <div className="flex flex-col h-full gap-4">
                    <div className="flex flex-col h-1/2">
                        <label className="text-sm font-medium text-text-secondary mb-2">Commit
                        Graph</label>
                        <div className="flex-grow p-2 bg-surface border border-border rounded-md
                        overflow-auto">
                            <CommitGraph logInput={logInput} />
                        </div>
                    </div>
                     <div className="flex flex-col h-1/2">
                        <div className="flex justify-between items-center mb-2">
                            <label className="text-sm font-medium text-text-secondary">AI Summary</label>
                            {analysis && !isLoading && (
                                <button onClick={() => downloadFile(analysis, 'summary.md', 'text/markdown')}
                                className="flex items-center gap-1 px-3 py-1 bg-gray-100 text-xs rounded-md
                                hover:bg-gray-200">
                                    <ArrowDownTrayIcon className="w-4 h-4"/> Download Summary
                                </button>
                            )}
                        </div>
                        <div className="flex-grow p-4 bg-background border border-border rounded-md
                        overflow-y-auto">
                            {isLoading && <div className="flex items-center justify-center
                            h-full"><LoadingSpinner /></div>}
                            {error && <p className="text-red-500">{error}</p>}
                            {analysis && !isLoading && <MarkdownRenderer content={analysis} />}
                            {!isLoading && !analysis && !error && <div className="text-text-secondary h-full
                            flex items-center justify-center">AI summary will appear here.</div>}
                        </div>
                    </div>
                </div>
            </div>
        </div>
    );
};

// ===== WorkerThreadDebugger_2.tsx =====

import React, { useState, useCallback, useEffect } from 'react';
import { BugAntIcon, ArrowDownTrayIcon } from '../icons.tsx';
import { analyzeConcurrencyStream } from '../../services/geminiService.ts';
import { LoadingSpinner, MarkdownRenderer } from '../shared/index.tsx';
import { downloadFile } from '../../services/fileUtils.ts';

const exampleCode = `// main.js
const worker = new Worker('worker.js');

// This object is sent back and forth.
```

```
// A race condition can occur because both threads
// read the counter, increment it, and send it back.
// The final value depends on which thread's message
// is processed last.
const data = { counter: 0 };

worker.onmessage = function(e) {
  // Main thread reads and updates
  data.counter = e.data.counter;
  console.log('Main received:', data.counter);
  data.counter++;
  worker.postMessage(data);
};

// Start the process
console.log('Main starting with:', data.counter);
data.counter++;
worker.postMessage(data);


// worker.js
// onmessage = function(e) {
//   // Worker reads and updates
//   let receivedCounter = e.data.counter;
//   console.log('Worker received:', receivedCounter);
//   receivedCounter++;
//   postMessage({ counter: receivedCounter });
// }
`;

export const WorkerThreadDebugger: React.FC<{ codeInput?: string }> = ({
codeInput: initialCode }) => {
    const [codeInput, setCodeInput] = useState(initialCode || exampleCode);
    const [analysis, setAnalysis] = useState('');
    const [isLoading, setIsLoading] = useState(false);
    const [error, setError] = useState('');

    const handleAnalyze = useCallback(async (codeToAnalyze: string) => {
        if (!codeToAnalyze.trim()) {
            setError('Please paste some code to analyze.');
            return;
        }
        setIsLoading(true);
        setError('');
        setAnalysis('');
        try {
            const stream = analyzeConcurrencyStream(codeToAnalyze);
            let fullResponse = '';
            for await (const chunk of stream) {
                fullResponse += chunk;
                setAnalysis(fullResponse);
            }
        } catch (err) {
            const errorMessage = err instanceof Error ? err.message : 'An unknown error
            occurred.';
            setError(`Failed to analyze code: ${errorMessage}`);
        } finally {
            setIsLoading(false);
        }
    }, []);

    useEffect(() => {
```

```
    if (initialCode) {
        setCodeInput(initialCode);
        handleAnalyze(initialCode);
    }
}, [initialCode, handleAnalyze]);

return (
    <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
        <header className="mb-6">
            <h1 className="text-3xl font-bold flex items-center">
                <BugAntIcon />
                <span className="ml-3">AI Concurrency Analyzer</span>
            </h1>
            <p className="text-text-secondary mt-1">Analyze JavaScript code for potential
            Web Worker concurrency issues.</p>
        </header>
        <div className="flex-grow flex flex-col gap-4 min-h-0">
            <div className="flex flex-col flex-1 min-h-0">
                <label htmlFor="code-input" className="text-sm font-medium text-text-secondary
                mb-2">JavaScript Code</label>
                <textarea
                    id="code-input"
                    value={codeInput}
                    onChange={(e) => setCodeInput(e.target.value)}
                    placeholder="Paste your worker-related JS code here..."
                    className="flex-grow p-4 bg-surface border border-border rounded-md resize-none
                    font-mono text-sm"
                />
            </div>
             <div className="flex-shrink-0">
                <button
                    onClick={() => handleAnalyze(codeInput)}
                    disabled={isLoading}
                    className="btn-primary w-full max-w-xs mx-auto flex items-center justify-center
                    px-6 py-3"
                >
                    {isLoading ? <LoadingSpinner /> : 'Analyze Code'}
                </button>
            </div>
            <div className="flex flex-col flex-1 min-h-0">
                <div className="flex justify-between items-center mb-2">
                    <label className="text-sm font-medium text-text-secondary">AI Analysis</label>
                    {analysis && !isLoading && (
                         <button onClick={() => downloadFile(analysis, 'analysis.md', 'text/markdown')}
                         className="flex items-center gap-1 px-3 py-1 bg-gray-100 text-xs rounded-md
                         hover:bg-gray-200">
                            <ArrowDownTrayIcon className="w-4 h-4"/> Download
                        </button>
                    )}
                </div>
                <div className="flex-grow p-4 bg-background border border-border rounded-md
                overflow-y-auto">
                    {isLoading && <div className="flex items-center justify-center
                    h-full"><LoadingSpinner /></div>}
                    {error && <p className="text-red-500">{error}</p>}
                    {analysis && !isLoading && <MarkdownRenderer content={analysis} />}
                    {!isLoading && !analysis && !error && <div className="text-text-secondary h-full
                    flex items-center justify-center">Analysis will appear here.</div>}
                </div>
            </div>
        </div>
    </div>
```

```
    );
};

// ===== XbrlConverter.tsx =====

import React, { useState, useCallback } from 'react';
import { convertJsonToXbrlStream } from '../../services/geminiService.ts';
import { XbrlConverterIcon } from '../icons.tsx';
import { LoadingSpinner, MarkdownRenderer } from '../shared/index.tsx';

const exampleJson = `{
  "company": "ExampleCorp",
  "year": 2024,
  "quarter": 2,
  "revenue": {
    "amount": 1500000,
    "currency": "USD"
  },
  "profit": {
    "amount": 250000,
    "currency": "USD"
  }
}`;

export const XbrlConverter: React.FC<{ jsonInput?: string }> = ({ jsonInput:
initialJsonInput }) => {
    const [jsonInput, setJsonInput] = useState<string>(initialJsonInput ||
    exampleJson);
    const [xbrlOutput, setXbrlOutput] = useState<string>('');
    const [isLoading, setIsLoading] = useState<boolean>(false);
    const [error, setError] = useState<string>('');

    const handleConvert = useCallback(async (jsonToConvert: string) => {
        if (!jsonToConvert.trim()) {
            setError('Please enter valid JSON to convert.');
            return;
        }
        setIsLoading(true);
        setError('');
        setXbrlOutput('');
        try {
            const stream = convertJsonToXbrlStream(jsonToConvert);
            let fullResponse = '';
            for await (const chunk of stream) {
                fullResponse += chunk;
                setXbrlOutput(fullResponse);
            }
        } catch (err) {
            const errorMessage = err instanceof Error ? err.message : 'An unknown error
            occurred.';
            setError(`Failed to convert: ${errorMessage}`);
        } finally {
            setIsLoading(false);
        }
    }, []);

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
            <header className="mb-6">
                <h1 className="text-3xl font-bold flex items-center">
                    <XbrlConverterIcon />
                    <span className="ml-3">JSON to XBRL Converter</span>
```

```
                </h1>
                <p className="text-text-secondary mt-1">Convert JSON data into a simplified
                XBRL-like XML format using AI.</p>
            </header>
            <div className="flex-grow flex flex-col gap-4 min-h-0">
                <div className="flex flex-col flex-1 min-h-0">
                    <label htmlFor="json-input" className="text-sm font-medium text-text-secondary
                    mb-2">JSON Input</label>
                    <textarea
                        id="json-input"
                        value={jsonInput}
                        onChange={(e) => setJsonInput(e.target.value)}
                        placeholder="Paste your JSON here..."
                        className="flex-grow p-4 bg-surface border border-border rounded-md resize-none
                        font-mono text-sm"
                    />
                </div>
                 <div className="flex-shrink-0">
                    <button
                        onClick={() => handleConvert(jsonInput)}
                        disabled={isLoading}
                        className="btn-primary w-full max-w-xs mx-auto flex items-center justify-center
                        px-6 py-3"
                    >
                        {isLoading ? <LoadingSpinner /> : 'Convert to XBRL'}
                    </button>
                </div>
                <div className="flex flex-col flex-1 min-h-0">
                    <label className="text-sm font-medium text-text-secondary mb-2">XBRL-like XML
                    Output</label>
                    <div className="relative flex-grow p-1 bg-background border border-border
                    rounded-md overflow-y-auto">
                        {isLoading && !xbrlOutput && <div className="flex items-center justify-center
                        h-full"><LoadingSpinner /></div>}
                        {error && <p className="p-4 text-red-500">{error}</p>}
                        {xbrlOutput && <MarkdownRenderer content={'```xml\n' +
                        xbrlOutput.replace(/```xml\n|```/g, '') + '\n```'} />}
                        {!isLoading && xbrlOutput && <button onClick={() =>
                        navigator.clipboard.writeText(xbrlOutput)} className="absolute top-2 right-2
                        px-2 py-1 bg-gray-100 hover:bg-gray-200 rounded-md text-xs">Copy XML</button>}
                        {!isLoading && !xbrlOutput && !error && <div className="text-text-secondary
                        h-full flex items-center justify-center">Output will appear here.</div>}
                    </div>
                </div>
            </div>
        </div>
    );
};

// ===== index.tsx =====

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

export const LoadingSpinner: React.FC = () => (
    <div className="flex items-center justify-center space-x-1" aria-
    label="Loading">
        <div className="w-2 h-2 rounded-full bg-current animate-pulse" style={{
        animationDelay: '0s' }}></div>
        <div className="w-2 h-2 rounded-full bg-current animate-pulse" style={{
        animationDelay: '0.2s' }}></div>
        <div className="w-2 h-2 rounded-full bg-current animate-pulse" style={{
```

```
        animationDelay: '0.4s' }}></div>
    </div>
);

interface MarkdownRendererProps {
    content: string;
}

export const MarkdownRenderer: React.FC<MarkdownRendererProps> = ({ content })
=> {
    const [sanitizedHtml, setSanitizedHtml] = useState<string | TrustedHTML>('');

    useEffect(() => {
        const parse = async () => {
            if (content) {
                const html = await marked.parse(content);
                setSanitizedHtml(html);
            } else {
                setSanitizedHtml('');
            }
        };
        parse();
    }, [content]);

    return (
        <div
            className="prose prose-sm max-w-none prose-headings:text-text-primary
            prose-p:text-text-primary prose-strong:text-text-primary prose-code:text-primary
            prose-code:before:content-none prose-code:after:content-none prose-pre:bg-
            gray-50 prose-pre:border prose-pre:border-border prose-pre:p-4 prose-pre:m-0"
            dangerouslySetInnerHTML={{ __html: sanitizedHtml }}
        />
    );
};


// ===== ProjectExplorer_1.tsx =====

import React from 'react';
import { FolderIcon } from '../icons.tsx';

export const ProjectExplorer: React.FC = () => {
    return (
        <div className="h-full flex flex-col items-center justify-center text-center
        text-text-secondary p-4">
            <FolderIcon />
            <h2 className="text-lg font-semibold mt-2">GitHub Integration Required</h2>
            <p>Project Explorer is disabled because the application is running without
            GitHub authentication.</p>
        </div>
    );
};


// ===== Connections_1.tsx =====

import React from 'react';
import { GithubIcon } from '../icons.tsx';

export const Connections: React.FC = () => {
    return (
        <div className="h-full p-4 sm:p-6 lg:p-8">
```

```
            <div className="max-w-4xl mx-auto">
                <header className="mb-8">
                    <h1 className="text-4xl font-extrabold tracking-tight">Connections</h1>
                    <p className="mt-2 text-lg text-text-secondary">Manage your GitHub connection to
                    unlock powerful workflows.</p>
                </header>

                <div className="space-y-6">
                     <div className="bg-surface border border-border rounded-lg p-6 flex items-center
                     justify-between">
                        <div className="flex items-center gap-4">
                            <div className="w-10 h-10"><GithubIcon /></div>
                            <div>
                                <h3 className="text-lg font-bold text-text-primary">GitHub</h3>
                                <p className="text-sm text-text-secondary">Not Connected</p>
                            </div>
                        </div>
                    </div>
                    <div className="bg-surface border border-border rounded-lg p-6 text-center">
                        <h3 className="text-lg font-bold text-text-primary mb-2">GitHub Integration
                        Disabled</h3>
                        <p className="text-sm text-text-secondary">
                            This application is running without GitHub authentication. Features like Project
                            Explorer and committing to a repository are unavailable.
                        </p>
                    </div>
                </div>
            </div>
        </div>
    );
};


// ===== ThemeDesigner_6.tsx =====

import React, { useState, useCallback, useEffect } from 'react';
import { SparklesIcon, ArrowDownTrayIcon } from '../icons.tsx';
import { generateThemeFromDescription } from '../../services/geminiService.ts';
import type { ColorTheme } from '../../types.ts';
import { LoadingSpinner } from '../shared/index.tsx';
import { HexColorPicker } from 'react-colorful';
import { downloadFile } from '../../services/fileUtils.ts';

const ColorInput: React.FC<{ label: string; color: string; onChange: (color:
string) => void; }> = ({ label, color, onChange }) => {
    const [isOpen, setIsOpen] = useState(false);
    return (
         <div className="relative">
             <div onClick={() => setIsOpen(!isOpen)} className="flex items-center justify-
             between p-3 bg-background rounded-md cursor-pointer border-2 border-border
             hover:border-gray-300">
                <div className="flex items-center gap-3">
                     <div className="w-6 h-6 rounded border border-border" style={{ backgroundColor:
                     color }} />
                     <span className="text-sm font-medium text-text-primary">{label}</span>
                </div>
                <span className="font-mono text-sm text-text-secondary">{color}</span>
            </div>
            {isOpen && (
                <div className="absolute z-10 top-full mt-2 right-0">
                     <div className="fixed inset-0" onClick={() => setIsOpen(false)} />
                    <HexColorPicker color={color} onChange={onChange} />
```

```
                </div>
            )}
        </div>
    );
};

const randomPrompts = [
    'A serene forest at dawn',
    'A retro 8-bit video game',
    'A cyberpunk cityscape at night',
    'A warm, cozy coffee shop',
    'An arctic expedition with icy blues',
    'A vibrant coral reef',
];

export const ThemeDesigner: React.FC<{ initialPrompt?: string }> = ({
initialPrompt }) => {
    const [theme, setTheme] = useState<ColorTheme>({
        primary: '#0047AB', background: '#F5F7FA', surface: '#FFFFFF', textPrimary:
        '#111827', textSecondary: '#6B7280',
    });
    const [prompt, setPrompt] = useState(initialPrompt || 'A professional and clean
    corporate blue theme.');
    const [isLoading, setIsLoading] = useState(false);
    const [error, setError] = useState('');

    const handleGenerate = useCallback(async (p: string) => {
        if (!p.trim()) { setError('Please enter a description.'); return; }
        setIsLoading(true); setError('');
        try {
            const newTheme = await generateThemeFromDescription(p);
            setTheme(newTheme);
        } catch (err) {
            setError(err instanceof Error ? err.message : "An unknown error occurred.");
        } finally {
            setIsLoading(false);
        }
    }, []);

    const handleRandom = () => {
        const randomPrompt = randomPrompts[Math.floor(Math.random() *
        randomPrompts.length)];
        setPrompt(randomPrompt);
        handleGenerate(randomPrompt);
    };

    const handleColorChange = (key: keyof ColorTheme, color: string) => {
        setTheme(prev => ({...prev, [key]: color}));
    };

    const getExportContent = (type: 'css' | 'tailwind') => {
        if (type === 'css') {
            return `:root {\n` +
                   `  --primary: ${theme.primary};\n` +
                   `  --background: ${theme.background};\n` +
                   `  --surface: ${theme.surface};\n` +
                   `  --text-primary: ${theme.textPrimary};\n` +
                   `  --text-secondary: ${theme.textSecondary};\n` +
                   `}`;
        }
        return `// tailwind.config.js
module.exports = {
```

```
  theme: {
    extend: {
      colors: {
        primary: '${theme.primary}',
        background: '${theme.background}',
        surface: '${theme.surface}',
        'text-primary': '${theme.textPrimary}',
        'text-secondary': '${theme.textSecondary}',
      },
    },
  },
};`;
    };

    useEffect(() => {
        if (initialPrompt) handleGenerate(initialPrompt);
    }, [initialPrompt, handleGenerate]);

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
            <header className="mb-6">
                <h1 className="text-3xl font-bold flex items-center"><SparklesIcon /><span
                className="ml-3">AI Theme Designer</span></h1>
                <p className="text-text-secondary mt-1">Describe, generate, fine-tune, and
                export a color theme for your application.</p>
            </header>
            <div className="flex-grow grid grid-cols-1 md:grid-cols-2 gap-6 min-h-0">
                <div className="md:col-span-1 flex flex-col gap-4 bg-surface border border-
                border p-6 rounded-lg overflow-y-auto">
                    <h3 className="text-xl font-bold">Describe your theme</h3>
                    <textarea value={prompt} onChange={e => setPrompt(e.target.value)}
                    className="p-2 bg-background border border-border rounded-md resize-none text-sm
                    h-24" placeholder="e.g., A light, airy theme for a blog" />
                     <div className="flex gap-2">
                         <button onClick={() => handleGenerate(prompt)} disabled={isLoading}
                         className="btn-primary w-full flex items-center justify-center gap-2 px-4 py-2">
                            {isLoading ? <LoadingSpinner /> : 'Generate Theme'}
                        </button>
                         <button onClick={handleRandom} disabled={isLoading} className="px-4 py-2 bg-
                         gray-100 border border-border rounded-md hover:bg-gray-200" title="Random
                         Theme">■</button>
                     </div>
                    {error && <p className="text-red-500 text-xs text-center">{error}</p>}

                    <div className="mt-4 border-t border-border pt-4 space-y-2 flex-grow overflow-y-
                    auto pr-2">
                         <h3 className="text-lg font-bold">Fine-Tune Palette</h3>
                         <ColorInput label="Primary" color={theme.primary} onChange={c =>
                         handleColorChange('primary', c)} />
                         <ColorInput label="Background" color={theme.background} onChange={c =>
                         handleColorChange('background', c)} />
                         <ColorInput label="Surface" color={theme.surface} onChange={c =>
                         handleColorChange('surface', c)} />
                         <ColorInput label="Text Primary" color={theme.textPrimary} onChange={c =>
                         handleColorChange('textPrimary', c)} />
                         <ColorInput label="Text Secondary" color={theme.textSecondary} onChange={c =>
                         handleColorChange('textSecondary', c)} />
                    </div>
                    <div className="flex-shrink-0 pt-4 border-t border-border flex flex-col gap-2">
                        <button onClick={() => downloadFile(getExportContent('css'), 'theme.css',
                        'text/css')} className="flex-1 text-sm py-2 bg-gray-100 border border-border
                        rounded-md flex items-center justify-center gap-2 hover:bg-gray-200">
```

```
                            <ArrowDownTrayIcon className="w-4 h-4"/> Download CSS
                        </button>
                        <button onClick={() => downloadFile(getExportContent('tailwind'),
                        'tailwind.config.js', 'application/javascript')} className="flex-1 text-sm py-2
                        bg-gray-100 border border-border rounded-md flex items-center justify-center
                        gap-2 hover:bg-gray-200">
                             <ArrowDownTrayIcon className="w-4 h-4"/> Download Tailwind Config
                        </button>
                    </div>
                </div>
                <div className="md:col-span-1 rounded-lg p-8 overflow-y-auto border border-
                border" style={{ backgroundColor: theme.background, color: theme.textPrimary }}>
                     <h3 className="text-2xl font-bold mb-6" style={{ color: theme.textPrimary
                     }}>Live Preview</h3>
                     <div className="p-6 rounded-lg grid grid-cols-1 md:grid-cols-2 gap-6" style={{
                     backgroundColor: theme.surface }}>
                        <div className="space-y-4">
                            <h4 className="text-lg font-bold">Sample Card</h4>
                            <p className="text-sm" style={{color: theme.textSecondary}}>This is a sample
                            card to demonstrate the theme colors. It contains a primary button and some
                            secondary text.</p>
                            <button className="px-4 py-2 rounded-md font-bold transition-colors" style={{
                            backgroundColor: theme.primary, color: theme.textSecondary.includes('#F') ?
                            '#000' : '#FFF' }}>Primary Button</button>
                        </div>
                         <div className="space-y-4">
                            <input type="text" placeholder="Text input" className="w-full px-3 py-2 rounded-
                            md border" style={{backgroundColor: theme.background, borderColor:
                            theme.primary, color: theme.textPrimary}} />
                            <div className="p-3 border rounded" style={{borderColor: theme.primary, color:
                            theme.textSecondary}}>
                                <p>A bordered container.</p>
                            </div>
                         </div>
                     </div>
                     <div className="mt-6">
                        <h4 className="text-lg font-bold mb-2">Typography</h4>
                        <h1>Heading 1</h1>
                        <h2>Heading 2</h2>
                        <p style={{color: theme.textPrimary}}>This is a paragraph of primary text. Lorem
                        ipsum dolor sit amet, consectetur adipiscing elit.</p>
                        <p style={{color: theme.textSecondary}}>This is secondary text, for less
                        important information.</p>
                     </div>
                </div>
            </div>
        </div>
    );
};

// ===== LogicFlowBuilder_6.tsx =====

import React, { useState, useRef, useMemo, useCallback } from 'react';
import { ALL_FEATURES } from './index.ts';
import { FEATURE_TAXONOMY, generatePipelineCode } from
'../../services/index.ts';
import type { Feature } from '../../types.ts';
import { MapIcon, SparklesIcon, XMarkIcon } from '../icons.tsx';
import { LoadingSpinner, MarkdownRenderer } from '../shared/index.tsx';

interface Node {
    id: number;
```

```
    featureId: string;
    x: number;
    y: number;
}

interface Link {
    from: number;
    to: number;
}

const featuresMap = new Map(ALL_FEATURES.map(f => [f.id, f]));
const taxonomyMap = new Map(FEATURE_TAXONOMY.map(f => [f.id, f]));

const FeaturePaletteItem: React.FC<{ feature: Feature, onDragStart: (e:
React.DragEvent, featureId: string) => void }> = ({ feature, onDragStart }) => (
    <div
        draggable
        onDragStart={e => onDragStart(e, feature.id)}
        className="p-3 rounded-md bg-gray-50 border border-border flex items-center
        gap-3 cursor-grab hover:bg-gray-100 transition-colors"
    >
        <div className="text-primary flex-shrink-0">{feature.icon}</div>
        <div>
            <h4 className="font-bold text-sm text-text-primary">{feature.name}</h4>
            <p className="text-xs text-text-secondary">{feature.category}</p>
        </div>
    </div>
);

const NodeComponent: React.FC<{
    node: Node;
    feature: Feature;
    onMouseDown: (e: React.MouseEvent, id: number) => void;
    onLinkStart: (e: React.MouseEvent, id: number) => void;
    onLinkEnd: (e: React.MouseEvent, id: number) => void;
}> = ({ node, feature, onMouseDown, onLinkStart, onLinkEnd }) => (
    <div
        className="absolute w-52 bg-surface rounded-lg shadow-md border-2 border-border
        cursor-grab active:cursor-grabbing flex flex-col"
        style={{ left: node.x, top: node.y, transform: 'translate(-50%, -50%)' }}
        onMouseDown={e => onMouseDown(e, node.id)}
        onMouseUp={e => onLinkEnd(e, node.id)}
    >
        <div className="p-2 flex items-center gap-2 border-b border-border">
            <div className="w-5 h-5 text-primary">{feature.icon}</div>
            <span className="text-sm font-semibold truncate text-text-
            primary">{feature.name}</span>
        </div>
        <div className="relative p-3 text-xs text-text-secondary min-h-[40px] flex
        items-center justify-center">
            Workflow Node
            <div
                onMouseDown={e => onLinkStart(e, node.id)}
                className="absolute right-[-9px] top-1/2 -translate-y-1/2 w-4 h-4 bg-primary
                rounded-full border-2 border-surface cursor-crosshair hover:scale-125
                transition-transform"
                title="Drag to connect"
            />
        </div>
    </div>
);
const SVGGrid: React.FC = React.memo(() => (
```

```
    <svg width="100%" height="100%" className="absolute inset-0">
        <defs>
            <pattern id="smallGrid" width="10" height="10" patternUnits="userSpaceOnUse">
                <path d="M 10 0 L 0 0 0 10" fill="none" stroke="rgba(0, 0, 0, 0.05)"
                strokeWidth="0.5"/>
            </pattern>
            <pattern id="grid" width="50" height="50" patternUnits="userSpaceOnUse">
                <rect width="50" height="50" fill="url(#smallGrid)"/>
                <path d="M 50 0 L 0 0 0 50" fill="none" stroke="rgba(0, 0, 0, 0.1)"
                strokeWidth="1"/>
            </pattern>
        </defs>
        <rect width="100%" height="100%" fill="url(#grid)" />
    </svg>
));

export const LogicFlowBuilder: React.FC = () => {
    const [nodes, setNodes] = useState<Node[]>([]);
    const [links, setLinks] = useState<Link[]>([]);
    const [draggingNode, setDraggingNode] = useState<{ id: number; offsetX: number;
    offsetY: number } | null>(null);
    const [linking, setLinking] = useState<{ from: number; fromPos: { x: number; y:
    number }; toPos: { x: number; y: number } } | null>(null);
    const [generatedCode, setGeneratedCode] = useState('');
    const [isGenerating, setIsGenerating] = useState(false);
    const canvasRef = useRef<HTMLDivElement>(null);

    const handleGenerateCode = useCallback(async () => {
        setIsGenerating(true);
        setGeneratedCode('');

        // Simple topological sort (assumes a directed acyclic graph)
        const sortedNodeIds: number[] = [];
        const inDegree = new Map<number, number>();
        nodes.forEach(node => inDegree.set(node.id, 0));
        links.forEach(link => inDegree.set(link.to, (inDegree.get(link.to) || 0) + 1));

        const queue = nodes.filter(node => inDegree.get(node.id) === 0).map(n => n.id);

        while(queue.length > 0) {
            const u = queue.shift()!;
            sortedNodeIds.push(u);
            links.filter(l => l.from === u).forEach(l => {
                inDegree.set(l.to, (inDegree.get(l.to) || 0) - 1);
                if(inDegree.get(l.to) === 0) queue.push(l.to);
            })
        }

        const flowDescription = sortedNodeIds.map((id, index) => {
            const node = nodes.find(n => n.id === id)!;
            const featureInfo = taxonomyMap.get(node.featureId);
            return `Step ${index + 1}: Execute the '${featureInfo?.name}' tool. Description:
            ${featureInfo?.description}. Inputs: ${featureInfo?.inputs}.`;
        }).join('\n');

        try {
            const code = await generatePipelineCode(flowDescription);
            setGeneratedCode(code);
        } catch (e) {
            setGeneratedCode(`// Error generating code: ${e instanceof Error ? e.message :
            'Unknown error'}`);
        } finally {
```

```
        setIsGenerating(false);
    }

}, [nodes, links]);

const handleDragStart = (e: React.DragEvent, featureId: string) => {
    e.dataTransfer.setData('application/json', JSON.stringify({ featureId }));
};

const handleDrop = (e: React.DragEvent) => {
    e.preventDefault();
    if (!canvasRef.current) return;
    const { featureId } = JSON.parse(e.dataTransfer.getData('application/json'));
    const canvasRect = canvasRef.current.getBoundingClientRect();
    const newNode: Node = {
        id: Date.now(),
        featureId,
        x: e.clientX - canvasRect.left,
        y: e.clientY - canvasRect.top,
    };
    setNodes(prev => [...prev, newNode]);
};

const handleNodeMouseDown = (e: React.MouseEvent, id: number) => {
    const node = nodes.find(n => n.id === id);
    if (!node || (e.target as HTMLElement).title === 'Drag to connect') return;
    const canvasRect = canvasRef.current!.getBoundingClientRect();
    setDraggingNode({ id, offsetX: e.clientX - canvasRect.left - node.x, offsetY:
    e.clientY - canvasRect.top - node.y });
};

const handleCanvasMouseMove = (e: React.MouseEvent) => {
    if (!canvasRef.current) return;
    const canvasRect = canvasRef.current.getBoundingClientRect();
    const mouseX = e.clientX - canvasRect.left;
    const mouseY = e.clientY - canvasRect.top;

    if (draggingNode) {
        setNodes(nodes.map(n => n.id === draggingNode.id ? { ...n, x: mouseX -
        draggingNode.offsetX, y: mouseY - draggingNode.offsetY } : n));
    }
    if (linking) {
        setLinking({ ...linking, toPos: { x: mouseX, y: mouseY } });
    }
};

const handleCanvasMouseUp = () => {
    setDraggingNode(null);
    setLinking(null);
};

const handleLinkStart = (e: React.MouseEvent, id: number) => {
    e.stopPropagation();
    const fromNode = nodes.find(n => n.id === id);
    if (!fromNode) return;
    setLinking({ from: id, fromPos: { x: fromNode.x, y: fromNode.y }, toPos: { x:
    fromNode.x, y: fromNode.y } });
};

const handleLinkEnd = (e: React.MouseEvent, id: number) => {
    e.stopPropagation();
    if (linking && linking.from !== id) {
```

```
        setLinks(prev => [...prev, { from: linking.from, to: id }]);
    }
    setLinking(null);
};

const nodePositions = useMemo(() => new Map(nodes.map(n => [n.id, { x: n.x, y:
n.y }])), [nodes]);

return (
    <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
        <header className="mb-6 flex justify-between items-start">
            <div>
                <h1 className="text-3xl font-bold flex items-center"><MapIcon /><span
                className="ml-3">Logic Flow Builder</span></h1>
                <p className="text-text-secondary mt-1">Visually build application logic flows
                and generate pipeline code.</p>
            </div>
            <button onClick={handleGenerateCode} disabled={isGenerating || nodes.length ===
            0} className="btn-primary flex items-center gap-2 px-4 py-2">
                <SparklesIcon /> {isGenerating ? 'Generating...' : 'Generate Code'}
            </button>
        </header>
        <div className="flex-grow flex gap-6 min-h-0">
            <aside className="w-72 flex-shrink-0 bg-surface border border-border p-4
            rounded-lg flex flex-col">
                <h3 className="font-bold mb-3 text-lg">Features</h3>
                <div className="flex-grow overflow-y-auto space-y-3 pr-2">
                    {ALL_FEATURES.map(feature => <FeaturePaletteItem key={feature.id}
                    feature={feature} onDragStart={handleDragStart} />)}
                </div>
            </aside>
            <main
                ref={canvasRef}
                className="flex-grow relative bg-background border-2 border-dashed border-border
                rounded-lg overflow-hidden"
                onDrop={handleDrop}
                onDragOver={e => e.preventDefault()}
                onMouseMove={handleCanvasMouseMove}
                onMouseUp={handleCanvasMouseUp}
                onMouseLeave={handleCanvasMouseUp}
            >
                <SVGGrid />
                <svg width="100%" height="100%" className="absolute inset-0 pointer-events-
                none">
                    {links.map((link, i) => {
                        const fromNode = nodePositions.get(link.from);
                        const toNode = nodePositions.get(link.to);
                        if (!fromNode || !toNode) return null;
                        return <line key={i} x1={fromNode.x} y1={fromNode.y} x2={toNode.x} y2={toNode.y}
                        stroke="var(--color-primary)" strokeWidth="2" markerEnd="url(#arrow)" />;
                    })}
                    {linking && <line x1={linking.fromPos.x} y1={linking.fromPos.y}
                    x2={linking.toPos.x} y2={linking.toPos.y} stroke="var(--color-primary)"
                    strokeWidth="2" strokeDasharray="5,5" />}
                    <defs><marker id="arrow" viewBox="0 0 10 10" refX="8" refY="5" markerWidth="6"
                    markerHeight="6" orient="auto-start-reverse"><path d="M 0 0 L 10 5 L 0 10 z"
                    fill="var(--color-primary)" /></marker></defs>
                </svg>
                {nodes.map(node => {
                    const feature = featuresMap.get(node.featureId);
                    return feature ? <NodeComponent key={node.id} node={node} feature={feature}
                    onMouseDown={handleNodeMouseDown} onLinkStart={handleLinkStart}
```

```
                        onLinkEnd={handleLinkEnd} /> : null;
                    })}
                </main>
            </div>
            {(isGenerating || generatedCode) && (
                <div className="fixed inset-0 bg-gray-900/80 backdrop-blur-sm z-50 flex items-
                center justify-center" onClick={() => setGeneratedCode('')}>
                    <div className="w-full max-w-3xl h-3/4 bg-surface border border-border rounded-
                    lg shadow-2xl p-6 flex flex-col" onClick={e => e.stopPropagation()}>
                        <div className="flex justify-between items-center mb-4">
                            <h2 className="text-xl font-bold">Generated Pipeline Code</h2>
                            <button onClick={() => setGeneratedCode('')}><XMarkIcon/></button>
                        </div>
                        <div className="flex-grow bg-background border border-border rounded-md
                        overflow-auto">
                            {isGenerating && !generatedCode ? <div className="flex justify-center items-
                            center h-full"><LoadingSpinner /></div> : <MarkdownRenderer
                            content={'```javascript\n' + generatedCode + '\n```'} />}
                        </div>
                    </div>
                </div>
            )}
        </div>
    );
};

// ===== AiCodeExplainer_4.tsx =====

import React, { useState, useCallback, useEffect, useMemo, useRef } from
'react';
import { explainCodeStructured } from '../../services/index.ts';
import type { StructuredExplanation } from '../../types.ts';
import { CpuChipIcon } from '../icons.tsx';
import { MarkdownRenderer, LoadingSpinner } from '../shared/index.tsx';

const exampleCode = `const bubbleSort = (arr) => {
  for (let i = 0; i < arr.length; i++) {
    for (let j = 0; j < arr.length - i - 1; j++) {
      if (arr[j] > arr[j + 1]) {
        [arr[j], arr[j + 1]] = [arr[j + 1], arr[j]];
      }
    }
  }
  return arr;
};`;

type ExplanationTab = 'summary' | 'lineByLine' | 'complexity' | 'suggestions';

const simpleSyntaxHighlight = (code: string) => {
    const escapedCode = code
        .replace(/&/g, '&amp;')
        .replace(/</g, '&lt;')
        .replace(/>/g, '&gt;');

    return escapedCode
        .replace(/\b(const|let|var|function|return|if|for|=>|import|from|export|default)
        \b/g, '<span class="text-indigo-600 font-semibold">$1</span>')
        .replace(/(\`|'|")(.*?)(\`|'|")/g, '<span class="text-
        emerald-700">$1$2$3</span>')
        .replace(/(\/\/.*)/g, '<span class="text-gray-500 italic">$1</span>')
        .replace(/(\{|\}|\(|\)|\[|\])/g, '<span class="text-gray-600">$1</span>');
};
```

```
export const AiCodeExplainer: React.FC<{ initialCode?: string }> = ({
initialCode }) => {
    const [code, setCode] = useState<string>(initialCode || exampleCode);
    const [explanation, setExplanation] = useState<StructuredExplanation |
    null>(null);
    const [isLoading, setIsLoading] = useState<boolean>(false);
    const [error, setError] = useState<string>('');
    const [activeTab, setActiveTab] = useState<ExplanationTab>('summary');
    const textareaRef = useRef<HTMLTextAreaElement>(null);
    const preRef = useRef<HTMLPreElement>(null);

    const handleExplain = useCallback(async (codeToExplain: string) => {
        if (!codeToExplain.trim()) {
            setError('Please enter some code to explain.');
            return;
        }
        setIsLoading(true);
        setError('');
        setExplanation(null);
        setActiveTab('summary');
        try {
            const result = await explainCodeStructured(codeToExplain);
            setExplanation(result);
        } catch (err) {
            const errorMessage = err instanceof Error ? err.message : 'An unknown error
            occurred.';
            setError(`Failed to get explanation: ${errorMessage}`);
        } finally {
            setIsLoading(false);
        }
    }, []);

    useEffect(() => {
        if (initialCode) {
            setCode(initialCode);
            handleExplain(initialCode);
        }
    }, [initialCode, handleExplain]);

    const handleScroll = () => {
        if (preRef.current && textareaRef.current) {
            preRef.current.scrollTop = textareaRef.current.scrollTop;
            preRef.current.scrollLeft = textareaRef.current.scrollLeft;
        }
    };

    const highlightedCode = useMemo(() => simpleSyntaxHighlight(code), [code]);

    const renderTabContent = () => {
        if (!explanation) return null;
        switch(activeTab) {
            case 'summary':
                return <MarkdownRenderer content={explanation.summary} />;
            case 'lineByLine':
                return (
                    <div className="space-y-3">
                        {explanation.lineByLine.map((item, index) => (
                            <div key={index} className="p-3 bg-background rounded-md border border-border">
                                <p className="font-mono text-xs text-primary mb-1">Lines: {item.lines}</p>
                                <p className="text-sm">{item.explanation}</p>
                            </div>
                        ))}
```

```
                </div>
            );
        case 'complexity':
            return (
                <div>
                    <p><strong>Time Complexity:</strong> <span className="font-mono text-
                    amber-600">{explanation.complexity.time}</span></p>
                    <p><strong>Space Complexity:</strong> <span className="font-mono text-
                    amber-600">{explanation.complexity.space}</span></p>
                </div>
            );
        case 'suggestions':
            return (
                 <ul className="list-disc list-inside space-y-2">
                    {explanation.suggestions.map((item, index) => <li key={index}>{item}</li>)}
                </ul>
            );
    }
}

return (
    <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
        <header className="mb-6 flex-shrink-0">
            <h1 className="text-3xl font-bold flex items-center">
                <CpuChipIcon />
                <span className="ml-3">AI Code Explainer</span>
            </h1>
            <p className="text-text-secondary mt-1">Get a detailed, structured analysis of
            any code snippet.</p>
        </header>
        <div className="flex-grow grid grid-cols-1 md:grid-cols-2 gap-6 min-h-0">

            {/* Left Column: Code Input */}
            <div className="flex flex-col min-h-0 md:col-span-1">
                <label htmlFor="code-input" className="text-sm font-medium text-text-secondary
                mb-2">Your Code</label>
                <div className="relative flex-grow bg-surface border border-border rounded-md
                focus-within:ring-2 focus-within:ring-primary overflow-hidden">
                    <textarea
                        ref={textareaRef}
                        id="code-input"
                        value={code}
                        onChange={(e) => setCode(e.target.value)}
                        onScroll={handleScroll}
                        placeholder="Paste your code here..."
                        spellCheck="false"
                        className="absolute inset-0 w-full h-full p-4 bg-transparent resize-none font-
                        mono text-sm text-transparent caret-primary outline-none z-10"
                    />
                    <pre
                        ref={preRef}
                        aria-hidden="true"
                        className="absolute inset-0 w-full h-full p-4 font-mono text-sm text-text-
                        primary pointer-events-none z-0 whitespace-pre-wrap overflow-auto no-scrollbar"
                        dangerouslySetInnerHTML={{ __html: highlightedCode + '\n' }}
                    />
                </div>
                <div className="mt-4 flex-shrink-0">
                    <button
                        onClick={() => handleExplain(code)}
                        disabled={isLoading}
                        className="btn-primary w-full flex items-center justify-center px-6 py-3"
```

```
                        >
                            {isLoading ? <LoadingSpinner/> : 'Analyze Code'}
                        </button>
                    </div>
                </div>

                {/* Right Column: AI Analysis */}
                <div className="flex flex-col min-h-0 md:col-span-1">
                    <label className="text-sm font-medium text-text-secondary mb-2">AI
                    Analysis</label>
                    <div className="relative flex-grow flex flex-col bg-surface border border-border
                    rounded-md overflow-hidden">
                        <div className="flex-shrink-0 flex border-b border-border">
                           {(['summary', 'lineByLine', 'complexity', 'suggestions'] as
                           ExplanationTab[]).map(tab => (
                               <button key={tab} onClick={() => setActiveTab(tab)} disabled={!explanation}
                                className={`px-4 py-2 text-sm font-medium capitalize transition-colors
                                ${activeTab === tab ? 'bg-background text-primary font-semibold' : 'text-text-
                                secondary hover:bg-gray-100 disabled:text-gray-400'}`}>
                                   {tab.replace(/([A-Z])/g, ' $1')}
                               </button>
                           ))}
                        </div>
                        <div className="p-4 flex-grow overflow-y-auto">
                            {isLoading && <div className="flex items-center justify-center
                            h-full"><LoadingSpinner /></div>}
                            {error && <p className="text-red-500">{error}</p>}
                            {explanation && !isLoading && renderTabContent()}
                            {!isLoading && !explanation && !error && <div className="text-text-secondary
                            h-full flex items-center justify-center">The analysis will appear here.</div>}
                        </div>
                    </div>
                </div>
            </div>
        </div>
    );
};

// ===== ProjectExplorer_4.tsx =====


import React, { useState, useEffect } from 'react';
import { useGlobalState } from '../../contexts/GlobalStateContext.tsx';
import { getRepos, getRepoTree, getFileContent } from '../../services/index.ts';
import type { Repo, FileNode } from '../../types.ts';
import { FolderIcon, DocumentIcon } from '../icons.tsx';
import { LoadingSpinner } from '../shared/index.tsx';

const FileTree: React.FC<{ node: FileNode, onFileSelect: (path: string) => void
}> = ({ node, onFileSelect }) => {
    const [isOpen, setIsOpen] = useState(true);

    if (node.type === 'file') {
        return (
            <div
                className="flex items-center space-x-2 pl-4 py-1 cursor-pointer hover:bg-
                gray-100 rounded"
                onClick={() => onFileSelect(node.path)}
            >
                <DocumentIcon />
                <span>{node.name}</span>
            </div>
```

```
        );
    }

    return (
        <div>
            <div
                className="flex items-center space-x-2 py-1 cursor-pointer hover:bg-gray-100
                rounded"
                onClick={() => setIsOpen(!isOpen)}
            >
                <div className={`transform transition-transform ${isOpen ? 'rotate-90' :
                ''}`}>■</div>
                <FolderIcon />
                <span className="font-semibold">{node.name}</span>
            </div>
            {isOpen && node.children && (
                <div className="pl-4 border-l border-border ml-3">
                    {node.children.map(child => <FileTree key={child.path} node={child}
                    onFileSelect={onFileSelect} />)}
                </div>
            )}
        </div>
    );
};

export const ProjectExplorer: React.FC = () => {
    const { state, dispatch } = useGlobalState();
    const { githubToken, selectedRepo, projectFiles } = state;
    const [repos, setRepos] = useState<Repo[]>([]);
    const [isLoading, setIsLoading] = useState<'repos' | 'tree' | null>(null);
    const [error, setError] = useState('');
    const [activeFileContent, setActiveFileContent] = useState<string | null>(null);

    useEffect(() => {
        if (githubToken) {
            setIsLoading('repos');
            setError('');
            getRepos()
                .then(setRepos)
                .catch(err => setError(err.message))
                .finally(() => setIsLoading(null));
        } else {
            setRepos([]);
        }
    }, [githubToken]);

    useEffect(() => {
        if (selectedRepo && githubToken) {
            setIsLoading('tree');
            setError('');
            setActiveFileContent(null);
            getRepoTree(selectedRepo.owner, selectedRepo.repo)
                .then(tree => dispatch({ type: 'LOAD_PROJECT_FILES', payload: tree }))
                .catch(err => setError(err.message))
                .finally(() => setIsLoading(null));
        }
    }, [selectedRepo, githubToken, dispatch]);

    const handleFileSelect = async (path: string) => {
        if (!selectedRepo) return;
        try {
            const content = await getFileContent(selectedRepo.owner, selectedRepo.repo,
```

```
            path);
            setActiveFileContent(content);
        } catch (err) {
            setError((err as Error).message);
        }
    };

    if (!githubToken) {
        return (
            <div className="h-full flex flex-col items-center justify-center text-center
            text-text-secondary p-4">
                <FolderIcon />
                <h2 className="text-lg font-semibold mt-2">Connect to GitHub</h2>
                <p>Please go to the "Connections" tab and provide a Personal Access Token to
                explore your repositories.</p>
            </div>
        );
    }

    return (
        <div className="h-full flex flex-col text-text-primary">
            <header className="p-4 border-b border-border flex-shrink-0">
                <h1 className="text-xl font-bold flex items-center"><FolderIcon /><span
                className="ml-3">Project Explorer</span></h1>
                <div className="mt-2">
                    <select
                        value={selectedRepo ? `${selectedRepo.owner}/${selectedRepo.repo}` : ''}
                        onChange={e => {
                            const [owner, repo] = e.target.value.split('/');
                            dispatch({ type: 'SET_SELECTED_REPO', payload: { owner, repo } });
                        }}
                        className="w-full p-2 bg-surface border border-border rounded-md text-sm"
                    >
                        <option value="" disabled>{isLoading === 'repos' ? 'Loading...' : 'Select a
                        repository'}</option>
                        {repos.map(r => <option key={r.id} value={r.full_name}>{r.full_name}</option>)}
                    </select>
                </div>
                {error && <p className="text-red-500 text-xs mt-2">{error}</p>}
            </header>
            <div className="flex-grow flex min-h-0">
                <aside className="w-1/3 bg-background border-r border-border p-4 overflow-y-
                auto">
                    {isLoading === 'tree' && <div className="flex justify-center"><LoadingSpinner
                    /></div>}
                    {projectFiles && <FileTree node={projectFiles} onFileSelect={handleFileSelect}
                    />}
                </aside>
                <main className="flex-1 bg-surface">
                    <pre className="w-full h-full p-4 text-sm overflow-auto whitespace-pre-wrap">
                        <code>{activeFileContent ?? 'Select a file to view its content.'}</code>
                    </pre>
                </main>
            </div>
        </div>
    );
};


// ===== SvgPathEditor_6.tsx =====

import React, { useState, useRef } from 'react';
```

```
import { CodeBracketSquareIcon, ArrowDownTrayIcon } from '../icons.tsx';
import { downloadFile } from '../../services/fileUtils.ts';

const initialPath = "M 20 80 Q 100 20 180 80 T 340 80";

const parsePath = (d: string) => {
    const commands = d.match(/[a-df-z][^a-df-z]*/ig) || [];
    return commands.map((cmdStr, i) => {
        const command = cmdStr[0];
        const args = cmdStr.slice(1).trim().split(/[\s,]+/).map(parseFloat).filter(n =>
        !isNaN(n));
        const points = [];
        for (let j = 0; j < args.length; j += 2) {
            points.push({ x: args[j], y: args[j + 1] });
        }
        return { id: i, command, points };
    });
};

const buildPath = (parsed: any[]) => {
    return parsed.map(cmd => `${cmd.command} ${cmd.points.map((p:any) => `${p.x}
    ${p.y}`).join(' ')}`).join(' ');
};

export const SvgPathEditor: React.FC = () => {
    const [pathData, setPathData] = useState(initialPath);
    const svgRef = useRef<SVGSVGElement>(null);
    const [draggingPoint, setDraggingPoint] = useState<any>(null);
    const parsedPath = parsePath(pathData);

    const handleMouseDown = (e: React.MouseEvent, cmdIndex: number, pointIndex:
    number) => {
        e.stopPropagation();
        setDraggingPoint({ cmdIndex, pointIndex });
    };

    const handleMouseMove = (e: React.MouseEvent) => {
        if (!draggingPoint || !svgRef.current) return;
        const pt = new DOMPoint(e.clientX, e.clientY);
        const svgPoint = pt.matrixTransform(svgRef.current.getScreenCTM()?.inverse());

        const newParsedPath = parsedPath.map((cmd, cIdx) => {
            if (cIdx === draggingPoint.cmdIndex) {
                const newPoints = cmd.points.map((p, pIdx) => {
                    if (pIdx === draggingPoint.pointIndex) {
                        return { x: Math.round(svgPoint.x), y: Math.round(svgPoint.y) };
                    }
                    return p;
                });
                return { ...cmd, points: newPoints };
            }
            return cmd;
        });
        setPathData(buildPath(newParsedPath));
    };

    const handleMouseUp = () => setDraggingPoint(null);

    const handleAddPoint = (e: React.MouseEvent) => {
        if (!svgRef.current) return;
        const pt = new DOMPoint(e.clientX, e.clientY);
        const svgPoint = pt.matrixTransform(svgRef.current.getScreenCTM()?.inverse());
```

```
        const newPathData = `${pathData} L ${Math.round(svgPoint.x)}
        ${Math.round(svgPoint.y)}`;
        setPathData(newPathData);
    };

    const handleDownload = () => {
        const svgContent = `<svg viewBox="0 0 400 160"
        xmlns="http://www.w3.org/2000/svg">
  <path d="${pathData}" stroke="black" fill="transparent" stroke-width="2"/>
</svg>`;
        downloadFile(svgContent, 'path.svg', 'image/svg+xml');
    };

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
            <header className="mb-6"><h1 className="text-3xl font-bold flex items-
            center"><CodeBracketSquareIcon /><span className="ml-3">SVG Path
            Editor</span></h1><p className="text-text-secondary mt-1">Visually create and
            manipulate SVG path data by dragging points.</p></header>
            <div className="flex-grow grid grid-cols-1 lg:grid-cols-2 gap-6 h-full overflow-
            hidden">
                <div className="flex flex-col h-full overflow-y-auto">
                    <div className="flex justify-between items-center mb-2">
                        <label htmlFor="path-input" className="text-sm font-medium text-text-
                        secondary">Path Data (d attribute)</label>
                         <button onClick={handleDownload} className="flex items-center gap-1 px-3 py-1
                         bg-gray-100 text-xs rounded-md hover:bg-gray-200">
                            <ArrowDownTrayIcon className="w-4 h-4"/> Download SVG
                        </button>
                    </div>
                    <textarea id="path-input" value={pathData} onChange={(e) =>
                    setPathData(e.target.value)} className="h-24 p-4 bg-surface border border-border
                    rounded-md resize-y font-mono text-sm text-primary" />
                     <div className="flex-grow mt-4 p-4 bg-surface border-2 border-dashed border-
                     border rounded-md overflow-hidden flex items-center justify-center
                     min-h-[200px]">
                        <svg ref={svgRef} viewBox="0 0 400 160" className="w-full h-full cursor-
                        crosshair" onMouseMove={handleMouseMove} onMouseUp={handleMouseUp}
                        onMouseLeave={handleMouseUp} onDoubleClick={handleAddPoint}>
                            <rect width="400" height="160" fill="var(--color-background)" />
                             <path d={pathData} stroke="var(--color-primary)" fill="transparent"
                             strokeWidth="2" />
                             {parsedPath.flatMap((cmd, cmdIndex) =>
                                 cmd.points.map((p, pointIndex) => (
                                     <circle
                                         key={`${cmd.id}-${pointIndex}`}
                                         cx={p.x}
                                         cy={p.y}
                                         r="5"
                                         fill={cmd.command.toLowerCase() === 'c' || cmd.command.toLowerCase() === 'q' ||
                                         cmd.command.toLowerCase() === 's' || cmd.command.toLowerCase() === 't' ?
                                         '#fde047' : '#f87171'}
                                         stroke="var(--color-surface)"
                                         strokeWidth="2"
                                         className="cursor-move hover:stroke-primary"
                                         onMouseDown={(e) => handleMouseDown(e, cmdIndex, pointIndex)}
                                     />
                                 ))
                             )}
                        </svg>
                    </div>
                    <p className="text-xs text-center text-text-secondary mt-2">Double-click on the
```

```
                    canvas to add a new point.</p>
                </div>
                <div className="flex flex-col h-full">
                    <label className="text-sm font-medium text-text-secondary mb-2">Parsed
                    Commands</label>
                    <div className="flex-grow p-2 bg-background border border-border rounded-md
                    overflow-y-auto font-mono text-xs space-y-2">
                        {parsedPath.map(cmd => (
                            <div key={cmd.id} className="p-2 bg-surface rounded">
                                <span className="font-bold text-amber-600">{cmd.command}</span>
                                <span className="text-text-secondary"> {cmd.points.map(p =>
                                `(${p.x},${p.y})`).join(' ')}</span>
                            </div>
                        ))}
                    </div>
                </div>
            </div>
        </div>
    );
};

// ===== ScreenshotToComponent_6.tsx =====

import React, { useState, useCallback, useRef } from 'react';
import { generateComponentFromImageStream } from
'../../services/geminiService.ts';
import { PhotoIcon, ArrowDownTrayIcon } from '../icons.tsx';
import { LoadingSpinner, MarkdownRenderer } from '../shared/index.tsx';
import { fileToBase64, blobToDataURL, downloadFile } from
'../../services/fileUtils.ts';

export const ScreenshotToComponent: React.FC = () => {
    const [previewImage, setPreviewImage] = useState<string | null>(null);
    const [rawCode, setRawCode] = useState('');
    const [isLoading, setIsLoading] = useState(false);
    const [error, setError] = useState('');
    const fileInputRef = useRef<HTMLInputElement>(null);

    const handleGenerate = async (base64Image: string) => {
        setIsLoading(true);
        setError('');
        setRawCode('');
        try {
            const stream = generateComponentFromImageStream(base64Image);
            let fullResponse = '';
            for await (const chunk of stream) {
                fullResponse += chunk;
                setRawCode(fullResponse.replace(/^```(?:\w+\n)?/, '').replace(/```$/, ''));
            }
        } catch (err) {
            setError(err instanceof Error ? err.message : 'An unknown error occurred.');
        } finally {
            setIsLoading(false);
        }
    };

    const processImageBlob = async (blob: Blob) => {
        try {
            const [dataUrl, base64Image] = await Promise.all([blobToDataURL(blob),
            fileToBase64(blob as File)]);
            setPreviewImage(dataUrl);
            handleGenerate(base64Image);
```

```
    } catch (e) {
        setError('Could not process the image.');
    }
};

const handlePaste = useCallback(async (event: React.ClipboardEvent) => {
    const items = event.clipboardData.items;
    for (const item of items) {
        if (item.type.indexOf('image') !== -1) {
            const blob = item.getAsFile();
            if (blob) await processImageBlob(blob);
            return;
        }
    }
}, []);

const handleFileChange = async (event: React.ChangeEvent<HTMLInputElement>) => {
    const file = event.target.files?.[0];
    if (file) await processImageBlob(file);
};

return (
    <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
        <header className="mb-6"><h1 className="text-3xl font-bold flex items-
        center"><PhotoIcon /><span className="ml-3">AI Screenshot-to-
        Component</span></h1><p className="text-text-secondary mt-1">Paste or upload a
        screenshot of a UI element to generate React/Tailwind code.</p></header>
        <div className="flex-grow grid grid-cols-1 lg:grid-cols-2 gap-6 min-h-0">
            <div onPaste={handlePaste} className="flex flex-col items-center justify-center
            bg-surface p-6 rounded-lg border-2 border-dashed border-border focus:outline-
            none focus:border-primary overflow-y-auto" tabIndex={0}>
                {previewImage ? (<img src={previewImage} alt="Pasted content" className="max-w-
                full max-h-full object-contain rounded-md shadow-lg" />) : (<div
                className="text-center text-text-secondary">
                        <h2 className="text-xl font-bold text-text-primary">Paste an image here</h2>
                        <p className="mb-2">(Cmd/Ctrl + V)</p>
                        <p className="text-sm">or</p>
                        <button onClick={() => fileInputRef.current?.click()} className="mt-2 btn-
                        primary px-4 py-2 text-sm">Upload File</button>
                        <input type="file" ref={fileInputRef} onChange={handleFileChange}
                        accept="image/*" className="hidden"/>
                    </div>)}
            </div>
            <div className="flex flex-col h-full">
                <div className="flex justify-between items-center mb-2">
                    <label className="text-sm font-medium text-text-secondary">Generated
                    Code</label>
                    {rawCode && !isLoading && (
                        <div className="flex items-center gap-2">
                            <button onClick={() => navigator.clipboard.writeText(rawCode)} className="px-3
                            py-1 bg-gray-100 text-xs rounded-md hover:bg-gray-200">Copy Code</button>
                            <button onClick={() => downloadFile(rawCode, 'Component.tsx',
                            'text/typescript')} className="flex items-center gap-1 px-3 py-1 bg-gray-100
                            text-xs rounded-md hover:bg-gray-200">
                                <ArrowDownTrayIcon className="w-4 h-4" /> Download
                            </button>
                        </div>
                    )}
                </div>
                <div className="flex-grow bg-background border border-border rounded-md
                overflow-y-auto">
                    {isLoading && (<div className="flex items-center justify-center
```

```
                        h-full"><LoadingSpinner /></div>)}
                        {error && <p className="p-4 text-red-500">{error}</p>}
                        {rawCode && !isLoading && <MarkdownRenderer
                        content={`\`\`\`tsx\n${rawCode}\n\`\`\`} />}
                        {!isLoading && !rawCode && !error && (<div className="text-text-secondary h-full
                        flex items-center justify-center">Generated component code will appear
                        here.</div>)}
                    </div>
                </div>
            </div>
        </div>
    );
};

// ===== Connections_4.tsx =====

import React, { useState, useEffect } from 'react';
import { GithubIcon } from '../icons.tsx';
import { useGlobalState } from '../../contexts/GlobalStateContext.tsx';
import { initializeOctokit, validateToken } from '../../services/index.ts';
import { LoadingSpinner } from '../shared/LoadingSpinner.tsx';
import type { User } from '../../types.ts';

const GitHubConnection: React.FC = () => {
    const { state, dispatch } = useGlobalState();
    const { githubToken, isGithubConnected, githubUser } = state;
    const [tokenInput, setTokenInput] = useState(githubToken || '');
    const [status, setStatus] = useState<'idle' | 'loading' | 'success' |
    'error'>('idle');
    const [error, setError] = useState('');

    useEffect(() => {
        // When the component loads, if there's a token in the global state,
        // try to initialize octokit with it.
        if (githubToken && !isGithubConnected) {
             handleConnect(githubToken);
        }
    // eslint-disable-next-line react-hooks/exhaustive-deps
    }, [githubToken]);

    const handleConnect = async (tokenToConnect: string) => {
        if (!tokenToConnect.trim()) {
            setError('Please enter a token.');
            setStatus('error');
            return;
        }
        setStatus('loading');
        setError('');
        try {
            const user: User = await validateToken(tokenToConnect);
            initializeOctokit(tokenToConnect);
            dispatch({ type: 'SET_GITHUB_TOKEN', payload: { token: tokenToConnect, user }
            });
            setStatus('success');
        } catch (err) {
            setError(err instanceof Error ? `Invalid Token: ${err.message}` : 'An unknown
            error occurred.');
            setStatus('error');
            initializeOctokit('');
            dispatch({ type: 'SET_GITHUB_TOKEN', payload: { token: null, user: null } });
        }
    };
```

```
    const handleDisconnect = () => {
        setTokenInput('');
        initializeOctokit('');
        dispatch({ type: 'SET_GITHUB_TOKEN', payload: { token: null, user: null } });
        setStatus('idle');
    };

    return (
        <div className="bg-surface border border-border rounded-lg p-6">
            <div className="flex items-center justify-between">
                <div className="flex items-center gap-4">
                    <div className="w-10 h-10"><GithubIcon /></div>
                    <div>
                        <h3 className="text-lg font-bold text-text-primary">GitHub</h3>
                        {isGithubConnected && githubUser ? (
                             <p className="text-sm text-green-600">Connected as {githubUser.login}</p>
                        ) : (
                             <p className="text-sm text-text-secondary">Not Connected</p>
                        )}
                    </div>
                </div>
                 {isGithubConnected && (
                    <button onClick={handleDisconnect} className="px-4 py-2 bg-red-500/10 text-
                    red-600 font-semibold rounded-lg hover:bg-red-500/20">
                        Disconnect
                    </button>
                 )}
            </div>

            {!isGithubConnected && (
                <div className="mt-4 pt-4 border-t border-border">
                    <label htmlFor="github-pat" className="block text-sm font-medium text-text-
                    secondary mb-1">Personal Access Token (Classic)</label>
                    <div className="flex gap-2">
                        <input
                            id="github-pat"
                            type="password"
                            value={tokenInput}
                            onChange={(e) => setTokenInput(e.target.value)}
                            placeholder="ghp_..."
                            className="flex-grow p-2 bg-background border border-border rounded-md text-sm"
                        />
                         <button onClick={() => handleConnect(tokenInput)} disabled={status ===
                         'loading'} className="btn-primary px-6 py-2 flex items-center justify-center
                         min-w-[100px]">
                            {status === 'loading' ? <LoadingSpinner /> : 'Connect'}
                        </button>
                    </div>
                    {error && <p className="text-red-500 text-xs mt-2">{error}</p>}
                    <p className="text-xs text-text-secondary mt-2">
                        Your token is stored only in your browser's local storage. Required scopes:
                        `repo`, `read:user`.
                    </p>
                </div>
            )}
        </div>
    );
};

export const Connections: React.FC = () => {
    return (
        <div className="h-full p-4 sm:p-6 lg:p-8">
```

```
            <div className="max-w-4xl mx-auto">
                <header className="mb-8">
                    <h1 className="text-4xl font-extrabold tracking-tight">Connections</h1>
                    <p className="mt-2 text-lg text-text-secondary">Manage your GitHub connection to
                    unlock powerful workflows.</p>
                </header>

                <div className="space-y-6">
                    <GitHubConnection />
                </div>
            </div>
        </div>
    );
};


// ===== ColorPaletteGenerator_6.tsx =====

import React, { useState, useCallback } from 'react';
import { HexColorPicker } from 'react-colorful';
import { generateColorPalette, downloadFile } from '../../services/index.ts';
import { SparklesIcon, ArrowDownTrayIcon } from '../icons.tsx';
import { LoadingSpinner } from '../shared/index.tsx';

interface PreviewColors {
    cardBg: string;
    pillBg: string;
    pillText: string;
    buttonBg: string;
}

const PreviewCard: React.FC<{ palette: string[], colors: PreviewColors,
setColors: React.Dispatch<React.SetStateAction<PreviewColors>> }> = ({ palette,
colors, setColors }) => {

    const ColorSelector: React.FC<{ label: string, value: string, onChange: (val:
    string) => void }> = ({ label, value, onChange }) => (
        <div className="flex items-center justify-between text-sm">
            <label className="text-text-primary">{label}</label>
            <div className="flex items-center gap-2">
                {palette.map(color => (
                     <button
                        key={color}
                        onClick={() => onChange(color)}
                        className={`w-5 h-5 rounded-full border border-gray-300 ${value === color ?
                        'ring-2 ring-primary ring-offset-1' : ''}`}
                        style={{ backgroundColor: color }}
                        title={color}
                     />
                ))}
            </div>
        </div>
    );

    return (
        <div className="bg-surface p-4 rounded-lg border border-border w-full max-w-sm">
            <h3 className="text-lg font-bold mb-4 text-text-primary">Live Preview</h3>
            <div className="p-8 rounded-xl mb-4" style={{ backgroundColor: colors.cardBg }}>
                <div className="px-4 py-1 rounded-full text-center text-sm inline-block"
                style={{ backgroundColor: colors.pillBg, color: colors.pillText }}>
                    New Feature
                </div>
```

```
                <div className="mt-8 text-center">
                     <button className="px-6 py-2 rounded-lg font-bold" style={{ backgroundColor:
                     colors.buttonBg, color: colors.cardBg }}>
                        Get Started
                    </button>
                </div>
            </div>
            <div className="space-y-3">
                <ColorSelector label="Card Background" value={colors.cardBg} onChange={val =>
                setColors(c => ({...c, cardBg: val}))} />
                <ColorSelector label="Pill Background" value={colors.pillBg} onChange={val =>
                setColors(c => ({...c, pillBg: val}))} />
                <ColorSelector label="Pill Text" value={colors.pillText} onChange={val =>
                setColors(c => ({...c, pillText: val}))} />
                <ColorSelector label="Button Background" value={colors.buttonBg} onChange={val
                => setColors(c => ({...c, buttonBg: val}))} />
            </div>
        </div>
    );
};

export const ColorPaletteGenerator: React.FC = () => {
    const [baseColor, setBaseColor] = useState("#0047AB");
    const [palette, setPalette] = useState<string[]>(['#F0F2F5', '#CCD3E8',
    '#99AADD', '#6688D1', '#3366CC', '#0047AB']);
    const [isLoading, setIsLoading] = useState<boolean>(false);
    const [error, setError] = useState<string>('');
    const [previewColors, setPreviewColors] = useState<PreviewColors>({
        cardBg: '#F0F2F5', pillBg: '#CCD3E8', pillText: '#0047AB', buttonBg: '#0047AB'
    });

    const handleGenerate = useCallback(async () => {
        setIsLoading(true);
        setError('');
        try {
            const result = await generateColorPalette(baseColor);
            setPalette(result.colors);
            setPreviewColors({
                cardBg: result.colors[0],
                pillBg: result.colors[2],
                pillText: result.colors[5],
                buttonBg: result.colors[5],
            })
        } catch (err) {
            const errorMessage = err instanceof Error ? err.message : 'An unknown error
            occurred.';
            setError(`Failed to generate palette: ${errorMessage}`);
        } finally {
            setIsLoading(false);
        }
    }, [baseColor]);

    const downloadColors = () => {
        const cssContent = `:root {\n${palette.map((c, i) => `  --color-palette-${i+1}:
        ${c};`).join('\n')}\n}`;
        downloadFile(cssContent, 'palette.css', 'text/css');
    };

    const downloadCard = () => {
        const htmlContent = `
<div class="card">
  <div class="pill">New Feature</div>
```

```
  <button class="button">Get Started</button>
</div>
        `;
        const cssContent = `
.card {
  background-color: ${previewColors.cardBg};
  padding: 2rem;
  border-radius: 1rem;
  text-align: center;
}
.pill {
  background-color: ${previewColors.pillBg};
  color: ${previewColors.pillText};
  display: inline-block;
  padding: 0.25rem 1rem;
  border-radius: 9999px;
  text-align: center;
  font-size: 0.875rem;
}
.button {
  margin-top: 2rem;
  background-color: ${previewColors.buttonBg};
  color: ${previewColors.cardBg};
  padding: 0.5rem 1.5rem;
  border-radius: 0.5rem;
  font-weight: bold;
  border: none;
  cursor: pointer;
}
        `;
        const combined = `<!-- HTML -->\n${htmlContent}\n\n<!-- CSS
        -->\n<style>\n${cssContent}\n</style>`;
        downloadFile(combined, 'preview-card.html', 'text/html');
    }

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
            <header className="mb-6 text-center">
                <h1 className="text-3xl font-bold flex items-center justify-center">
                    <SparklesIcon />
                    <span className="ml-3">AI Color Palette Generator</span>
                </h1>
                <p className="text-text-secondary mt-1">Pick a base color, let Gemini design a
                palette, and preview it on a UI card.</p>
            </header>
            <div className="flex-grow flex flex-col lg:flex-row items-center justify-center
            gap-8">
                <div className="flex flex-col items-center gap-4">
                     <HexColorPicker color={baseColor} onChange={setBaseColor} className="!w-64
                     !h-64"/>
                     <div className="p-2 bg-surface rounded-md font-mono text-lg border border-
                     border" style={{color: baseColor}}>{baseColor}</div>
                      <button onClick={handleGenerate} disabled={isLoading} className="btn-primary
                      w-full flex items-center justify-center px-6 py-3">
                        {isLoading ? <LoadingSpinner /> : 'Generate Palette'}
                    </button>
                    {error && <p className="text-red-500 text-sm mt-2">{error}</p>}
                </div>
                <div className="flex flex-col gap-2 w-full max-w-sm">
                    <label className="text-sm font-medium text-text-secondary mb-2">Generated
                    Palette:</label>
                    {isLoading ? (
```

```
                        <div className="flex items-center justify-center h-48"><LoadingSpinner /></div>
                    ) : (
                        palette.map((color) => (
                            <div key={color} className="group flex items-center justify-between p-4 rounded-
                            md shadow-sm border border-border" style={{ backgroundColor: color }}>
                                <span className="font-mono font-bold text-black/70 mix-blend-
                                overlay">{color}</span>
                                <button onClick={() => navigator.clipboard.writeText(color)}
                                className="opacity-0 group-hover:opacity-100 transition-opacity bg-white/30
                                hover:bg-white/50 px-3 py-1 rounded text-xs text-black font-semibold backdrop-
                                blur-sm">Copy</button>
                            </div>
                        ))
                    )}
                    <div className="flex gap-2 mt-2">
                        <button onClick={downloadColors} className="flex-1 flex items-center justify-
                        center gap-2 text-sm py-2 bg-gray-100 border border-border rounded-md hover:bg-
                        gray-200"><ArrowDownTrayIcon className="w-4 h-4"/> Download Colors</button>
                        <button onClick={downloadCard} className="flex-1 flex items-center justify-
                        center gap-2 text-sm py-2 bg-gray-100 border border-border rounded-md hover:bg-
                        gray-200"><ArrowDownTrayIcon className="w-4 h-4"/> Download Card</button>
                    </div>
                </div>
                {!isLoading && <PreviewCard palette={palette} colors={previewColors}
                setColors={setPreviewColors} />}
            </div>
        </div>
    );
};

// ===== LogicFlowBuilder_9.tsx =====

import React, { useState, useRef, useMemo, useCallback } from 'react';
import { ALL_FEATURES } from './index.ts';
import { FEATURE_TAXONOMY } from '../../services/taxonomyService.ts';
import { generatePipelineCode } from '../../services/geminiService.ts';
import type { Feature } from '../../types.ts';
import { MapIcon, SparklesIcon, XMarkIcon } from '../icons.tsx';
import { LoadingSpinner, MarkdownRenderer } from '../shared/index.tsx';

interface Node {
    id: number;
    featureId: string;
    x: number;
    y: number;
}

interface Link {
    from: number;
    to: number;
}

const featuresMap = new Map(ALL_FEATURES.map(f => [f.id, f]));
const taxonomyMap = new Map(FEATURE_TAXONOMY.map(f => [f.id, f]));

const FeaturePaletteItem: React.FC<{ feature: Feature, onDragStart: (e:
React.DragEvent, featureId: string) => void }> = ({ feature, onDragStart }) => (
    <div
        draggable
        onDragStart={e => onDragStart(e, feature.id)}
        className="p-3 rounded-md bg-gray-50 border border-border flex items-center
        gap-3 cursor-grab hover:bg-gray-100 transition-colors"
```

```
    >
        <div className="text-primary flex-shrink-0">{feature.icon}</div>
        <div>
            <h4 className="font-bold text-sm text-text-primary">{feature.name}</h4>
            <p className="text-xs text-text-secondary">{feature.category}</p>
        </div>
    </div>
);

const NodeComponent: React.FC<{
    node: Node;
    feature: Feature;
    onMouseDown: (e: React.MouseEvent, id: number) => void;
    onLinkStart: (e: React.MouseEvent, id: number) => void;
    onLinkEnd: (e: React.MouseEvent, id: number) => void;
}> = ({ node, feature, onMouseDown, onLinkStart, onLinkEnd }) => (
    <div
        className="absolute w-52 bg-surface rounded-lg shadow-md border-2 border-border
        cursor-grab active:cursor-grabbing flex flex-col"
        style={{ left: node.x, top: node.y, transform: 'translate(-50%, -50%)' }}
        onMouseDown={e => onMouseDown(e, node.id)}
        onMouseUp={e => onLinkEnd(e, node.id)}
    >
        <div className="p-2 flex items-center gap-2 border-b border-border">
            <div className="w-5 h-5 text-primary">{feature.icon}</div>
            <span className="text-sm font-semibold truncate text-text-
            primary">{feature.name}</span>
        </div>
        <div className="relative p-3 text-xs text-text-secondary min-h-[40px] flex
        items-center justify-center">
            Workflow Node
            <div
                onMouseDown={e => onLinkStart(e, node.id)}
                className="absolute right-[-9px] top-1/2 -translate-y-1/2 w-4 h-4 bg-primary
                rounded-full border-2 border-surface cursor-crosshair hover:scale-125
                transition-transform"
                title="Drag to connect"
            />
        </div>
    </div>
);

const SVGGrid: React.FC = React.memo(() => (
    <svg width="100%" height="100%" className="absolute inset-0">
        <defs>
            <pattern id="smallGrid" width="10" height="10" patternUnits="userSpaceOnUse">
                <path d="M 10 0 L 0 0 0 10" fill="none" stroke="rgba(0, 0, 0, 0.05)"
                strokeWidth="0.5"/>
            </pattern>
            <pattern id="grid" width="50" height="50" patternUnits="userSpaceOnUse">
                <rect width="50" height="50" fill="url(#smallGrid)"/>
                <path d="M 50 0 L 0 0 0 50" fill="none" stroke="rgba(0, 0, 0, 0.1)"
                strokeWidth="1"/>
            </pattern>
        </defs>
        <rect width="100%" height="100%" fill="url(#grid)" />
    </svg>
));

export const LogicFlowBuilder: React.FC = () => {
    const [nodes, setNodes] = useState<Node[]>([]);
    const [links, setLinks] = useState<Link[]>([]);
```

```
const [draggingNode, setDraggingNode] = useState<{ id: number; offsetX: number;
offsetY: number } | null>(null);
const [linking, setLinking] = useState<{ from: number; fromPos: { x: number; y:
number }; toPos: { x: number; y: number } } | null>(null);
const [generatedCode, setGeneratedCode] = useState('');
const [isGenerating, setIsGenerating] = useState(false);
const canvasRef = useRef<HTMLDivElement>(null);

const handleGenerateCode = useCallback(async () => {
    setIsGenerating(true);
    setGeneratedCode('');

    const sortedNodeIds: number[] = [];
    const inDegree = new Map<number, number>();
    nodes.forEach(node => inDegree.set(node.id, 0));
    links.forEach(link => inDegree.set(link.to, (inDegree.get(link.to) || 0) + 1));

    const queue = nodes.filter(node => inDegree.get(node.id) === 0).map(n => n.id);

    while(queue.length > 0) {
        const u = queue.shift()!;
        sortedNodeIds.push(u);
        links.filter(l => l.from === u).forEach(l => {
            inDegree.set(l.to, (inDegree.get(l.to) || 0) - 1);
            if(inDegree.get(l.to) === 0) queue.push(l.to);
        })
    }

    const flowDescription = sortedNodeIds.map((id, index) => {
        const node = nodes.find(n => n.id === id)!;
        const featureInfo = taxonomyMap.get(node.featureId);
        return `Step ${index + 1}: Execute the '${featureInfo?.name}' tool. Description:
        ${featureInfo?.description}. Inputs: ${featureInfo?.inputs}.`;
    }).join('\n');

    try {
        const code = await generatePipelineCode(flowDescription);
        setGeneratedCode(code);
    } catch (e) {
        setGeneratedCode(`// Error generating code: ${e instanceof Error ? e.message :
        'Unknown error'}`);
    } finally {
        setIsGenerating(false);
    }

}, [nodes, links]);

const handleDragStart = (e: React.DragEvent, featureId: string) => {
    e.dataTransfer.setData('application/json', JSON.stringify({ featureId }));
};

const handleDrop = (e: React.DragEvent) => {
    e.preventDefault();
    if (!canvasRef.current) return;
    const { featureId } = JSON.parse(e.dataTransfer.getData('application/json'));
    const canvasRect = canvasRef.current.getBoundingClientRect();
    const newNode: Node = {
        id: Date.now(),
        featureId,
        x: e.clientX - canvasRect.left,
        y: e.clientY - canvasRect.top,
    };
```

```
    setNodes(prev => [...prev, newNode]);
};

const handleNodeMouseDown = (e: React.MouseEvent, id: number) => {
    const node = nodes.find(n => n.id === id);
    if (!node || (e.target as HTMLElement).title === 'Drag to connect') return;
    const canvasRect = canvasRef.current!.getBoundingClientRect();
    setDraggingNode({ id, offsetX: e.clientX - canvasRect.left - node.x, offsetY:
    e.clientY - canvasRect.top - node.y });
};

const handleCanvasMouseMove = (e: React.MouseEvent) => {
    if (!canvasRef.current) return;
    const canvasRect = canvasRef.current.getBoundingClientRect();
    const mouseX = e.clientX - canvasRect.left;
    const mouseY = e.clientY - canvasRect.top;

    if (draggingNode) {
        setNodes(nodes.map(n => n.id === draggingNode.id ? { ...n, x: mouseX -
        draggingNode.offsetX, y: mouseY - draggingNode.offsetY } : n));
    }
    if (linking) {
        setLinking({ ...linking, toPos: { x: mouseX, y: mouseY } });
    }
};

const handleCanvasMouseUp = () => {
    setDraggingNode(null);
    setLinking(null);
};

const handleLinkStart = (e: React.MouseEvent, id: number) => {
    e.stopPropagation();
    const fromNode = nodes.find(n => n.id === id);
    if (!fromNode) return;
    setLinking({ from: id, fromPos: { x: fromNode.x, y: fromNode.y }, toPos: { x:
    fromNode.x, y: fromNode.y } });
};

const handleLinkEnd = (e: React.MouseEvent, id: number) => {
    e.stopPropagation();
    if (linking && linking.from !== id) {
        setLinks(prev => [...prev, { from: linking.from, to: id }]);
    }
    setLinking(null);
};

const nodePositions = useMemo(() => new Map(nodes.map(n => [n.id, { x: n.x, y:
n.y }])), [nodes]);

return (
    <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
        <header className="mb-6 flex justify-between items-start">
            <div>
                <h1 className="text-3xl font-bold flex items-center"><MapIcon /><span
                className="ml-3">Logic Flow Builder</span></h1>
                <p className="text-text-secondary mt-1">Visually build application logic flows
                and generate pipeline code.</p>
            </div>
            <button onClick={handleGenerateCode} disabled={isGenerating || nodes.length ===
            0} className="btn-primary flex items-center gap-2 px-4 py-2">
                <SparklesIcon /> {isGenerating ? 'Generating...' : 'Generate Code'}
```

```
    </button>
</header>
<div className="flex-grow flex gap-6 min-h-0">
    <aside className="w-72 flex-shrink-0 bg-surface border border-border p-4
    rounded-lg flex flex-col">
        <h3 className="font-bold mb-3 text-lg">Features</h3>
        <div className="flex-grow overflow-y-auto space-y-3 pr-2">
            {ALL_FEATURES.map(feature => <FeaturePaletteItem key={feature.id}
            feature={feature} onDragStart={handleDragStart} />)}
        </div>
    </aside>
    <main
        ref={canvasRef}
        className="flex-grow relative bg-background border-2 border-dashed border-border
        rounded-lg overflow-hidden"
        onDrop={handleDrop}
        onDragOver={e => e.preventDefault()}
        onMouseMove={handleCanvasMouseMove}
        onMouseUp={handleCanvasMouseUp}
        onMouseLeave={handleCanvasMouseUp}
    >
        <SVGGrid />
        <svg width="100%" height="100%" className="absolute inset-0 pointer-events-
        none">
            {links.map((link, i) => {
                const fromNode = nodePositions.get(link.from);
                const toNode = nodePositions.get(link.to);
                if (!fromNode || !toNode) return null;
                return <line key={i} x1={fromNode.x} y1={fromNode.y} x2={toNode.x} y2={toNode.y}
                stroke="var(--color-primary)" strokeWidth="2" markerEnd="url(#arrow)" />;
            })}
            {linking && <line x1={linking.fromPos.x} y1={linking.fromPos.y}
            x2={linking.toPos.x} y2={linking.toPos.y} stroke="var(--color-primary)"
            strokeWidth="2" strokeDasharray="5,5" />}
            <defs><marker id="arrow" viewBox="0 0 10 10" refX="8" refY="5" markerWidth="6"
            markerHeight="6" orient="auto-start-reverse"><path d="M 0 0 L 10 5 L 0 10 z"
            fill="var(--color-primary)" /></marker></defs>
        </svg>
        {nodes.map(node => {
            const feature = featuresMap.get(node.featureId);
            return feature ? <NodeComponent key={node.id} node={node} feature={feature}
            onMouseDown={handleNodeMouseDown} onLinkStart={handleLinkStart}
            onLinkEnd={handleLinkEnd} /> : null;
        })}
    </main>
</div>
{(isGenerating || generatedCode) && (
    <div className="fixed inset-0 bg-gray-900/80 backdrop-blur-sm z-50 flex items-
    center justify-center" onClick={() => setGeneratedCode('')}>
        <div className="w-full max-w-3xl h-3/4 bg-surface border border-border rounded-
        lg shadow-2xl p-6 flex flex-col" onClick={e => e.stopPropagation()}>
            <div className="flex justify-between items-center mb-4">
                <h2 className="text-xl font-bold">Generated Pipeline Code</h2>
                <button onClick={() => setGeneratedCode('')}><XMarkIcon/></button>
            </div>
            <div className="flex-grow bg-background border border-border rounded-md
            overflow-auto">
                {isGenerating && !generatedCode ? <div className="flex justify-center items-
                center h-full"><LoadingSpinner /></div> : <MarkdownRenderer
                content={'```javascript\n' + generatedCode + '\n```'} />}
            </div>
        </div>
```

```
                </div>
            )}
        </div>
    );
};

// ===== AiCodeExplainer_7.tsx =====

import React, { useState, useCallback, useEffect, useMemo, useRef } from
'react';
import { explainCodeStructured } from '../../services/geminiService.ts';
import type { StructuredExplanation } from '../../types.ts';
import { CpuChipIcon } from '../icons.tsx';
import { MarkdownRenderer, LoadingSpinner } from '../shared/index.tsx';

const exampleCode = `const bubbleSort = (arr) => {
  for (let i = 0; i < arr.length; i++) {
    for (let j = 0; j < arr.length - i - 1; j++) {
      if (arr[j] > arr[j + 1]) {
        [arr[j], arr[j + 1]] = [arr[j + 1], arr[j]];
      }
    }
  }
  return arr;
};`;

type ExplanationTab = 'summary' | 'lineByLine' | 'complexity' | 'suggestions';

const simpleSyntaxHighlight = (code: string) => {
    const escapedCode = code
        .replace(/&/g, '&amp;')
        .replace(/</g, '&lt;')
        .replace(/>/g, '&gt;');

    return escapedCode
        .replace(/\b(const|let|var|function|return|if|for|=>|import|from|export|default)
        \b/g, '<span class="text-indigo-600 font-semibold">$1</span>')
        .replace(/(\`|'|")(.*?)(\`|'|")/g, '<span class="text-
        emerald-700">$1$2$3</span>')
        .replace(/(\/\/.*)/g, '<span class="text-gray-500 italic">$1</span>')
        .replace(/(\{|\}|\(|\)|\[|\])/g, '<span class="text-gray-600">$1</span>');
};

export const AiCodeExplainer: React.FC<{ initialCode?: string }> = ({
initialCode }) => {
    const [code, setCode] = useState<string>(initialCode || exampleCode);
    const [explanation, setExplanation] = useState<StructuredExplanation |
    null>(null);
    const [isLoading, setIsLoading] = useState<boolean>(false);
    const [error, setError] = useState<string>('');
    const [activeTab, setActiveTab] = useState<ExplanationTab>('summary');
    const textareaRef = useRef<HTMLTextAreaElement>(null);
    const preRef = useRef<HTMLPreElement>(null);

    const handleExplain = useCallback(async (codeToExplain: string) => {
        if (!codeToExplain.trim()) {
            setError('Please enter some code to explain.');
            return;
        }
        setIsLoading(true);
        setError('');
        setExplanation(null);
```

```
    setActiveTab('summary');
    try {
        const result = await explainCodeStructured(codeToExplain);
        setExplanation(result);
    } catch (err) {
        const errorMessage = err instanceof Error ? err.message : 'An unknown error
        occurred.';
        setError(`Failed to get explanation: ${errorMessage}`);
    } finally {
        setIsLoading(false);
    }
}, []);

useEffect(() => {
    if (initialCode) {
        setCode(initialCode);
        handleExplain(initialCode);
    }
}, [initialCode, handleExplain]);

const handleScroll = () => {
    if (preRef.current && textareaRef.current) {
        preRef.current.scrollTop = textareaRef.current.scrollTop;
        preRef.current.scrollLeft = textareaRef.current.scrollLeft;
    }
};

const highlightedCode = useMemo(() => simpleSyntaxHighlight(code), [code]);

const renderTabContent = () => {
    if (!explanation) return null;
    switch(activeTab) {
        case 'summary':
            return <MarkdownRenderer content={explanation.summary} />;
        case 'lineByLine':
            return (
                <div className="space-y-3">
                    {explanation.lineByLine.map((item, index) => (
                        <div key={index} className="p-3 bg-background rounded-md border border-border">
                            <p className="font-mono text-xs text-primary mb-1">Lines: {item.lines}</p>
                            <p className="text-sm">{item.explanation}</p>
                        </div>
                    ))}
                </div>
            );
        case 'complexity':
            return (
                <div>
                    <p><strong>Time Complexity:</strong> <span className="font-mono text-
                    amber-600">{explanation.complexity.time}</span></p>
                    <p><strong>Space Complexity:</strong> <span className="font-mono text-
                    amber-600">{explanation.complexity.space}</span></p>
                </div>
            );
        case 'suggestions':
            return (
                 <ul className="list-disc list-inside space-y-2">
                    {explanation.suggestions.map((item, index) => <li key={index}>{item}</li>)}
                </ul>
            );
    }
}
```

```
return (
    <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
        <header className="mb-6 flex-shrink-0">
            <h1 className="text-3xl font-bold flex items-center">
                <CpuChipIcon />
                <span className="ml-3">AI Code Explainer</span>
            </h1>
            <p className="text-text-secondary mt-1">Get a detailed, structured analysis of
            any code snippet.</p>
        </header>
        <div className="flex-grow grid grid-cols-1 md:grid-cols-2 gap-6 min-h-0">

            {/* Left Column: Code Input */}
            <div className="flex flex-col min-h-0 md:col-span-1">
                <label htmlFor="code-input" className="text-sm font-medium text-text-secondary
                mb-2">Your Code</label>
                <div className="relative flex-grow bg-surface border border-border rounded-md
                focus-within:ring-2 focus-within:ring-primary overflow-hidden">
                    <textarea
                        ref={textareaRef}
                        id="code-input"
                        value={code}
                        onChange={(e) => setCode(e.target.value)}
                        onScroll={handleScroll}
                        placeholder="Paste your code here..."
                        spellCheck="false"
                        className="absolute inset-0 w-full h-full p-4 bg-transparent resize-none font-
                        mono text-sm text-transparent caret-primary outline-none z-10"
                    />
                    <pre
                        ref={preRef}
                        aria-hidden="true"
                        className="absolute inset-0 w-full h-full p-4 font-mono text-sm text-text-
                        primary pointer-events-none z-0 whitespace-pre-wrap overflow-auto no-scrollbar"
                        dangerouslySetInnerHTML={{ __html: highlightedCode + '\n' }}
                    />
                </div>
                <div className="mt-4 flex-shrink-0">
                    <button
                        onClick={() => handleExplain(code)}
                        disabled={isLoading}
                        className="btn-primary w-full flex items-center justify-center px-6 py-3"
                    >
                        {isLoading ? <LoadingSpinner/> : 'Analyze Code'}
                    </button>
                </div>
            </div>

            {/* Right Column: AI Analysis */}
            <div className="flex flex-col min-h-0 md:col-span-1">
                <label className="text-sm font-medium text-text-secondary mb-2">AI
                Analysis</label>
                <div className="relative flex-grow flex flex-col bg-surface border border-border
                rounded-md overflow-hidden">
                    <div className="flex-shrink-0 flex border-b border-border">
                       {(['summary', 'lineByLine', 'complexity', 'suggestions'] as
                       ExplanationTab[]).map(tab => (
                           <button key={tab} onClick={() => setActiveTab(tab)} disabled={!explanation}
                            className={`px-4 py-2 text-sm font-medium capitalize transition-colors
                            ${activeTab === tab ? 'bg-background text-primary font-semibold' : 'text-text-
                            secondary hover:bg-gray-100 disabled:text-gray-400'}`}>
                               {tab.replace(/([A-Z])/g, ' $1')}
```

```
                            </button>
                        ))}
                    </div>
                    <div className="p-4 flex-grow overflow-y-auto">
                        {isLoading && <div className="flex items-center justify-center
                        h-full"><LoadingSpinner /></div>}
                        {error && <p className="text-red-500">{error}</p>}
                        {explanation && !isLoading && renderTabContent()}
                        {!isLoading && !explanation && !error && <div className="text-text-secondary
                        h-full flex items-center justify-center">The analysis will appear here.</div>}
                    </div>
                </div>
            </div>
        </div>
    </div>
    );
};

// ===== AiFeatureBuilder_9.tsx =====

import React, { useState, useCallback, useEffect } from 'react';
import type { GeneratedFile } from '../../types.ts';
import { generateFeature, generateUnitTestsStream, generateCommitMessageStream }
from '../../services/geminiService.ts';
import { saveFile, getAllFiles, clearAllFiles } from
'../../services/dbService.ts';
import { CpuChipIcon, DocumentTextIcon, BeakerIcon, GitBranchIcon } from
'../icons.tsx';
import { LoadingSpinner, MarkdownRenderer } from '../shared/index.tsx';

type ActiveTab = 'CODE' | 'TESTS' | 'COMMIT';

export const AiFeatureBuilder: React.FC = () => {
    const [prompt, setPrompt] = useState<string>('A simple "Hello World" React
    component with a button that shows an alert.');
    const [generatedFiles, setGeneratedFiles] = useState<GeneratedFile[]>([]);
    const [unitTests, setUnitTests] = useState<string>('');
    const [commitMessage, setCommitMessage] = useState<string>('');
    const [selectedFile, setSelectedFile] = useState<GeneratedFile | null>(null);
    const [isLoading, setIsLoading] = useState<boolean>(false);
    const [error, setError] = useState<string>('');
    const [activeTab, setActiveTab] = useState<ActiveTab>('CODE');

    useEffect(() => {
        const loadFiles = async () => {
            const files = await getAllFiles();
            setGeneratedFiles(files);
            if (files.length > 0) setSelectedFile(files[0]);
        };
        loadFiles();
    }, []);

    const handleGenerate = useCallback(async () => {
        if (!prompt.trim()) { setError('Please enter a feature description.'); return; }
        setIsLoading(true);
        setError('');
        await clearAllFiles(); // Start fresh for each generation
        setGeneratedFiles([]);
        setUnitTests('');
        setCommitMessage('');
        setSelectedFile(null);
        setActiveTab('CODE');
```

```
    try {
        const resultFiles = await generateFeature(prompt);
        for (const file of resultFiles) { await saveFile(file); }
        setGeneratedFiles(resultFiles);

        if (resultFiles.length > 0) {
            const componentFile = resultFiles.find(f => f.filePath.endsWith('.tsx') ||
            f.filePath.endsWith('.jsx'));
            setSelectedFile(componentFile || resultFiles[0]);

            if (componentFile) {
                const testStream = generateUnitTestsStream(componentFile.content);
                let tests = '';
                for await (const chunk of testStream) { tests += chunk; setUnitTests(tests); }
            }

            const diffContext = resultFiles.map(f => `File:
            ${f.filePath}\n\n${f.content}`).join('\n---\n');
            const commitStream = generateCommitMessageStream(diffContext);
            let commit = '';
            for await (const chunk of commitStream) { commit += chunk;
            setCommitMessage(commit); }
        }
    } catch (err) {
        const errorMessage = err instanceof Error ? err.message : 'An unknown error
        occurred.';
        setError(`Failed to generate feature: ${errorMessage}`);
    } finally {
        setIsLoading(false);
    }
}, [prompt]);

const renderContent = () => {
    switch (activeTab) {
        case 'TESTS': return <MarkdownRenderer content={unitTests} />;
        case 'COMMIT': return <pre className="w-full h-full p-4 whitespace-pre-wrap
        font-sans text-sm text-text-primary">{commitMessage}</pre>;
        case 'CODE':
        default:
             return selectedFile ? <MarkdownRenderer
             content={`\`\`\`tsx\n${selectedFile.content}\n\`\`\``} /> : <div className="flex
             items-center justify-center h-full text-text-secondary">Select a file to view
             its content.</div>;
    }
}

return (
    <div className="h-full flex flex-col text-text-primary bg-surface">
        <header className="p-4 border-b border-border flex-shrink-0">
            <h1 className="text-xl font-bold flex items-center"><CpuChipIcon /><span
            className="ml-3">AI Feature Builder</span></h1>
        </header>

        <div className="flex-grow flex min-h-0">
            <aside className="w-64 bg-surface border-r border-border p-4 flex flex-col">
                <h2 className="text-sm font-semibold text-text-secondary mb-2">Generated
                Files</h2>
                <div className="overflow-y-auto space-y-1">
                    {generatedFiles.map(file => (
                        <div key={file.filePath} onClick={() => { setSelectedFile(file);
                        setActiveTab('CODE'); }} className={`flex items-center space-x-2 p-2 rounded-md
                        cursor-pointer text-sm ${selectedFile?.filePath === file.filePath && activeTab
```

```
                            === 'CODE' ? 'bg-primary/10 text-primary' : 'hover:bg-gray-100'}`}>
                                <DocumentTextIcon /><span>{file.filePath.split('/').pop()}</span>
                            </div>
                        ))}
                    </div>
                </aside>

                <main className="flex-1 flex flex-col min-w-0">
                    <div className="flex-grow flex flex-col bg-background">
                        <div className="border-b border-border flex items-center bg-surface">
                            <button onClick={() => setActiveTab('CODE')} className={`flex items-center gap-2
                            px-4 py-2 text-sm ${activeTab === 'CODE' ? 'bg-background border-b-2 border-
                            primary text-text-primary' : 'text-text-secondary hover:bg-
                            gray-50'}`}><DocumentTextIcon /> Code</button>
                            {unitTests && <button onClick={() => setActiveTab('TESTS')} className={`flex
                            items-center gap-2 px-4 py-2 text-sm ${activeTab === 'TESTS' ? 'bg-background
                            border-b-2 border-primary text-text-primary' : 'text-text-secondary hover:bg-
                            gray-50'}`}><BeakerIcon /> Tests</button>}
                            {commitMessage && <button onClick={() => setActiveTab('COMMIT')}
                            className={`flex items-center gap-2 px-4 py-2 text-sm ${activeTab === 'COMMIT' ?
                            'bg-background border-b-2 border-primary text-text-primary' : 'text-text-
                            secondary hover:bg-gray-50'}`}><GitBranchIcon /> Commit</button>}
                        </div>
                        <div className="flex-grow p-2 overflow-auto">
                            {isLoading && !generatedFiles.length ? <div className="flex justify-center
                            items-center h-full"><LoadingSpinner/></div> : renderContent()}
                        </div>
                    </div>

                    <div className="flex-shrink-0 p-4 border-t border-border bg-surface">
                        <textarea value={prompt} onChange={(e) => setPrompt(e.target.value)}
                        placeholder="e.g., A user profile card with an avatar, name, and bio."
                        className="w-full p-2 bg-background border border-border rounded-md resize-none
                        text-sm h-20"/>
                         <button onClick={handleGenerate} disabled={isLoading} className="btn-primary
                         mt-2 w-full flex items-center justify-center gap-2 px-4 py-2">
                            {isLoading ? <><LoadingSpinner /> Generating...</> : 'Generate Feature'}
                        </button>
                         {error && <p className="text-red-600 text-xs mt-2 text-center">{error}</p>}
                    </div>
                </main>
            </div>
        </div>
    );
};

// ===== AiCommandCenter_7.tsx =====

import React, { useState, useCallback } from 'react';
import { Type, FunctionDeclaration } from "@google/genai";
import { logError } from '../../services/telemetryService.ts';
import { getInferenceFunction, CommandResponse } from
'../../services/geminiService.ts';
import { FEATURE_TAXONOMY } from '../../services/taxonomyService.ts';
import { useGlobalState } from '../../contexts/GlobalStateContext.tsx';
import { CommandLineIcon } from '../icons.tsx';
import { LoadingSpinner } from '../shared/index.tsx';
import { ALL_FEATURES } from './index.ts';

const functionDeclarations: FunctionDeclaration[] = [
    {
        name: 'navigateTo',
```

```
        description: 'Navigates to a specific feature page.',
        parameters: {
            type: Type.OBJECT,
            properties: {
                featureId: {
                    type: Type.STRING,
                    description: 'The ID of the feature to navigate to.',
                    enum: ALL_FEATURES.map(f => f.id)
                },
            },
            required: ['featureId'],
        },
    },
    {
        name: 'runFeatureWithInput',
        description: 'Navigates to a feature and passes initial data to it.',
        parameters: {
            type: Type.OBJECT,
            properties: {
                 featureId: {
                    type: Type.STRING,
                    description: 'The ID of the feature to run.',
                    enum: ALL_FEATURES.map(f => f.id)
                },
                props: {
                    type: Type.OBJECT,
                    description: 'An object containing the initial properties for the feature, based
                    on its required inputs.',
                    properties: {
                        initialCode: { type: Type.STRING },
                        initialPrompt: { type: Type.STRING },
                        beforeCode: { type: Type.STRING },
                        afterCode: { type: Type.STRING },
                        logInput: { type: Type.STRING },
                        diff: { type: Type.STRING },
                        codeInput: { type: Type.STRING },
                        jsonInput: { type: Type.STRING },
                    }
                }
            },
            required: ['featureId', 'props']
        }
    }
];

const knowledgeBase = FEATURE_TAXONOMY.map(f => `- ${f.name} (${f.id}):
${f.description} Inputs: ${f.inputs}`).join('\n');

const ExamplePromptButton: React.FC<{ text: string, onClick: (text: string) =>
void }> = ({ text, onClick }) => (
    <button
        onClick={() => onClick(text)}
        className="px-3 py-1.5 bg-surface border border-border rounded-full text-xs
        hover:bg-gray-100 transition-colors"
    >
        {text}
    </button>
)

export const AiCommandCenter: React.FC = () => {
    const { dispatch } = useGlobalState();
    const [prompt, setPrompt] = useState('');
```

```
const [isLoading, setIsLoading] = useState(false);
const [lastResponse, setLastResponse] = useState('');

const handleCommand = useCallback(async () => {
    if (!prompt.trim()) return;

    setIsLoading(true);
    setLastResponse('');

    try {
        const response: CommandResponse = await getInferenceFunction(prompt,
        functionDeclarations, knowledgeBase);

        if (response.functionCalls && response.functionCalls.length > 0) {
            const call = response.functionCalls[0];
            const { name, args } = call;

            setLastResponse(`Understood! Executing command: ${name}`);

            switch (name) {
                case 'navigateTo':
                    dispatch({ type: 'SET_VIEW', payload: { view: args.featureId }});
                    break;
                case 'runFeatureWithInput':
                     dispatch({ type: 'SET_VIEW', payload: { view: args.featureId, props: args.props
                     } });
                    break;
                default:
                    setLastResponse(`Unknown command: ${name}`);
            }
             setPrompt('');
        } else {
             setLastResponse(response.text);
        }

    } catch (err) {
        logError(err as Error, { prompt });
        setLastResponse(err instanceof Error ? err.message : 'An unknown error
        occurred.');
    } finally {
        setIsLoading(false);
    }
}, [prompt, dispatch]);

const handleKeyDown = (e: React.KeyboardEvent) => {
    if (e.key === 'Enter' && !e.shiftKey) {
        e.preventDefault();
        handleCommand();
    }
};

const handleExampleClick = (text: string) => {
    setPrompt(text);
}

return (
    <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
        <header className="mb-6 text-center">
            <h1 className="text-4xl font-extrabold tracking-tight flex items-center justify-
            center">
                <CommandLineIcon />
                <span className="ml-3">AI Command Center</span>
```

```
                </h1>
                <p className="mt-2 text-lg text-text-secondary">What would you like to do?</p>
            </header>

            <div className="flex-grow flex flex-col justify-end max-w-3xl w-full mx-auto">
                {lastResponse && (
                    <div className="mb-4 p-4 bg-surface rounded-lg text-text-primary border border-
                    border">
                        <p><strong>AI:</strong> {lastResponse}</p>
                    </div>
                )}
                 <div className="relative">
                    <textarea
                        value={prompt}
                        onChange={e => setPrompt(e.target.value)}
                        onKeyDown={handleKeyDown}
                        disabled={isLoading}
                        placeholder='Try "explain this code: const a = 1;" or "open the theme designer"'
                        className="w-full p-4 pr-28 rounded-lg bg-surface border border-border
                        focus:ring-2 focus:ring-primary focus:outline-none resize-none shadow-sm"
                        rows={2}
                    />
                    <button
                        onClick={handleCommand}
                        disabled={isLoading}
                        className="btn-primary absolute right-3 top-1/2 -translate-y-1/2 px-4 py-2"
                    >
                       {isLoading ? <LoadingSpinner/> : 'Send'}
                    </button>
                </div>
                 <div className="flex flex-wrap items-center justify-center gap-2 mt-4">
                    <ExamplePromptButton text="Open Theme Designer" onClick={handleExampleClick} />
                    <ExamplePromptButton text="Generate a commit for a bug fix"
                    onClick={handleExampleClick} />
                    <ExamplePromptButton text="Create a regex for email validation"
                    onClick={handleExampleClick} />
                </div>
                 <p className="text-xs text-text-secondary text-center mt-2">Press Enter to send,
                 Shift+Enter for new line.</p>
            </div>
        </div>
    );
};

// ===== PrSummaryGenerator_9.tsx =====



// ===== ProjectExplorer_7.tsx =====

import React, { useState, useEffect, useCallback } from 'react';
import { useGlobalState } from '../../contexts/GlobalStateContext.tsx';
import { useVaultModal } from '../../contexts/VaultModalContext.tsx';
import * as vaultService from '../../services/vaultService.ts';
import { initializeOctokit } from '../../services/authService.ts';
import { getRepos, getRepoTree, getFileContent } from
'../../services/githubService.ts';
import type { Repo, FileNode } from '../../types.ts';
import { FolderIcon, DocumentIcon } from '../icons.tsx';
import { LoadingSpinner } from '../shared/index.tsx';

const FileTree: React.FC<{ node: FileNode, onFileSelect: (path: string) => void
```

```
}> = ({ node, onFileSelect }) => {
    const [isOpen, setIsOpen] = useState(true);

    if (node.type === 'file') {
        return (
            <div
                className="flex items-center space-x-2 pl-4 py-1 cursor-pointer hover:bg-
                gray-100 rounded"
                onClick={() => onFileSelect(node.path)}
            >
                <DocumentIcon />
                <span>{node.name}</span>
            </div>
        );
    }

    return (
        <div>
            <div
                className="flex items-center space-x-2 py-1 cursor-pointer hover:bg-gray-100
                rounded"
                onClick={() => setIsOpen(!isOpen)}
            >
                <div className={`transform transition-transform ${isOpen ? 'rotate-90' :
                ''}`}>■</div>
                <FolderIcon />
                <span className="font-semibold">{node.name}</span>
            </div>
            {isOpen && node.children && (
                <div className="pl-4 border-l border-border ml-3">
                    {node.children.map(child => <FileTree key={child.path} node={child}
                    onFileSelect={onFileSelect} />)}
                </div>
            )}
        </div>
    );
};

export const ProjectExplorer: React.FC = () => {
    const { state, dispatch } = useGlobalState();
    const { vaultState, connections, selectedRepo, projectFiles } = state;
    const { requestUnlock } = useVaultModal();
    const [repos, setRepos] = useState<Repo[]>([]);
    const [isLoading, setIsLoading] = useState<'repos' | 'tree' | 'file' |
    null>(null);
    const [error, setError] = useState('');
    const [activeFileContent, setActiveFileContent] = useState<string | null>(null);

    const getApiClient = useCallback(async () => {
        if (!vaultState.isUnlocked) {
            const unlocked = await requestUnlock();
            if (!unlocked) throw new Error("Vault must be unlocked to access GitHub.");
        }
        const token = await vaultService.getDecryptedCredential('github_pat');
        if (!token) throw new Error("GitHub token not found in vault.");
        return initializeOctokit(token);
    }, [vaultState.isUnlocked, requestUnlock]);


    useEffect(() => {
        const loadRepos = async () => {
            if (connections.github) {
```

```
            setIsLoading('repos');
            setError('');
            try {
                const octokit = await getApiClient();
                const userRepos = await getRepos(octokit);
                setRepos(userRepos);
            } catch (err) {
                setError(err instanceof Error ? err.message : 'Failed to load repositories');
            } finally {
                setIsLoading(null);
            }
        }
    };
    loadRepos();
}, [connections.github, getApiClient]);

useEffect(() => {
    const loadTree = async () => {
         if (selectedRepo && connections.github) {
            setIsLoading('tree');
            setError('');
            setActiveFileContent(null);
            try {
                const octokit = await getApiClient();
                const tree = await getRepoTree(octokit, selectedRepo.owner, selectedRepo.repo);
                dispatch({ type: 'LOAD_PROJECT_FILES', payload: tree });
            } catch (err) {
                 setError(err instanceof Error ? err.message : 'Failed to load repository tree');
            } finally {
                setIsLoading(null);
            }
        }
    };
    loadTree();
}, [selectedRepo, connections.github, dispatch, getApiClient]);

const handleFileSelect = async (path: string) => {
    if (!selectedRepo) return;
    setIsLoading('file');
    try {
        const octokit = await getApiClient();
        const content = await getFileContent(octokit, selectedRepo.owner,
        selectedRepo.repo, path);
        setActiveFileContent(content);
    } catch (err) {
        setError((err as Error).message);
    } finally {
        setIsLoading(null);
    }
};

if (!connections.github) {
    return (
        <div className="h-full flex flex-col items-center justify-center text-center
        text-text-secondary p-4">
            <FolderIcon />
            <h2 className="text-lg font-semibold mt-2">Connect to GitHub</h2>
            <p>Please go to the "Security Vault" tab and provide a Personal Access Token to
            explore your repositories.</p>
        </div>
    );
}
```

```
    return (
        <div className="h-full flex flex-col text-text-primary">
            <header className="p-4 border-b border-border flex-shrink-0">
                <h1 className="text-xl font-bold flex items-center"><FolderIcon /><span
                className="ml-3">Project Explorer</span></h1>
                <div className="mt-2">
                    <select
                        value={selectedRepo ? `${selectedRepo.owner}/${selectedRepo.repo}` : ''}
                        onChange={e => {
                            const [owner, repo] = e.target.value.split('/');
                            dispatch({ type: 'SET_SELECTED_REPO', payload: { owner, repo } });
                        }}
                        className="w-full p-2 bg-surface border border-border rounded-md text-sm"
                    >
                        <option value="" disabled>{isLoading === 'repos' ? 'Loading...' : 'Select a
                        repository'}</option>
                        {repos.map(r => <option key={r.id} value={r.full_name}>{r.full_name}</option>)}
                    </select>
                </div>
                {error && <p className="text-red-500 text-xs mt-2">{error}</p>}
            </header>
            <div className="flex-grow flex min-h-0">
                <aside className="w-1/3 bg-background border-r border-border p-4 overflow-y-
                auto">
                    {isLoading === 'tree' && <div className="flex justify-center"><LoadingSpinner
                    /></div>}
                    {projectFiles && <FileTree node={projectFiles} onFileSelect={handleFileSelect}
                    />}
                </aside>
                <main className="flex-1 bg-surface">
                     {isLoading === 'file' ? <div className="flex items-center justify-center
                     h-full"><LoadingSpinner /></div> :
                        <pre className="w-full h-full p-4 text-sm overflow-auto whitespace-pre-wrap">
                            <code>{activeFileContent ?? 'Select a file to view its content.'}</code>
                        </pre>
                    }
                </main>
            </div>
        </div>
    );
};


// ===== AiCommitGenerator_7.tsx =====

import React, { useState, useCallback, useEffect } from 'react';
import { generateCommitMessageStream } from '../../services/geminiService.ts';
import { downloadFile } from '../../services/fileUtils.ts';
import { GitBranchIcon, ArrowDownTrayIcon } from '../icons.tsx';
import { LoadingSpinner } from '../shared/index.tsx';

const exampleDiff = `diff --git a/src/components/Button.tsx
b/src/components/Button.tsx
index 1b2c3d4..5e6f7g8 100644
--- a/src/components/Button.tsx
+++ b/src/components/Button.tsx
@@ -1,7 +1,7 @@
 import React from 'react';

 interface ButtonProps {
-  text: string;
+  label: string;
```

```
   onClick: () => void;
 }
`;

export const AiCommitGenerator: React.FC<{ diff?: string }> = ({ diff:
initialDiff }) => {
    const [diff, setDiff] = useState<string>(initialDiff || exampleDiff);
    const [message, setMessage] = useState<string>('');
    const [isLoading, setIsLoading] = useState<boolean>(false);
    const [error, setError] = useState<string>('');

    const handleGenerate = useCallback(async (diffToAnalyze: string) => {
        if (!diffToAnalyze.trim()) {
            setError('Please paste a diff to generate a message.');
            return;
        }
        setIsLoading(true);
        setError('');
        setMessage('');
        try {
            const stream = generateCommitMessageStream(diffToAnalyze);
            let fullResponse = '';
            for await (const chunk of stream) {
                fullResponse += chunk;
                setMessage(fullResponse);
            }
        } catch (err) {
            const errorMessage = err instanceof Error ? err.message : 'An unknown error
            occurred.';
            setError(`Failed to generate message: ${errorMessage}`);
        } finally {
            setIsLoading(false);
        }
    }, []);

    useEffect(() => {
        if (initialDiff) {
            setDiff(initialDiff);
            handleGenerate(initialDiff);
        }
    }, [initialDiff, handleGenerate]);

    const handleCopy = () => {
        navigator.clipboard.writeText(message);
    };

    const handleDownload = () => {
        downloadFile(message, 'commit_message.txt', 'text/plain');
    };

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
            <header className="mb-6">
                <h1 className="text-3xl font-bold flex items-center">
                    <GitBranchIcon />
                    <span className="ml-3">AI Commit Message Generator</span>
                </h1>
                <p className="text-text-secondary mt-1">Paste your diff and let Gemini craft the
                perfect commit message.</p>
            </header>
            <div className="flex-grow flex flex-col gap-4 min-h-0">
                <div className="flex flex-col flex-1 min-h-0">
```

```
                    <label htmlFor="diff-input" className="text-sm font-medium text-text-secondary
                    mb-2">Git Diff</label>
                    <textarea
                        id="diff-input"
                        value={diff}
                        onChange={(e) => setDiff(e.target.value)}
                        placeholder="Paste your git diff here..."
                        className="flex-grow p-4 bg-surface border border-border rounded-md resize-none
                        font-mono text-sm text-text-primary focus:ring-2 focus:ring-primary
                        focus:outline-none"
                    />
                </div>
                <div className="flex-shrink-0">
                    <button
                        onClick={() => handleGenerate(diff)}
                        disabled={isLoading}
                        className="btn-primary w-full max-w-xs mx-auto flex items-center justify-center
                        px-6 py-3"
                    >
                        {isLoading ? <LoadingSpinner /> : 'Generate Commit Message'}
                    </button>
                </div>
                <div className="flex flex-col flex-1 min-h-0">
                    <div className="flex justify-between items-center mb-2">
                        <label className="text-sm font-medium text-text-secondary">Generated
                        Message</label>
                        {message && !isLoading && (
                            <div className="flex items-center gap-2">
                                <button onClick={handleCopy} className="px-3 py-1 bg-gray-100 text-xs rounded-md
                                hover:bg-gray-200">Copy</button>
                                <button onClick={handleDownload} className="flex items-center gap-1 px-3 py-1
                                bg-gray-100 text-xs rounded-md hover:bg-gray-200">
                                    <ArrowDownTrayIcon className="w-4 h-4" /> Download
                                </button>
                            </div>
                        )}
                    </div>
                    <div className="relative flex-grow p-4 bg-surface border border-border rounded-
                    md overflow-y-auto">
                        {isLoading && (
                             <div className="flex items-center justify-center h-full">
                                <LoadingSpinner />
                             </div>
                        )}
                        {error && <p className="text-red-500">{error}</p>}
                        {message && !isLoading && (
                           <pre className="whitespace-pre-wrap font-sans text-text-primary">{message}</pre>
                        )}
                         {!isLoading && !message && !error && (
                            <div className="text-text-secondary h-full flex items-center justify-center">
                                The commit message will appear here.
                            </div>
                        )}
                    </div>
                </div>
            </div>
        </div>
    );
};

// ===== AiImageGenerator_7.tsx =====
import React, { useState, useCallback, useRef } from 'react';
```

```
import { generateImage, generateImageFromImageAndText } from
'../../services/geminiService.ts';
import { fileToBase64, blobToDataURL } from '../../services/fileUtils.ts';
import { ImageGeneratorIcon, SparklesIcon, ArrowDownTrayIcon, XMarkIcon } from
'../icons.tsx';
import { LoadingSpinner } from '../shared/index.tsx';

const surprisePrompts = [
    'A majestic lion wearing a crown, painted in the style of Van Gogh.',
    'A futuristic cityscape on another planet with two moons in the sky.',
    'A cozy, magical library inside a giant tree.',
    'A surreal image of a ship sailing on a sea of clouds.',
    'An astronaut riding a space-themed bicycle on the moon.',
];

interface UploadedImage {
    base64: string;
    dataUrl: string;
    mimeType: string;
}

export const AiImageGenerator: React.FC = () => {
    const [prompt, setPrompt] = useState<string>('A photorealistic image of a
    futuristic city at sunset, with flying cars.');
    const [uploadedImage, setUploadedImage] = useState<UploadedImage | null>(null);
    const [generatedImageUrl, setGeneratedImageUrl] = useState<string | null>(null);
    const [isLoading, setIsLoading] = useState<boolean>(false);
    const [error, setError] = useState<string>('');
    const fileInputRef = useRef<HTMLInputElement>(null);

    const handleGenerate = useCallback(async () => {
        if (!prompt.trim()) {
            setError('Please enter a prompt to generate an image.');
            return;
        }
        setIsLoading(true);
        setError('');
        setGeneratedImageUrl(null);
        try {
            let resultUrl: string;
            if (uploadedImage) {
                resultUrl = await generateImageFromImageAndText(prompt, uploadedImage.base64,
                uploadedImage.mimeType);
            } else {
                resultUrl = await generateImage(prompt);
            }
            setGeneratedImageUrl(resultUrl);
        } catch (err) {
            const errorMessage = err instanceof Error ? err.message : 'An unknown error
            occurred.';
            setError(`Failed to generate image: ${errorMessage}`);
        } finally {
            setIsLoading(false);
        }
    }, [prompt, uploadedImage]);

    const handleSurpriseMe = () => {
        const randomPrompt = surprisePrompts[Math.floor(Math.random() *
        surprisePrompts.length)];
        setPrompt(randomPrompt);
    };
    const processImageBlob = async (blob: Blob) => {
```

```
    try {
        const [dataUrl, base64] = await Promise.all([
            blobToDataURL(blob),
            fileToBase64(blob as File)
        ]);
        setUploadedImage({ dataUrl, base64, mimeType: blob.type });
    } catch (e) {
        setError('Could not process the image.');
    }
};

const handlePaste = useCallback(async (event: React.ClipboardEvent) => {
    const items = event.clipboardData.items;
    for (const item of items) {
        if (item.type.indexOf('image') !== -1) {
            const blob = item.getAsFile();
            if (blob) {
                await processImageBlob(blob);
                return;
            }
        }
    }
}, []);

const handleFileChange = async (event: React.ChangeEvent<HTMLInputElement>) => {
    const file = event.target.files?.[0];
    if (file) {
        await processImageBlob(file);
    }
};

const handleDownload = () => {
    if (!generatedImageUrl) return;
    const link = document.createElement('a');
    link.href = generatedImageUrl;
    link.download = `${prompt.slice(0, 30).replace(/\s/g, '_')}.png`;
    document.body.appendChild(link);
    link.click();
    document.body.removeChild(link);
}

return (
    <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
        <header className="mb-6">
            <h1 className="text-3xl font-bold flex items-center">
                <ImageGeneratorIcon />
                <span className="ml-3">AI Image Generator</span>
            </h1>
            <p className="text-text-secondary mt-1">Generate images from text, or provide an
            image for inspiration.</p>
        </header>

        <div className="flex-grow grid grid-cols-1 lg:grid-cols-2 gap-6 min-h-0">
            {/* Left Column: Inputs */}
            <div className="flex flex-col gap-4">
                <div>
                    <label htmlFor="prompt-input" className="text-sm font-medium text-text-
                    secondary">Your Prompt</label>
                    <textarea
                        id="prompt-input"
                        value={prompt}
                        onChange={(e) => setPrompt(e.target.value)}
```

```
            placeholder="e.g., A cute cat wearing a wizard hat"
            className="w-full p-3 mt-1 rounded-md bg-surface border border-border
            focus:ring-2 focus:ring-primary focus:outline-none resize-y"
            rows={3}
        />
    </div>

    <div className="flex flex-col flex-grow min-h-[200px]">
         <label className="text-sm font-medium text-text-secondary mb-1">Inspiration
         Image (Optional)</label>
         <div onPaste={handlePaste} className="relative flex-grow flex flex-col items-
         center justify-center bg-surface p-4 rounded-lg border-2 border-dashed border-
         border focus:outline-none focus:border-primary" tabIndex={0}>
            {uploadedImage ? (
                <>
                    <img src={uploadedImage.dataUrl} alt="Uploaded content" className="max-w-full
                    max-h-full object-contain rounded-md shadow-lg" />
                    <button onClick={() => setUploadedImage(null)} className="absolute top-2 right-2
                    p-1 bg-black/30 text-white rounded-full hover:bg-black/50"><XMarkIcon
                    /></button>
                </>
            ) : (
                <div className="text-center text-text-secondary">
                    <h2 className="text-lg font-bold text-text-primary">Paste an image here</h2>
                    <p className="text-sm">(Cmd/Ctrl + V)</p>
                    <p className="text-xs my-1">or</p>
                    <button onClick={() => fileInputRef.current?.click()} className="text-sm font-
                    semibold text-primary hover:underline">Upload File</button>
                    <input type="file" ref={fileInputRef} onChange={handleFileChange}
                    accept="image/*" className="hidden"/>
                </div>
            )}
         </div>
    </div>

    <div className="flex gap-2">
        <button
            onClick={handleGenerate}
            disabled={isLoading}
            className="btn-primary w-full flex items-center justify-center px-6 py-3"
        >
            {isLoading ? <LoadingSpinner /> : 'Generate Image'}
        </button>
        <button
            onClick={handleSurpriseMe}
            disabled={isLoading}
            className="px-4 py-3 bg-surface border border-border rounded-md hover:bg-
            gray-100 transition-colors"
            title="Surprise Me!"
        >
            <SparklesIcon />
        </button>
    </div>
</div>

{/* Right Column: Output */}
<div className="flex flex-col h-full">
    <label className="text-sm font-medium text-text-secondary mb-2">Generated
    Image</label>
    <div className="flex-grow flex items-center justify-center bg-background
    border-2 border-dashed border-border rounded-lg p-4 relative overflow-auto">
        {isLoading && <LoadingSpinner />}
```

```
                        {error && <p className="text-red-500 text-center">{error}</p>}
                        {generatedImageUrl && !isLoading && (
                            <>
                                <img src={generatedImageUrl} alt={prompt || "Generated by AI"} className="max-w-
                                full max-h-full object-contain rounded-md shadow-lg" />
                                <button
                                  onClick={handleDownload}
                                  className="absolute top-4 right-4 p-2 bg-black/30 text-white rounded-full
                                  hover:bg-black/50 backdrop-blur-sm"
                                  title="Download Image"
                                >
                                    <ArrowDownTrayIcon />
                                </button>
                            </>
                        )}
                        {!isLoading && !generatedImageUrl && !error && (
                            <div className="text-center text-text-secondary">
                                <p>Your generated image will appear here.</p>
                            </div>
                        )}
                    </div>
                </div>
            </div>
        </div>
    );
};

// ===== Connections_7.tsx =====

import React, { useState, useEffect } from 'react';
import { GithubIcon } from '../icons.tsx';
import { useGlobalState } from '../../contexts/GlobalStateContext.tsx';
import * as vaultService from '../../services/vaultService.ts';
import { validateToken } from '../../services/authService.ts';
import { useVaultModal } from '../../contexts/VaultModalContext.tsx';
import { LoadingSpinner } from '../shared/LoadingSpinner.tsx';
import type { User } from '../../types.ts';

const GitHubConnection: React.FC = () => {
    const { state, dispatch } = useGlobalState();
    const { github: githubUser } = state.connections;
    const { isInitialized, isUnlocked } = state.vaultState;
    const { requestUnlock, requestCreation } = useVaultModal();
    const [status, setStatus] = useState<'idle' | 'loading' | 'success' |
    'error'>('idle');
    const [tokenInput, setTokenInput] = useState('');
    const [error, setError] = useState('');

    // On load, check if a credential exists and try to validate it.
    useEffect(() => {
        const checkConnection = async () => {
            if (!isUnlocked) return;
            setStatus('loading');
            try {
                const token = await vaultService.getDecryptedCredential('github_pat');
                if (token) {
                    const user = await validateToken(token);
                    dispatch({ type: 'SET_GITHUB_CONNECTION', payload: user });
                    setStatus('success');
                } else {
                    dispatch({ type: 'SET_GITHUB_CONNECTION', payload: null });
                    setStatus('idle');
```

```
            }
        } catch (err) {
            setError('Failed to validate stored token.');
            setStatus('error');
        }
    };
    checkConnection();
}, [isUnlocked, dispatch]);

const handleConnect = async () => {
    if (!tokenInput.trim()) {
        setError('Token cannot be empty.');
        return;
    }

    let vaultReady = isUnlocked;
    if (!isInitialized) {
        vaultReady = await requestCreation();
    } else if (!isUnlocked) {
        vaultReady = await requestUnlock();
    }

    if (!vaultReady) return;

    setStatus('loading');
    setError('');
    try {
        const user: User = await validateToken(tokenInput);
        await vaultService.saveCredential('github_pat', tokenInput);
        dispatch({ type: 'SET_GITHUB_CONNECTION', payload: user });
        setStatus('success');
        setTokenInput('');
    } catch (err) {
        setError(err instanceof Error ? `Invalid Token: ${err.message}` : 'Could not
        connect.');
        setStatus('error');
    }
};

const handleDisconnect = async () => {
    // Here we'd need a flow to remove the credential from the vault,
    // which requires unlocking it. For now, we'll just log out from the session.
    dispatch({ type: 'LOGOUT_GITHUB' });
    setStatus('idle');
    alert("Disconnected from GitHub for this session. To permanently remove the
    credential, reset the vault in Settings.");
};

return (
    <div className="bg-surface border border-border rounded-lg p-6">
        <div className="flex items-center justify-between">
            <div className="flex items-center gap-4">
                <div className="w-10 h-10"><GithubIcon /></div>
                <div>
                    <h3 className="text-lg font-bold text-text-primary">GitHub</h3>
                    {githubUser ? (
                         <p className="text-sm text-green-600">Connected as {githubUser.login}</p>
                    ) : (
                         <p className="text-sm text-text-secondary">Not Connected</p>
                    )}
                </div>
            </div>
```

```
                {githubUser && (
                   <button onClick={handleDisconnect} className="px-4 py-2 bg-red-500/10 text-
                   red-600 font-semibold rounded-lg hover:bg-red-500/20">
                       Disconnect
                   </button>
                )}
            </div>

            {!githubUser && (
                <div className="mt-4 pt-4 border-t border-border">
                   <label htmlFor="github-pat" className="block text-sm font-medium text-text-
                   secondary mb-1">Personal Access Token (Classic)</label>
                   <div className="flex gap-2">
                       <input
                           id="github-pat"
                           type="password"
                           value={tokenInput}
                           onChange={(e) => setTokenInput(e.target.value)}
                           placeholder="ghp_..."
                           className="flex-grow p-2 bg-background border border-border rounded-md text-sm"
                       />
                        <button onClick={handleConnect} disabled={status === 'loading'} className="btn-
                        primary px-6 py-2 flex items-center justify-center min-w-[100px]">
                           {status === 'loading' ? <LoadingSpinner /> : 'Connect'}
                       </button>
                   </div>
                   {error && <p className="text-red-500 text-xs mt-2">{error}</p>}
                   <p className="text-xs text-text-secondary mt-2">
                       Your token is encrypted and stored only in your browser's local database.
                       Required scopes: `repo`, `read:user`.
                   </p>
                </div>
            )}
        </div>
    );
};

export const Connections: React.FC = () => {
    return (
        <div className="h-full p-4 sm:p-6 lg:p-8">
            <div className="max-w-4xl mx-auto">
                <header className="mb-8">
                   <h1 className="text-4xl font-extrabold tracking-tight">Security Vault</h1>
                   <p className="mt-2 text-lg text-text-secondary">Manage your encrypted API
                   credentials.</p>
                </header>
                <div className="space-y-6">
                   <GitHubConnection />
                </div>
            </div>
        </div>
    );
};


// ===== ThemeDesigner_12.tsx =====

import React, { useState, useCallback, useEffect } from 'react';
import { SparklesIcon, ArrowDownTrayIcon, PhotoIcon } from '../icons.tsx';
import { generateSemanticTheme } from '../../services/geminiService.ts';
import { fileToBase64 } from '../../services/fileUtils.ts';
import type { SemanticColorTheme } from '../../types.ts';
```

```
import { LoadingSpinner } from '../shared/index.tsx';

const ColorDisplay: React.FC<{ name: string; color: { name: string; value:
string; } }> = ({ name, color }) => (
    <div className="flex items-center justify-between p-2 bg-background rounded-md
    border border-border">
        <div className="flex items-center gap-3">
            <div className="w-6 h-6 rounded-full border border-border" style={{
            backgroundColor: color.value }} />
            <div>
                <p className="text-sm font-semibold text-text-primary capitalize">{name}</p>
                <p className="text-xs text-text-secondary">{color.name}</p>
            </div>
        </div>
        <span className="font-mono text-sm text-text-secondary">{color.value}</span>
    </div>
);

const AccessibilityCheck: React.FC<{ name: string, check: { ratio: number;
score: string; } }> = ({ name, check }) => {
    const scoreColor = check.score === 'AAA' ? 'text-green-600' : check.score ===
    'AA' ? 'text-emerald-600' : 'text-red-600';
    return (
        <div className="flex items-center justify-between p-2 bg-background rounded-md
        border border-border text-sm">
            <p className="text-text-secondary">{name}</p>
            <div className="flex items-center gap-2">
                <span className="font-mono">{check.ratio.toFixed(2)}</span>
                <span className={`font-bold px-2 py-0.5 rounded-full text-xs ${scoreColor}
                ${scoreColor.replace('text-', 'bg-')}/10`}>{check.score}</span>
            </div>
        </div>
    );
}

export const ThemeDesigner: React.FC = () => {
    const [theme, setTheme] = useState<SemanticColorTheme | null>(null);
    const [prompt, setPrompt] = useState('A calming, minimalist theme for a blog');
    const [image, setImage] = useState<{ base64: string, name: string } |
    null>(null);
    const [isLoading, setIsLoading] = useState(false);
    const [error, setError] = useState('');

    const handleGenerate = useCallback(async () => {
        const textPart = { text: `Generate a theme based on this description:
        "${prompt}"` };
        const imagePart = image ? { inlineData: { mimeType: 'image/png', data:
        image.base64 } } : null;
        const parts = imagePart ? [textPart, imagePart] : [textPart];

        setIsLoading(true); setError('');
        try {
            const newTheme = await generateSemanticTheme({ parts });
            setTheme(newTheme);
        } catch (err) {
            setError(err instanceof Error ? err.message : "An unknown error occurred.");
        } finally {
            setIsLoading(false);
        }
    }, [prompt, image]);

    const handleFileChange = async (e: React.ChangeEvent<HTMLInputElement>) => {
```

```
    const file = e.target.files?.[0];
    if (file) {
        const base64 = await fileToBase64(file);
        setImage({ base64, name: file.name });
        setPrompt(`A theme based on the uploaded image: ${file.name}`);
    }
};

useEffect(() => { handleGenerate(); }, []);

return (
    <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
        <header className="mb-6">
            <h1 className="text-3xl font-bold flex items-center"><SparklesIcon /><span
            className="ml-3">AI Theme Designer</span></h1>
            <p className="text-text-secondary mt-1">Generate a full design system from a
            description or image.</p>
        </header>
        <div className="flex-grow grid grid-cols-1 md:grid-cols-2 gap-6 min-h-0">
            <div className="md:col-span-1 flex flex-col gap-4 bg-surface border border-
            border p-6 rounded-lg overflow-y-auto">
                <h3 className="text-xl font-bold">Describe or Upload</h3>
                <textarea value={prompt} onChange={e => setPrompt(e.target.value)}
                className="p-2 bg-background border border-border rounded-md resize-none text-sm
                h-24" placeholder="e.g., A light, airy theme for a blog" />
                 <div className="relative border border-dashed border-border rounded-lg p-4 text-
                 center">
                    <input type="file" onChange={handleFileChange} className="absolute inset-0
                    w-full h-full opacity-0 cursor-pointer" />
                    <PhotoIcon/>
                    <p className="text-sm mt-1">{image ? `Image: ${image.name}` : 'Upload an image
                    (optional)'}</p>
                </div>
                <button onClick={handleGenerate} disabled={isLoading} className="btn-primary
                w-full flex items-center justify-center gap-2 px-4 py-2">
                    {isLoading ? <LoadingSpinner /> : 'Generate Theme'}
                </button>
                {error && <p className="text-red-500 text-xs text-center">{error}</p>}

                {theme && !isLoading && (
                    <div className="mt-4 border-t border-border pt-4 space-y-4">
                        <div><h3 className="text-lg font-bold mb-2">Palette</h3><div
                        className="space-y-2"><ColorDisplay name="Primary"
                        color={theme.palette.primary}/><ColorDisplay name="Secondary"
                        color={theme.palette.secondary}/><ColorDisplay name="Accent"
                        color={theme.palette.accent}/><ColorDisplay name="Neutral"
                        color={theme.palette.neutral}/></div></div>
                        <div><h3 className="text-lg font-bold mb-2">Theme Roles</h3><div
                        className="space-y-2"><ColorDisplay name="Background"
                        color={theme.theme.background}/><ColorDisplay name="Surface"
                        color={theme.theme.surface}/><ColorDisplay name="Text Primary"
                        color={theme.theme.textPrimary}/><ColorDisplay name="Text Secondary"
                        color={theme.theme.textSecondary}/></div></div>
                        <div><h3 className="text-lg font-bold mb-2">Accessibility (WCAG 2.1)</h3><div
                        className="space-y-2"><AccessibilityCheck name="Primary on Surface"
                        check={theme.accessibility.primaryOnSurface}/><AccessibilityCheck name="Text on
                        Surface" check={theme.accessibility.textPrimaryOnSurface}/><AccessibilityCheck
                        name="Subtle Text on Surface"
                        check={theme.accessibility.textSecondaryOnSurface}/></div></div>
                    </div>
                )}
            </div>
```

```
                <div className="md:col-span-1 rounded-lg p-8 overflow-y-auto border border-
                border" style={{ backgroundColor: theme?.theme.background.value, color:
                theme?.theme.textPrimary.value }}>
                     <h3 className="text-2xl font-bold mb-6">Live Preview</h3>
                     {theme ? (
                         <div className="p-6 rounded-lg grid grid-cols-1 md:grid-cols-2 gap-6" style={{
                         backgroundColor: theme.theme.surface.value }}>
                            <div className="space-y-4">
                                <h4 className="text-lg font-bold">Sample Card</h4>
                                <p className="text-sm" style={{color: theme.theme.textSecondary.value}}>This is
                                a sample card to demonstrate the theme colors. It contains a primary button and
                                some secondary text.</p>
                                <button className="px-4 py-2 rounded-md font-bold transition-colors" style={{
                                backgroundColor: theme.palette.primary.value, color: theme.theme.surface.value
                                }}>Primary Button</button>
                            </div>
                             <div className="space-y-4">
                                <input type="text" placeholder="Text input" className="w-full px-3 py-2 rounded-
                                md border" style={{backgroundColor: theme.theme.background.value, borderColor:
                                theme.palette.primary.value, color: theme.theme.textPrimary.value}} />
                                <div className="p-3 border rounded" style={{borderColor:
                                theme.palette.primary.value, color: theme.theme.textSecondary.value}}>
                                    <p>A bordered container.</p>
                                </div>
                             </div>
                         </div>
                     ) : <div className="flex items-center justify-center h-full text-text-
                     secondary">Theme preview will appear here.</div>}
                </div>
            </div>
        </div>
    );
};

// ===== AiCodeExplainer_10.tsx =====

import React, { useState, useCallback, useEffect, useMemo, useRef } from
'react';
import mermaid from 'mermaid';
import { explainCodeStructured, generateMermaidJs } from
'../../services/geminiService.ts';
import type { StructuredExplanation } from '../../types.ts';
import { CpuChipIcon } from '../icons.tsx';
import { MarkdownRenderer, LoadingSpinner } from '../shared/index.tsx';

const exampleCode = `const bubbleSort = (arr) => {
  for (let i = 0; i < arr.length; i++) {
    for (let j = 0; j < arr.length - i - 1; j++) {
      if (arr[j] > arr[j + 1]) {
        [arr[j], arr[j + 1]] = [arr[j + 1], arr[j]];
      }
    }
  }
  return arr;
};`;

type ExplanationTab = 'summary' | 'lineByLine' | 'complexity' | 'suggestions' |
'flowchart';

const simpleSyntaxHighlight = (code: string) => {
    const escapedCode = code
        .replace(/&/g, '&amp;')
```

```
        .replace(/</g, '&lt;')
        .replace(/>/g, '&gt;');

    return escapedCode
        .replace(/\b(const|let|var|function|return|if|for|=>|import|from|export|default)
        \b/g, '<span class="text-indigo-400 font-semibold">$1</span>')
        .replace(/(\`|'|")(.*?)(\`|'|")/g, '<span class="text-
        emerald-400">$1$2$3</span>')
        .replace(/(\/\/.*)/g, '<span class="text-gray-400 italic">$1</span>')
        .replace(/(\{|\}|\(|\)|\[|\])/g, '<span class="text-gray-400">$1</span>');
};

mermaid.initialize({ startOnLoad: false, theme: 'neutral', securityLevel:
'loose' });

export const AiCodeExplainer: React.FC<{ initialCode?: string }> = ({
initialCode }) => {
    const [code, setCode] = useState<string>(initialCode || exampleCode);
    const [explanation, setExplanation] = useState<StructuredExplanation |
    null>(null);
    const [mermaidCode, setMermaidCode] = useState<string>('');
    const [isLoading, setIsLoading] = useState<boolean>(false);
    const [error, setError] = useState<string>('');
    const [activeTab, setActiveTab] = useState<ExplanationTab>('summary');
    const textareaRef = useRef<HTMLTextAreaElement>(null);
    const preRef = useRef<HTMLPreElement>(null);
    const mermaidContainerRef = useRef<HTMLDivElement>(null);

    const handleExplain = useCallback(async (codeToExplain: string) => {
        if (!codeToExplain.trim()) {
            setError('Please enter some code to explain.');
            return;
        }
        setIsLoading(true);
        setError('');
        setExplanation(null);
        setMermaidCode('');
        setActiveTab('summary');
        try {
            const [explanationResult, mermaidResult] = await Promise.all([
                explainCodeStructured(codeToExplain),
                generateMermaidJs(codeToExplain)
            ]);
            setExplanation(explanationResult);
            setMermaidCode(mermaidResult.replace(/```mermaid\n|```/g, ''));

        } catch (err) {
            const errorMessage = err instanceof Error ? err.message : 'An unknown error
            occurred.';
            setError(`Failed to get explanation: ${errorMessage}`);
        } finally {
            setIsLoading(false);
        }
    }, []);

    useEffect(() => {
        if (initialCode) {
            setCode(initialCode);
            handleExplain(initialCode);
        }
    }, [initialCode, handleExplain]);
    useEffect(() => {
```

```
    const renderMermaid = async () => {
         if (activeTab === 'flowchart' && mermaidCode && mermaidContainerRef.current) {
            try {
                mermaidContainerRef.current.innerHTML = ''; // Clear previous
                const { svg } = await mermaid.render(`mermaid-graph-${Date.now()}`,
                mermaidCode);
                mermaidContainerRef.current.innerHTML = svg;
            } catch (e) {
                console.error("Mermaid rendering error:", e);
                mermaidContainerRef.current.innerHTML = `<p class="text-red-500">Error rendering
                flowchart.</p>`;
            }
        }
    }
    renderMermaid();
}, [activeTab, mermaidCode]);


const handleScroll = () => {
    if (preRef.current && textareaRef.current) {
        preRef.current.scrollTop = textareaRef.current.scrollTop;
        preRef.current.scrollLeft = textareaRef.current.scrollLeft;
    }
};

const highlightedCode = useMemo(() => simpleSyntaxHighlight(code), [code]);

const renderTabContent = () => {
    if (!explanation) return null;
    switch(activeTab) {
        case 'summary':
            return <MarkdownRenderer content={explanation.summary} />;
        case 'lineByLine':
            return (
                <div className="space-y-3">
                    {explanation.lineByLine.map((item, index) => (
                        <div key={index} className="p-3 bg-background rounded-md border border-border">
                            <p className="font-mono text-xs text-primary mb-1">Lines: {item.lines}</p>
                            <p className="text-sm">{item.explanation}</p>
                        </div>
                    ))}
                </div>
            );
        case 'complexity':
            return (
                <div>
                    <p><strong>Time Complexity:</strong> <span className="font-mono text-
                    amber-600">{explanation.complexity.time}</span></p>
                    <p><strong>Space Complexity:</strong> <span className="font-mono text-
                    amber-600">{explanation.complexity.space}</span></p>
                </div>
            );
        case 'suggestions':
            return (
                 <ul className="list-disc list-inside space-y-2">
                    {explanation.suggestions.map((item, index) => <li key={index}>{item}</li>)}
                </ul>
            );
        case 'flowchart':
            return (
                <div ref={mermaidContainerRef} className="w-full h-full flex items-center
                justify-center">
```

```
                    <LoadingSpinner />
                </div>
            );
    }
}

return (
    <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
        <header className="mb-6 flex-shrink-0">
            <h1 className="text-3xl font-bold flex items-center">
                <CpuChipIcon />
                <span className="ml-3">AI Code Explainer</span>
            </h1>
            <p className="text-text-secondary mt-1">Get a detailed, structured analysis of
            any code snippet.</p>
        </header>
        <div className="flex-grow grid grid-cols-1 md:grid-cols-2 gap-6 min-h-0">

            {/* Left Column: Code Input */}
            <div className="flex flex-col min-h-0 md:col-span-1">
                <label htmlFor="code-input" className="text-sm font-medium text-text-secondary
                mb-2">Your Code</label>
                <div className="relative flex-grow bg-surface border border-border rounded-md
                focus-within:ring-2 focus-within:ring-primary overflow-hidden">
                    <textarea
                        ref={textareaRef}
                        id="code-input"
                        value={code}
                        onChange={(e) => setCode(e.target.value)}
                        onScroll={handleScroll}
                        placeholder="Paste your code here..."
                        spellCheck="false"
                        className="absolute inset-0 w-full h-full p-4 bg-transparent resize-none font-
                        mono text-sm text-transparent caret-primary outline-none z-10"
                    />
                    <pre
                        ref={preRef}
                        aria-hidden="true"
                        className="absolute inset-0 w-full h-full p-4 font-mono text-sm text-text-
                        primary pointer-events-none z-0 whitespace-pre-wrap overflow-auto no-scrollbar"
                        dangerouslySetInnerHTML={{ __html: highlightedCode + '\n' }}
                    />
                </div>
                <div className="mt-4 flex-shrink-0">
                    <button
                        onClick={() => handleExplain(code)}
                        disabled={isLoading}
                        className="btn-primary w-full flex items-center justify-center px-6 py-3"
                    >
                        {isLoading ? <LoadingSpinner/> : 'Analyze Code'}
                    </button>
                </div>
            </div>

            {/* Right Column: AI Analysis */}
            <div className="flex flex-col min-h-0 md:col-span-1">
                <label className="text-sm font-medium text-text-secondary mb-2">AI
                Analysis</label>
                <div className="relative flex-grow flex flex-col bg-surface border border-border
                rounded-md overflow-hidden">
                    <div className="flex-shrink-0 flex border-b border-border">
                       {(['summary', 'lineByLine', 'complexity', 'suggestions', 'flowchart'] as
```

```
                        ExplanationTab[]).map(tab => (
                            <button key={tab} onClick={() => setActiveTab(tab)} disabled={!explanation}
                             className={`px-4 py-2 text-sm font-medium capitalize transition-colors
                             ${activeTab === tab ? 'bg-background text-primary font-semibold' : 'text-text-
                             secondary hover:bg-gray-100 dark:hover:bg-slate-700 disabled:text-gray-400
                             dark:disabled:text-slate-500'}`}>
                                {tab.replace(/([A-Z])/g, ' $1')}
                            </button>
                        ))}
                    </div>
                    <div className="p-4 flex-grow overflow-y-auto">
                        {isLoading && <div className="flex items-center justify-center
                        h-full"><LoadingSpinner /></div>}
                        {error && <p className="text-red-500">{error}</p>}
                        {explanation && !isLoading && renderTabContent()}
                        {!isLoading && !explanation && !error && <div className="text-text-secondary
                        h-full flex items-center justify-center">The analysis will appear here.</div>}
                    </div>
                </div>
            </div>
        </div>
    </div>
    );
};

// ===== SnippetVault_12.tsx =====

import React, { useState, useEffect, useMemo } from 'react';
import { LockClosedIcon, SparklesIcon, TrashIcon, ClipboardDocumentIcon,
ArrowDownTrayIcon } from '../icons.tsx';
import { useLocalStorage } from '../../hooks/useLocalStorage.ts';
import { enhanceSnippetStream, generateTagsForCode } from
'../../services/geminiService.ts';
import { LoadingSpinner } from '../shared/index.tsx';
import { downloadFile } from '../../services/fileUtils.ts';
import { useNotification } from '../../contexts/NotificationContext.tsx';

interface Snippet {
    id: number; name: string; code: string; language: string; tags: string[];
}

const langToExt: Record<string, string> = {
    javascript: 'js',
    typescript: 'ts',
    python: 'py',
    css: 'css',
    html: 'html',
    json: 'json',
    markdown: 'md',
    plaintext: 'txt',
};

export const SnippetVault: React.FC = () => {
    const [snippets, setSnippets] = useLocalStorage<Snippet[]>('devcore_snippets',
    [{ id: 1, name: 'React Hook Boilerplate', language: 'javascript', code: `import
    { useState } from 'react';\n\nconst useCustomHook = () => {\n  const [value,
    setValue] = useState(null);\n  return { value, setValue };\n};`, tags: ['react',
    'hook'] }]);
    const [activeSnippet, setActiveSnippet] = useState<Snippet | null>(null);
    const [isEnhancing, setIsEnhancing] = useState(false);
    const [searchTerm, setSearchTerm] = useState('');
    const [isEditingName, setIsEditingName] = useState(false);
```

```
const { addNotification } = useNotification();

const filteredSnippets = useMemo(() => {
    if (!searchTerm) return snippets;
    const lowerSearch = searchTerm.toLowerCase();
    return snippets.filter((s: Snippet) =>
        s.name.toLowerCase().includes(lowerSearch) ||
        s.code.toLowerCase().includes(lowerSearch) ||
        (s.tags && s.tags.some(t => t.toLowerCase().includes(lowerSearch)))
    );
}, [snippets, searchTerm]);

useEffect(() => {
    if (!activeSnippet && filteredSnippets.length > 0)
    setActiveSnippet(filteredSnippets[0]);
    if (activeSnippet) setActiveSnippet(snippets.find((s: Snippet) => s.id ===
    activeSnippet.id) || null);
}, [snippets, activeSnippet, filteredSnippets]);

const updateSnippet = (snippet: Snippet) => {
    setSnippets(snippets.map((s: Snippet) => s.id === snippet.id ? snippet : s));
    setActiveSnippet(snippet);
};

const handleEnhance = async () => {
    if (!activeSnippet) return;
    setIsEnhancing(true);
    try {
        const stream = enhanceSnippetStream(activeSnippet.code);
        let fullResponse = '';
        for await (const chunk of stream) {
            fullResponse += chunk;
            updateSnippet({ ...activeSnippet, code: fullResponse.replace(/^```(?:\w+\n)?/,
            '').replace(/```$/, '') });
        }
    } finally { setIsEnhancing(false); }
};

const handleAiTagging = async (snippet: Snippet) => {
    if (!snippet.code.trim()) return;
    try {
        const suggestedTags = await generateTagsForCode(snippet.code);
        const newTags = [...new Set([...(snippet.tags || []), ...suggestedTags])];
        updateSnippet({...snippet, tags: newTags});
        addNotification('AI tags added!', 'success');
    } catch(e) {
        console.error("AI tagging failed:", e);
        addNotification('AI tagging failed.', 'error');
    }
};

const handleAddNew = () => {
    const newSnippet: Snippet = { id: Date.now(), name: 'New Snippet', language:
    'plaintext', code: '', tags: [] };
    setSnippets([...snippets, newSnippet]);
    setActiveSnippet(newSnippet);
};

const handleDelete = (id: number) => {
    setSnippets(snippets.filter((s: Snippet) => s.id !== id));
    if(activeSnippet?.id === id) setActiveSnippet(filteredSnippets.length > 1 ?
    filteredSnippets[0] : null);
```

```
};

const handleDownload = () => {
    if(!activeSnippet) return;
    const extension = langToExt[activeSnippet.language] || 'txt';
    const filename = `${activeSnippet.name.replace(/\s/g, '_')}.${extension}`;
    downloadFile(activeSnippet.code, filename);
}

const handleNameChange = (e: React.ChangeEvent<HTMLInputElement>) => {
    if (activeSnippet) updateSnippet({...activeSnippet, name: e.target.value});
};

const handleTagsChange = (e: React.KeyboardEvent<HTMLInputElement>) => {
    if (e.key === 'Enter' && activeSnippet) {
        const newTag = e.currentTarget.value.trim();
        if (newTag && !activeSnippet.tags.includes(newTag)) {
            updateSnippet({...activeSnippet, tags: [...(activeSnippet.tags ?? []),
            newTag]});
        }
        e.currentTarget.value = '';
    }
};

return (
    <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
        <header className="mb-6"><h1 className="text-3xl font-bold flex items-
        center"><LockClosedIcon /><span className="ml-3">Snippet Vault</span></h1><p
        className="text-text-secondary mt-1">Store, search, tag, and enhance your
        reusable code snippets with AI.</p></header>
        <div className="flex-grow flex gap-6 min-h-0">
            <aside className="w-1/3 bg-surface border border-border p-4 rounded-lg flex
            flex-col">
                <input type="text" placeholder="Search snippets..." value={searchTerm}
                onChange={e => setSearchTerm(e.target.value)} className="w-full px-3 py-1.5 mb-3
                rounded-md bg-background border border-border text-sm"/>
                <ul className="space-y-2 flex-grow overflow-y-auto
                pr-2">{filteredSnippets.map((s: Snippet) => (<li key={s.id} className="group
                flex items-center justify-between"><button onClick={() => setActiveSnippet(s)}
                className={`w-full text-left px-3 py-2 rounded-md ${activeSnippet?.id === s.id ?
                'bg-primary/10 text-primary' : 'hover:bg-gray-100 dark:hover:bg-
                slate-700'}`}>{s.name}</button><div className="flex opacity-0 group-
                hover:opacity-100 transition-opacity"><button onClick={() =>
                {navigator.clipboard.writeText(s.code); addNotification("Copied snippet!",
                "success")}} className="ml-2 p-1 text-text-secondary hover:text-primary"
                title="Copy"><ClipboardDocumentIcon /></button><button onClick={() =>
                handleDelete(s.id)} className="ml-2 p-1 text-text-secondary hover:text-red-500"
                title="Delete"><TrashIcon/></button></div></li>))}</ul>
                <div className="mt-4 pt-4 border-t border-border"><button onClick={handleAddNew}
                className="btn-primary w-full text-sm py-2">Add New Snippet</button></div>
            </aside>
            <main className="w-2/3 flex flex-col">
                {activeSnippet ? (<>
                    <div className="flex justify-between items-center mb-2">
                        {isEditingName ? <input type="text" value={activeSnippet.name}
                        onChange={handleNameChange} onBlur={() => setIsEditingName(false)} autoFocus
                        className="text-lg font-bold bg-gray-100 dark:bg-slate-700 rounded px-2"/> : <h3
                        onDoubleClick={() => setIsEditingName(true)} className="text-lg font-bold
                        cursor-pointer">{activeSnippet.name}</h3>}
                        <div className="flex gap-2">
                            <button onClick={() => handleAiTagging(activeSnippet)} className="flex items-
                            center gap-2 px-3 py-1 bg-teal-500/80 text-white font-bold text-xs rounded-
```

```
                                md"><SparklesIcon /> AI Tag</button>
                                <button onClick={handleEnhance} disabled={isEnhancing} className="flex items-
                                center gap-2 px-3 py-1 bg-purple-500/80 text-white font-bold text-xs rounded-md
                                disabled:bg-gray-400"><SparklesIcon /> AI Enhance</button>
                                <button onClick={handleDownload} className="flex items-center gap-1 px-3 py-1
                                bg-gray-100 dark:bg-slate-700 text-xs rounded-md"><ArrowDownTrayIcon
                                className="w-4 h-4"/> Download</button>
                            </div>
                        </div>
                        <textarea value={activeSnippet.code} onChange={e =>
                        updateSnippet({...activeSnippet, code: e.target.value})} className="flex-grow
                        p-4 bg-surface border border-border rounded-md resize-none font-mono text-sm
                        focus:ring-2 focus:ring-primary focus:outline-none"/>
                        <div className="mt-2 text-xs text-text-secondary">
                           <div className="flex items-center gap-2 flex-wrap">
                             <span className="font-bold">Tags:</span> {(activeSnippet.tags ?? []).map(t =>
                             <span key={t} className="bg-gray-200 dark:bg-slate-700 px-2 py-0.5 rounded-
                             full">{t}</span>)}
                             <input type="text" placeholder="+ Add tag" onKeyDown={handleTagsChange}
                             className="bg-transparent border-b border-border focus:outline-none
                             focus:border-primary w-24 text-xs px-1"/>
                           </div>
                        </div>
                    </>) : (<div className="flex-grow flex items-center justify-center bg-background
                    border border-border rounded-lg text-text-secondary">Select a snippet or create
                    a new one.</div>)}
                </main>
            </div>
        </div>
    );
};

// ===== AiFeatureBuilder_12.tsx =====

import React, { useState, useCallback, useEffect } from 'react';
import type { GeneratedFile } from '../../types.ts';
import { generateFeature, generateUnitTestsStream, generateCommitMessageStream,
generateDockerfile } from '../../services/geminiService.ts';
import { saveFile, getAllFiles, clearAllFiles } from
'../../services/dbService.ts';
import { CpuChipIcon, DocumentTextIcon, BeakerIcon, GitBranchIcon, CloudIcon }
from '../icons.tsx';
import { LoadingSpinner, MarkdownRenderer } from '../shared/index.tsx';

type ActiveTab = 'CODE' | 'TESTS' | 'COMMIT' | 'DEPLOYMENT';

export const AiFeatureBuilder: React.FC = () => {
    const [prompt, setPrompt] = useState<string>('A simple "Hello World" React
    component with a button that shows an alert.');
    const [framework, setFramework] = useState('React');
    const [styling, setStyling] = useState('Tailwind CSS');
    const [generatedFiles, setGeneratedFiles] = useState<GeneratedFile[]>([]);
    const [unitTests, setUnitTests] = useState<string>('');
    const [commitMessage, setCommitMessage] = useState<string>('');
    const [dockerfile, setDockerfile] = useState<string>('');
    const [selectedFile, setSelectedFile] = useState<GeneratedFile | null>(null);
    const [isLoading, setIsLoading] = useState<boolean>(false);
    const [error, setError] = useState<string>('');
    const [activeTab, setActiveTab] = useState<ActiveTab>('CODE');

    useEffect(() => {
        const loadFiles = async () => {
```

```
        const files = await getAllFiles();
        setGeneratedFiles(files);
        if (files.length > 0) setSelectedFile(files[0]);
    };
    loadFiles();
}, []);

const handleGenerate = useCallback(async () => {
    if (!prompt.trim()) { setError('Please enter a feature description.'); return; }
    setIsLoading(true);
    setError('');
    await clearAllFiles(); // Start fresh for each generation
    setGeneratedFiles([]);
    setUnitTests('');
    setCommitMessage('');
    setDockerfile('');
    setSelectedFile(null);
    setActiveTab('CODE');

    try {
        const resultFiles = await generateFeature(prompt, framework, styling);
        for (const file of resultFiles) { await saveFile(file); }
        setGeneratedFiles(resultFiles);

        if (resultFiles.length > 0) {
            const componentFile = resultFiles.find(f => f.filePath.endsWith('.tsx') ||
            f.filePath.endsWith('.jsx'));
            setSelectedFile(componentFile || resultFiles[0]);

            // Kick off supplemental generations in parallel
            const testStream = generateUnitTestsStream(componentFile?.content ||
            resultFiles[0].content);
            const diffContext = resultFiles.map(f => `File:
            ${f.filePath}\n\n${f.content}`).join('\n---\n');
            const commitStream = generateCommitMessageStream(diffContext);
            const dockerfileStream = generateDockerfile(framework);

            // Process streams
            let tests = '';
            for await (const chunk of testStream) { tests += chunk; setUnitTests(tests); }

            let commit = '';
            for await (const chunk of commitStream) { commit += chunk;
            setCommitMessage(commit); }

            let docker = '';
            for await (const chunk of dockerfileStream) { docker += chunk;
            setDockerfile(docker); }
        }
    } catch (err) {
        const errorMessage = err instanceof Error ? err.message : 'An unknown error
        occurred.';
        setError(`Failed to generate feature: ${errorMessage}`);
    } finally {
        setIsLoading(false);
    }
}, [prompt, framework, styling]);

const renderContent = () => {
    switch (activeTab) {
        case 'TESTS': return <MarkdownRenderer content={unitTests} />;
        case 'COMMIT': return <pre className="w-full h-full p-4 whitespace-pre-wrap
```

```
        font-sans text-sm text-text-primary">{commitMessage}</pre>;
        case 'DEPLOYMENT': return <MarkdownRenderer content={dockerfile} />;
        case 'CODE':
        default:
             return selectedFile ? <MarkdownRenderer content={'```tsx\n' +
             selectedFile.content + '\n```'} /> : <div className="flex items-center justify-
             center h-full text-text-secondary">Select a file to view its content.</div>;
    }
}

return (
    <div className="h-full flex flex-col text-text-primary bg-surface">
        <header className="p-4 border-b border-border flex-shrink-0">
            <h1 className="text-xl font-bold flex items-center"><CpuChipIcon /><span
            className="ml-3">AI Feature Builder</span></h1>
        </header>

        <div className="flex-grow flex min-h-0">
            <aside className="w-64 bg-surface border-r border-border p-4 flex flex-col">
                <h2 className="text-sm font-semibold text-text-secondary mb-2">Generated
                Files</h2>
                <div className="overflow-y-auto space-y-1">
                    {generatedFiles.map(file => (
                        <div key={file.filePath} onClick={() => { setSelectedFile(file);
                        setActiveTab('CODE'); }} className={`flex items-center space-x-2 p-2 rounded-md
                        cursor-pointer text-sm ${selectedFile?.filePath === file.filePath && activeTab
                        === 'CODE' ? 'bg-primary/10 text-primary' : 'hover:bg-gray-100 dark:hover:bg-
                        slate-700'}`}>
                            <DocumentTextIcon /><span>{file.filePath.split('/').pop()}</span>
                        </div>
                    ))}
                </div>
            </aside>

            <main className="flex-1 flex flex-col min-w-0">
                <div className="flex-grow flex flex-col bg-background">
                     <div className="border-b border-border flex items-center bg-surface">
                        <button onClick={() => setActiveTab('CODE')} className={`flex items-center gap-2
                        px-4 py-2 text-sm ${activeTab === 'CODE' ? 'bg-background border-b-2 border-
                        primary text-text-primary' : 'text-text-secondary hover:bg-gray-50
                        dark:hover:bg-slate-700'}`}><DocumentTextIcon /> Code</button>
                        {unitTests && <button onClick={() => setActiveTab('TESTS')} className={`flex
                        items-center gap-2 px-4 py-2 text-sm ${activeTab === 'TESTS' ? 'bg-background
                        border-b-2 border-primary text-text-primary' : 'text-text-secondary hover:bg-
                        gray-50 dark:hover:bg-slate-700'}`}><BeakerIcon /> Tests</button>}
                        {commitMessage && <button onClick={() => setActiveTab('COMMIT')}
                        className={`flex items-center gap-2 px-4 py-2 text-sm ${activeTab === 'COMMIT' ?
                        'bg-background border-b-2 border-primary text-text-primary' : 'text-text-
                        secondary hover:bg-gray-50 dark:hover:bg-slate-700'}`}><GitBranchIcon />
                        Commit</button>}
                        {dockerfile && <button onClick={() => setActiveTab('DEPLOYMENT')}
                        className={`flex items-center gap-2 px-4 py-2 text-sm ${activeTab ===
                        'DEPLOYMENT' ? 'bg-background border-b-2 border-primary text-text-primary' :
                        'text-text-secondary hover:bg-gray-50 dark:hover:bg-slate-700'}`}><CloudIcon />
                        Deployment</button>}
                    </div>
                    <div className="flex-grow p-2 overflow-auto">
                        {isLoading && !generatedFiles.length ? <div className="flex justify-center
                        items-center h-full"><LoadingSpinner/></div> : renderContent()}
                    </div>
                </div>
                <div className="flex-shrink-0 p-4 border-t border-border bg-surface">
```

```
                        <div className="grid grid-cols-2 gap-2 mb-2">
                           <select value={framework} onChange={e => setFramework(e.target.value)}
                           className="w-full p-2 bg-background border border-border rounded-md text-
                           sm"><option>React</option><option>Vue</option><option>Svelte</option></select>
                           <select value={styling} onChange={e => setStyling(e.target.value)}
                           className="w-full p-2 bg-background border border-border rounded-md text-
                           sm"><option>Tailwind CSS</option><option>CSS Modules</option><option>Styled
                           Components</option></select>
                        </div>
                        <textarea value={prompt} onChange={(e) => setPrompt(e.target.value)}
                        placeholder="e.g., A user profile card with an avatar, name, and bio."
                        className="w-full p-2 bg-background border border-border rounded-md resize-none
                        text-sm h-20"/>
                         <button onClick={handleGenerate} disabled={isLoading} className="btn-primary
                         mt-2 w-full flex items-center justify-center gap-2 px-4 py-2">
                            {isLoading ? <><LoadingSpinner /> Generating...</> : 'Generate Feature'}
                        </button>
                         {error && <p className="text-red-600 text-xs mt-2 text-center">{error}</p>}
                    </div>
                </main>
            </div>
        </div>
    );
};

// ===== ProjectExplorer_10.tsx =====

import React, { useState, useEffect, useCallback } from 'react';
import { useGlobalState } from '../../contexts/GlobalStateContext.tsx';
import { useVaultModal } from '../../contexts/VaultModalContext.tsx';
import { useNotification } from '../../contexts/NotificationContext.tsx';
import * as vaultService from '../../services/vaultService.ts';
import { initializeOctokit } from '../../services/authService.ts';
import { getRepos, getRepoTree, getFileContent, commitFiles } from
'../../services/githubService.ts';
import { generateCommitMessageStream } from '../../services/geminiService.ts';
import type { Repo, FileNode } from '../../types.ts';
import { FolderIcon, DocumentIcon } from '../icons.tsx';
import { LoadingSpinner } from '../shared/index.tsx';
import * as Diff from 'diff';

const FileTree: React.FC<{ node: FileNode, onFileSelect: (path: string, name:
string) => void, activePath: string | null }> = ({ node, onFileSelect,
activePath }) => {
    const [isOpen, setIsOpen] = useState(true);

    if (node.type === 'file') {
        const isActive = activePath === node.path;
        return (
            <div
                className={`flex items-center space-x-2 pl-4 py-1 cursor-pointer rounded
                ${isActive ? 'bg-primary/10 text-primary' : 'hover:bg-gray-100 dark:hover:bg-
                slate-700'}`}
                onClick={() => onFileSelect(node.path, node.name)}
            >
                <DocumentIcon />
                <span>{node.name}</span>
            </div>
        );
    }

    return (
```

```
        <div>
            <div
                className="flex items-center space-x-2 py-1 cursor-pointer hover:bg-gray-100
                dark:hover:bg-slate-700 rounded"
                onClick={() => setIsOpen(!isOpen)}
            >
                <div className={`transform transition-transform ${isOpen ? 'rotate-90' :
                ''}`}>■</div>
                <FolderIcon />
                <span className="font-semibold">{node.name}</span>
            </div>
            {isOpen && node.children && (
                <div className="pl-4 border-l border-border ml-3">
                    {node.children.map(child => <FileTree key={child.path} node={child}
                    onFileSelect={onFileSelect} activePath={activePath} />)}
                </div>
            )}
        </div>
    );
};

export const ProjectExplorer: React.FC = () => {
    const { state, dispatch } = useGlobalState();
    const { vaultState, connections, selectedRepo, projectFiles } = state;
    const { requestUnlock } = useVaultModal();
    const { addNotification } = useNotification();
    const [repos, setRepos] = useState<Repo[]>([]);
    const [isLoading, setIsLoading] = useState<'repos' | 'tree' | 'file' | 'commit'
    | null>(null);
    const [error, setError] = useState('');
    const [activeFile, setActiveFile] = useState<{ path: string; name: string;
    originalContent: string; editedContent: string} | null>(null);

    const getApiClient = useCallback(async () => {
        if (!vaultState.isUnlocked) {
            const unlocked = await requestUnlock();
            if (!unlocked) throw new Error("Vault must be unlocked to access GitHub.");
        }
        const token = await vaultService.getDecryptedCredential('github_pat');
        if (!token) throw new Error("GitHub token not found in vault.");
        return initializeOctokit(token);
    }, [vaultState.isUnlocked, requestUnlock]);


    useEffect(() => {
        const loadRepos = async () => {
            if (connections.github) {
                setIsLoading('repos');
                setError('');
                try {
                    const octokit = await getApiClient();
                    const userRepos = await getRepos(octokit);
                    setRepos(userRepos);
                } catch (err) {
                    setError(err instanceof Error ? err.message : 'Failed to load repositories');
                } finally {
                    setIsLoading(null);
                }
            }
        };
        loadRepos();
    }, [connections.github, getApiClient]);
```

```
useEffect(() => {
    const loadTree = async () => {
         if (selectedRepo && connections.github) {
            setIsLoading('tree');
            setError('');
            setActiveFile(null);
            try {
                const octokit = await getApiClient();
                const tree = await getRepoTree(octokit, selectedRepo.owner, selectedRepo.repo);
                dispatch({ type: 'LOAD_PROJECT_FILES', payload: tree });
            } catch (err) {
                 setError(err instanceof Error ? err.message : 'Failed to load repository tree');
            } finally {
                setIsLoading(null);
            }
        }
    };
    loadTree();
}, [selectedRepo, connections.github, dispatch, getApiClient]);

const handleFileSelect = async (path: string, name: string) => {
    if (!selectedRepo) return;
    setIsLoading('file');
    try {
        const octokit = await getApiClient();
        const content = await getFileContent(octokit, selectedRepo.owner,
        selectedRepo.repo, path);
        setActiveFile({ path, name, originalContent: content, editedContent: content });
    } catch (err) {
        setError((err as Error).message);
    } finally {
        setIsLoading(null);
    }
};

const handleCommit = async () => {
    if (!activeFile || !selectedRepo || activeFile.originalContent ===
    activeFile.editedContent) return;

    setIsLoading('commit');
    setError('');
    try {
        const diff = Diff.createPatch(activeFile.path, activeFile.originalContent,
        activeFile.editedContent);

        const stream = generateCommitMessageStream(diff);
        let commitMessage = '';
        for await (const chunk of stream) { commitMessage += chunk; }

        const finalMessage = window.prompt("Confirm or edit commit message:",
        commitMessage);
        if (!finalMessage) {
            setIsLoading(null);
            return;
        }

        const octokit = await getApiClient();
        const commitUrl = await commitFiles(
            octokit,
            selectedRepo.owner,
            selectedRepo.repo,
            [{ path: activeFile.path, content: activeFile.editedContent }],
```

```
            finalMessage
        );

        addNotification(`Successfully committed to ${selectedRepo.repo}`, 'success');
        setActiveFile(prev => prev ? { ...prev, originalContent: prev.editedContent } :
        null);

    } catch (err) {
        const message = err instanceof Error ? err.message : 'Failed to commit changes';
        setError(message);
        addNotification(message, 'error');
    } finally {
        setIsLoading(null);
    }
};

if (!connections.github) {
    return (
        <div className="h-full flex flex-col items-center justify-center text-center
        text-text-secondary p-4">
            <FolderIcon />
            <h2 className="text-lg font-semibold mt-2">Connect to GitHub</h2>
            <p>Please go to the "Security Vault" tab and provide a Personal Access Token to
            explore your repositories.</p>
        </div>
    );
}

const hasChanges = activeFile ? activeFile.originalContent !==
activeFile.editedContent : false;

return (
    <div className="h-full flex flex-col text-text-primary">
        <header className="p-4 border-b border-border flex-shrink-0">
            <h1 className="text-xl font-bold flex items-center"><FolderIcon /><span
            className="ml-3">Project Explorer</span></h1>
            <div className="mt-2">
                <select
                    value={selectedRepo ? `${selectedRepo.owner}/${selectedRepo.repo}` : ''}
                    onChange={e => {
                        const [owner, repo] = e.target.value.split('/');
                        dispatch({ type: 'SET_SELECTED_REPO', payload: { owner, repo } });
                    }}
                    className="w-full p-2 bg-surface border border-border rounded-md text-sm"
                >
                    <option value="" disabled>{isLoading === 'repos' ? 'Loading...' : 'Select a
                    repository'}</option>
                    {repos.map(r => <option key={r.id} value={r.full_name}>{r.full_name}</option>)}
                </select>
            </div>
            {error && <p className="text-red-500 text-xs mt-2">{error}</p>}
        </header>
        <div className="flex-grow flex min-h-0">
            <aside className="w-1/3 bg-background border-r border-border p-4 overflow-y-
            auto">
                {isLoading === 'tree' && <div className="flex justify-center"><LoadingSpinner
                /></div>}
                {projectFiles && <FileTree node={projectFiles} onFileSelect={handleFileSelect}
                activePath={activeFile?.path ?? null} />}
            </aside>
            <main className="flex-1 bg-surface flex flex-col">
                 <div className="flex justify-between items-center p-2 border-b border-border bg-
```

```
                    gray-50 dark:bg-slate-800">
                       <span className="text-sm font-semibold">{activeFile?.name || 'No file
                       selected'}</span>
                       <button onClick={handleCommit} disabled={!hasChanges || isLoading === 'commit'}
                       className="btn-primary px-4 py-1 text-sm flex items-center justify-center
                       min-w-[100px]">
                          {isLoading === 'commit' ? <LoadingSpinner/> : 'Commit'}
                       </button>
                    </div>
                    {isLoading === 'file' ? <div className="flex items-center justify-center
                    h-full"><LoadingSpinner /></div> :
                       <textarea
                           value={activeFile?.editedContent ?? 'Select a file to view its content.'}
                           onChange={e => setActiveFile(prev => prev ? { ...prev, editedContent:
                           e.target.value } : null)}
                           disabled={!activeFile}
                           className="w-full h-full p-4 text-sm font-mono bg-transparent resize-none
                           focus:outline-none"
                       />
                    }
                </main>
            </div>
        </div>
    );
};

// ===== ApiMockGenerator.tsx =====

import React, { useState, useEffect } from 'react';
import { generateMockData } from '../../services/geminiService.ts';
import { startMockServer, stopMockServer, setMockRoutes, isMockServerRunning }
from '../../services/mocking/mockServer.ts';
import { saveMockCollection, getAllMockCollections, deleteMockCollection } from
'../../services/mocking/db.ts';
import { ServerStackIcon, SparklesIcon, PlusIcon, TrashIcon } from
'../icons.tsx';
import { LoadingSpinner } from '../shared/index.tsx';

const exampleSchema = "a user with an id, name, email, and a nested address
object containing a city and country";

export const ApiMockGenerator: React.FC = () => {
    const [schema, setSchema] = useState(exampleSchema);
    const [count, setCount] = useState(5);
    const [collectionName, setCollectionName] = useState('users');
    const [collections, setCollections] = useState<any[]>([]);
    const [generatedData, setGeneratedData] = useState<any[] | null>(null);
    const [isLoading, setIsLoading] = useState(false);
    const [error, setError] = useState('');
    const [isServerRunning, setIsServerRunning] = useState(isMockServerRunning());
    const [routes, setRoutes] = useState([{ path: '/api/users', method: 'GET' }]);

    useEffect(() => {
        const loadCollections = async () => {
            const storedCollections = await getAllMockCollections();
            setCollections(storedCollections);
        };
        loadCollections();
    }, []);

    const handleGenerate = async () => {
        if (!schema.trim() || !collectionName.trim()) {
```

```
        setError('Schema description and collection name are required.');
        return;
    }
    setIsLoading(true);
    setError('');
    try {
        const data = await generateMockData(schema, count);
        setGeneratedData(data);
        const collectionId = collectionName.toLowerCase().replace(/\s/g, '-');
        await saveMockCollection({ id: collectionId, schemaDescription: schema, data });
        setCollections(await getAllMockCollections());
    } catch (err) {
        setError(err instanceof Error ? err.message : 'Failed to generate data.');
    } finally {
        setIsLoading(false);
    }
};

const handleServerToggle = async () => {
    if (isServerRunning) {
        await stopMockServer();
        setIsServerRunning(false);
    } else {
        try {
            await startMockServer();
            setIsServerRunning(true);
            updateRoutes();
        } catch (err) {
            setError(err instanceof Error ? err.message : 'Could not start server.');
        }
    }
};

const updateRoutes = () => {
    const mockRoutes = routes.map(route => {
        // A simple implementation: find first matching collection for path
        const matchingCollection = collections.find(c => route.path.includes(c.id));
        return {
            ...route,
            response: {
                status: 200,
                body: matchingCollection ? matchingCollection.data : { message: 'No data found
                for this route.' }
            }
        };
    });
    setMockRoutes(mockRoutes as any);
};

useEffect(() => {
    if (isServerRunning) {
        updateRoutes();
    }
}, [routes, collections, isServerRunning]);

return (
    <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
        <header className="mb-6 flex justify-between items-start">
            <div>
                <h1 className="text-3xl font-bold flex items-center"><ServerStackIcon /><span
                className="ml-3">AI API Mock Server</span></h1>
                <p className="text-text-secondary mt-1">Generate and serve mock API data locally
```

```
        using a service worker.</p>
    </div>
    <button onClick={handleServerToggle} className={`px-4 py-2 rounded-md font-
    semibold flex items-center gap-2 ${isServerRunning ? 'bg-green-100 text-
    green-700' : 'bg-gray-100'}`}>
        <span className={`w-3 h-3 rounded-full ${isServerRunning ? 'bg-green-500' : 'bg-
        gray-400'}`}></span>
        {isServerRunning ? 'Server Running' : 'Server Stopped'}
    </button>
</header>
<div className="flex-grow grid grid-cols-1 lg:grid-cols-3 gap-6 min-h-0">
    {/* Left: Generator */}
    <div className="lg:col-span-1 flex flex-col gap-4 bg-surface p-4 border border-
    border rounded-lg">
        <h3 className="text-lg font-bold">1. Generate Data</h3>
        <div><label className="text-sm">Describe the data schema</label><textarea
        value={schema} onChange={e => setSchema(e.target.value)} className="w-full mt-1
        p-2 bg-background border border-border rounded" rows={4}/></div>
        <div className="flex gap-2">
            <div className="flex-grow"><label className="text-sm">Collection
            Name</label><input type="text" value={collectionName} onChange={e =>
            setCollectionName(e.target.value)} className="w-full mt-1 p-2 bg-background
            border border-border rounded"/></div>
            <div><label className="text-sm">Count</label><input type="number" value={count}
            onChange={e => setCount(Number(e.target.value))} className="w-20 mt-1 p-2 bg-
            background border border-border rounded"/></div>
        </div>
        <button onClick={handleGenerate} disabled={isLoading} className="btn-primary
        py-2 flex items-center justify-center gap-2">{isLoading ? <LoadingSpinner/> :
        <><SparklesIcon/> Generate & Save</>}</button>
        {error && <p className="text-red-500 text-xs">{error}</p>}
    </div>

    {/* Middle: Data & Routes */}
    <div className="lg:col-span-2 flex flex-col gap-4 min-h-0">
       <div className="bg-surface p-4 border border-border rounded-lg flex-grow flex
       flex-col min-h-0">
            <h3 className="text-lg font-bold mb-2">2. View Data & Configure Routes</h3>
            <div className="flex-grow grid grid-cols-2 gap-4 min-h-0">
                <div className="overflow-y-auto">
                    <h4 className="font-semibold text-sm mb-1">Saved Collections</h4>
                    {collections.map(c => <div key={c.id} className="text-xs p-2 bg-background
                    rounded border border-border mb-1">{c.id} ({c.data.length} items)</div>)}
                    <h4 className="font-semibold text-sm mb-1 mt-2">Last Generated Data</h4>
                    <pre className="text-xs p-2 bg-background rounded border border-border
                    whitespace-pre-wrap">{generatedData ? JSON.stringify(generatedData, null, 2) :
                    'No data generated yet.'}</pre>
                </div>
                <div className="overflow-y-auto">
                    <h4 className="font-semibold text-sm mb-1">Mock Routes</h4>
                    {routes.map((r, i) => <div key={i} className="flex gap-1 items-center
                    mb-1"><select value={r.method} className="p-1 text-xs bg-background border
                    rounded"><option>GET</option><option>POST</option></select><input type="text"
                    value={r.path} className="flex-grow p-1 text-xs bg-background border rounded"
                    /></div>)}
                     <p className="text-xs text-text-secondary mt-2">Routes are automatically mapped
                     to collections by name (e.g., `/api/users` maps to `users` collection).</p>
                </div>
            </div>
       </div>
    </div>
</div>
```

```
        </div>
    );
};


// ===== EnvManager.tsx =====

import React, { useState } from 'react';
import { downloadEnvFile } from '../../services/fileUtils.ts';
import { DocumentTextIcon, PlusIcon, TrashIcon, ArrowDownTrayIcon } from
'../icons.tsx';

interface EnvVar {
    id: number;
    key: string;
    value: string;
}

export const EnvManager: React.FC = () => {
    const [envVars, setEnvVars] = useState<EnvVar[]>([
        { id: 1, key: 'VITE_API_URL', value: 'https://api.example.com' },
        { id: 2, key: 'VITE_ENABLE_FEATURE_X', value: 'true' },
    ]);

    const handleAdd = () => {
        setEnvVars([...envVars, { id: Date.now(), key: '', value: '' }]);
    };

    const handleUpdate = (id: number, field: 'key' | 'value', val: string) => {
        setEnvVars(envVars.map(v => v.id === id ? { ...v, [field]: val } : v));
    };

    const handleRemove = (id: number) => {
        setEnvVars(envVars.filter(v => v.id !== id));
    };

    const handleDownload = () => {
        const envObject = envVars.reduce((acc, v) => {
            if (v.key) acc[v.key] = v.value;
            return acc;
        }, {} as Record<string, string>);
        downloadEnvFile(envObject);
    };

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
            <header className="mb-6">
                <h1 className="text-3xl font-bold flex items-center"><DocumentTextIcon /><span
                className="ml-3">Environment Variable Manager</span></h1>
                <p className="text-text-secondary mt-1">Create and manage your `.env` files with
                a simple interface.</p>
            </header>
            <div className="flex-grow bg-surface p-6 rounded-lg border border-border w-full
            max-w-4xl mx-auto overflow-y-auto">
                <div className="space-y-3">
                    <div className="grid grid-cols-12 gap-4 font-semibold text-sm text-text-
                    secondary px-2">
                        <div className="col-span-5">Key</div>
                        <div className="col-span-6">Value</div>
                        <div className="col-span-1"></div>
                    </div>
                    {envVars.map((v, index) => (
```

```
                        <div key={v.id} className="grid grid-cols-12 gap-4 items-center">
                            <div className="col-span-5">
                                <input
                                    type="text"
                                    value={v.key}
                                    onChange={e => handleUpdate(v.id, 'key', e.target.value)}
                                    placeholder={`KEY_${index + 1}`}
                                    className="w-full p-2 bg-background border border-border rounded-md font-mono
                                    text-sm"
                                />
                            </div>
                            <div className="col-span-6">
                                <input
                                    type="text"
                                    value={v.value}
                                    onChange={e => handleUpdate(v.id, 'value', e.target.value)}
                                    placeholder="value"
                                    className="w-full p-2 bg-background border border-border rounded-md font-mono
                                    text-sm"
                                />
                            </div>
                            <div className="col-span-1">
                                <button onClick={() => handleRemove(v.id)} className="p-2 text-text-secondary
                                hover:text-red-500 rounded-md"><TrashIcon /></button>
                            </div>
                        </div>
                    ))}
                </div>
                 <div className="mt-4 pt-4 border-t border-border flex justify-between items-
                 center">
                    <button onClick={handleAdd} className="flex items-center gap-2 px-4 py-2 bg-
                    gray-100 text-sm font-semibold rounded-md hover:bg-gray-200">
                        <PlusIcon /> Add Variable
                    </button>
                    <button onClick={handleDownload} disabled={envVars.length === 0} className="btn-
                    primary flex items-center gap-2 px-4 py-2">
                        <ArrowDownTrayIcon /> Download .env File
                    </button>
                </div>
            </div>
        </div>
    );
};


// ===== PerformanceProfiler.tsx =====

import React, { useState, useCallback } from 'react';
import { analyzePerformanceTrace } from '../../services/geminiService.ts';
import { startTracing, stopTracing, TraceEntry } from
'../../services/profiling/performanceService.ts';
import { parseViteStats, BundleStatsNode } from
'../../services/profiling/bundleAnalyzer.ts';
import { ChartBarIcon, SparklesIcon } from '../icons.tsx';
import { LoadingSpinner, MarkdownRenderer } from '../shared/index.tsx';

const FlameChart: React.FC<{ trace: TraceEntry[] }> = ({ trace }) => {
    if (trace.length === 0) return <p className="text-text-secondary">No trace data
    collected.</p>;
    const maxTime = Math.max(...trace.map(t => t.startTime + t.duration));
    return (
        <div className="space-y-1 font-mono text-xs">
```

```
            {trace.filter(t => t.entryType === 'measure').map((entry, i) => (
                <div key={i} className="group relative h-6 bg-primary/20 rounded">
                    <div className="h-full bg-primary" style={{ marginLeft: `${(entry.startTime /
                    maxTime) * 100}%`, width: `${(entry.duration / maxTime) * 100}%` }}></div>
                    <div className="absolute inset-0 px-2 flex items-center text-primary font-
                    bold">{entry.name} ({entry.duration.toFixed(1)}ms)</div>
                </div>
            ))}
        </div>
    );
};

export const PerformanceProfiler: React.FC = () => {
    const [activeTab, setActiveTab] = useState<'runtime' | 'bundle'>('runtime');
    const [isTracing, setIsTracing] = useState(false);
    const [trace, setTrace] = useState<TraceEntry[]>([]);
    const [bundleStats, setBundleStats] = useState<string>('');
    const [bundleTree, setBundleTree] = useState<BundleStatsNode | null>(null);
    const [isLoadingAi, setIsLoadingAi] = useState(false);
    const [aiAnalysis, setAiAnalysis] = useState('');

    const handleTraceToggle = () => {
        if (isTracing) {
            const collectedTrace = stopTracing();
            setTrace(collectedTrace);
            setIsTracing(false);
        } else {
            setTrace([]);
            startTracing();
            setIsTracing(true);
        }
    };

    const handleAnalyzeBundle = () => {
        try {
            setBundleTree(parseViteStats(bundleStats));
        } catch (e) {
            alert(e instanceof Error ? e.message : 'Parsing failed.');
        }
    };

    const handleAiAnalysis = async () => {
        const dataToAnalyze = activeTab === 'runtime' ? trace : bundleTree;
        if (!dataToAnalyze || (Array.isArray(dataToAnalyze) && dataToAnalyze.length ===
        0)) {
            alert('No data to analyze.');
            return;
        }
        setIsLoadingAi(true);
        setAiAnalysis('');
        try {
            const analysis = await analyzePerformanceTrace(dataToAnalyze);
            setAiAnalysis(analysis);
        } catch (e) {
            setAiAnalysis('Error getting analysis from AI.');
        } finally {
            setIsLoadingAi(false);
        }
    };

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
```

```
            <header className="mb-6"><h1 className="text-3xl font-bold flex items-
            center"><ChartBarIcon /><span className="ml-3">AI Performance
            Profiler</span></h1><p className="text-text-secondary mt-1">Analyze runtime
            performance and bundle sizes with AI insights.</p></header>
            <div className="flex border-b border-border mb-4"><button onClick={() =>
            setActiveTab('runtime')} className={`px-4 py-2 text-sm ${activeTab === 'runtime'
            ? 'border-b-2 border-primary' : ''}`}>Runtime Performance</button><button
            onClick={() => setActiveTab('bundle')} className={`px-4 py-2 text-sm ${activeTab
            === 'bundle' ? 'border-b-2 border-primary' : ''}`}>Bundle
            Analysis</button></div>
            <div className="flex-grow grid grid-cols-1 lg:grid-cols-2 gap-6 min-h-0">
                <div className="bg-surface p-4 border border-border rounded-lg flex flex-col">
                    {activeTab === 'runtime' ? (
                        <>
                            <button onClick={handleTraceToggle} className="btn-primary mb-4 py-2">{isTracing
                            ? 'Stop Tracing' : 'Start Tracing'}</button>
                            <div className="flex-grow overflow-y-auto"><FlameChart trace={trace} /></div>
                        </>
                    ) : (
                         <>
                            <textarea value={bundleStats} onChange={e => setBundleStats(e.target.value)}
                            placeholder="Paste your stats.json content here" className="w-full h-48 p-2 bg-
                            background border rounded font-mono text-xs mb-2"/>
                            <button onClick={handleAnalyzeBundle} className="btn-primary py-2">Analyze
                            Bundle</button>
                            <div className="flex-grow overflow-y-auto mt-2">
                                <pre className="text-xs">{bundleTree ? JSON.stringify(bundleTree, null, 2) :
                                'Analysis will appear here.'}</pre>
                            </div>
                        </>
                    )}
                </div>
                 <div className="bg-surface p-4 border border-border rounded-lg flex flex-col">
                    <button onClick={handleAiAnalysis} disabled={isLoadingAi} className="btn-primary
                    flex items-center justify-center gap-2 py-2 mb-4"><SparklesIcon />{isLoadingAi ?
                    'Analyzing...' : 'Get AI Optimization Suggestions'}</button>
                    <div className="flex-grow bg-background border border-border rounded p-2
                    overflow-y-auto">
                        {isLoadingAi ? <div className="flex justify-center items-center
                        h-full"><LoadingSpinner/></div> : <MarkdownRenderer content={aiAnalysis} />}
                    </div>
                 </div>
            </div>
        </div>
    );
};


// ===== AccessibilityAuditor.tsx =====

import React, { useState, useRef } from 'react';
import { suggestA11yFix } from '../../services/geminiService.ts';
import { runAxeAudit, AxeResult } from
'../../services/auditing/accessibilityService.ts';
import { EyeIcon, SparklesIcon } from '../icons.tsx';
import { LoadingSpinner, MarkdownRenderer } from '../shared/index.tsx';

export const AccessibilityAuditor: React.FC = () => {
    const [url, setUrl] = useState('https://react.dev');
    const [auditUrl, setAuditUrl] = useState('');
    const [results, setResults] = useState<AxeResult | null>(null);
    const [isLoading, setIsLoading] = useState(false);
```

```
const [isLoadingAi, setIsLoadingAi] = useState<string | null>(null);
const [aiFixes, setAiFixes] = useState<Record<string, string>>({});
const iframeRef = useRef<HTMLIFrameElement>(null);

const handleAudit = () => {
    const targetUrl = url.startsWith('http') ? url : `https://${url}`;
    setAuditUrl(targetUrl);
    setIsLoading(true);
    setResults(null);
    setAiFixes({});
};

const handleIframeLoad = async () => {
    if (isLoading && iframeRef.current) {
        try {
            const auditResults = await
            runAxeAudit(iframeRef.current.contentWindow!.document);
            setResults(auditResults);
        } catch (error) {
            console.error(error);
            alert('Could not audit this page. This may be due to security restrictions
            (CORS).');
        } finally {
            setIsLoading(false);
        }
    }
};

const handleGetFix = async (issue: any) => {
    const issueId = issue.id;
    setIsLoadingAi(issueId);
    try {
        const fix = await suggestA11yFix(issue);
        setAiFixes(prev => ({...prev, [issueId]: fix}));
    } catch(e) {
        setAiFixes(prev => ({...prev, [issueId]: 'Could not get suggestion.'}));
    } finally {
        setIsLoadingAi(null);
    }
};

return (
    <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
        <header className="mb-6"><h1 className="text-3xl font-bold flex items-
        center"><EyeIcon /><span className="ml-3">Automated Accessibility
        Auditor</span></h1><p className="text-text-secondary mt-1">Audit a live URL for
        accessibility issues and get AI-powered fixes.</p></header>
        <div className="flex gap-2 mb-4"><input type="text" value={url} onChange={e =>
        setUrl(e.target.value)} placeholder="https://example.com" className="flex-grow
        p-2 border rounded"/><button onClick={handleAudit} disabled={isLoading}
        className="btn-primary px-6 py-2">{isLoading ? 'Auditing...' :
        'Audit'}</button></div>
        <div className="flex-grow grid grid-cols-1 lg:grid-cols-2 gap-6 min-h-0">
            <div className="bg-background border-2 border-dashed border-border rounded-lg
            overflow-hidden"><iframe ref={iframeRef} src={auditUrl} title="Audit Target"
            className="w-full h-full bg-white" onLoad={handleIframeLoad} sandbox="allow-
            scripts allow-same-origin"/></div>
            <div className="bg-surface p-4 border border-border rounded-lg flex flex-col">
                <h3 className="text-lg font-bold mb-2">Audit Results</h3>
                <div className="flex-grow overflow-y-auto pr-2">
                    {isLoading && <div className="flex justify-center items-center
                    h-full"><LoadingSpinner/></div>}
```

```
                        {results && (results.violations.length === 0 ? <p>No violations found!</p> :
                            results.violations.map((v, i) => (
                                <div key={v.id + i} className="p-3 mb-2 bg-background border border-border
                                rounded">
                                    <p className="font-bold text-red-600">{v.help}</p>
                                    <p className="text-sm my-1">{v.description}</p>
                                    <button onClick={() => handleGetFix(v)} disabled={!!isLoadingAi}
                                    className="text-xs flex items-center gap-1 text-primary font-
                                    semibold"><SparklesIcon/> {isLoadingAi === v.id ? 'Getting fix...' : 'Ask AI for
                                    a fix'}</button>
                                    {aiFixes[v.id] && <div className="mt-2 text-xs border-t pt-2"><MarkdownRenderer
                                    content={aiFixes[v.id]}/></div>}
                                </div>
                            ))
                        )}
                    </div>
                </div>
            </div>
        </div>
    );
};


// ===== CiCdPipelineGenerator.tsx =====

import React, { useState } from 'react';
import { generateCiCdConfig } from '../../services/geminiService.ts';
import { PaperAirplaneIcon, SparklesIcon } from '../icons.tsx';
import { LoadingSpinner, MarkdownRenderer } from '../shared/index.tsx';

const platforms = ['GitHub Actions', 'GitLab CI', 'CircleCI', 'Jenkins'];
const exampleDescription = "Install Node.js dependencies, run linting and tests,
build the production app, and then deploy to Vercel.";

export const CiCdPipelineGenerator: React.FC = () => {
    const [platform, setPlatform] = useState(platforms[0]);
    const [description, setDescription] = useState(exampleDescription);
    const [config, setConfig] = useState('');
    const [isLoading, setIsLoading] = useState(false);
    const [error, setError] = useState('');

    const handleGenerate = async () => {
        if (!description.trim()) {
            setError('Please provide a description of the pipeline stages.');
            return;
        }
        setIsLoading(true);
        setError('');
        try {
            const result = await generateCiCdConfig(platform, description);
            setConfig(result);
        } catch (err) {
            setError(err instanceof Error ? err.message : 'Failed to generate config.');
        } finally {
            setIsLoading(false);
        }
    };

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
            <header className="mb-6">
                <h1 className="text-3xl font-bold flex items-center"><PaperAirplaneIcon /><span
```

```
                className="ml-3">AI CI/CD Pipeline Architect</span></h1>
                <p className="text-text-secondary mt-1">Describe your deployment process and get
                a modern configuration file.</p>
            </header>
            <div className="flex-grow flex flex-col gap-4 min-h-0">
                 <div className="flex flex-col flex-1 min-h-0">
                    <div className="grid grid-cols-1 md:grid-cols-3 gap-4 mb-4">
                        <div><label className="block text-sm">Platform</label><select value={platform}
                        onChange={e => setPlatform(e.target.value)} className="w-full mt-1 p-2 bg-
                        surface border rounded"><option>GitHub Actions</option><option>GitLab
                        CI</option><option>CircleCI</option></select></div>
                        <div className="md:col-span-2"><label className="block text-sm">Describe
                        Stages</label><input type="text" value={description} onChange={e =>
                        setDescription(e.target.value)} className="w-full mt-1 p-2 bg-surface border
                        rounded"/></div>
                    </div>
                     <button onClick={handleGenerate} disabled={isLoading} className="btn-primary
                     w-full max-w-xs mx-auto flex items-center justify-center py-2"><SparklesIcon />
                     {isLoading ? 'Generating...' : 'Generate Configuration'}</button>
                </div>
                 <div className="flex flex-col flex-grow min-h-0">
                    <label className="text-sm font-medium text-text-secondary mb-2">Generated
                    Configuration File</label>
                    <div className="relative flex-grow p-1 bg-background border border-border
                    rounded-md overflow-y-auto">
                        {isLoading && !config && <div className="flex items-center justify-center
                        h-full"><LoadingSpinner /></div>}
                        {error && <p className="p-4 text-red-500">{error}</p>}
                        {config && <MarkdownRenderer content={config} />}
                         {!isLoading && !config && !error && <div className="text-text-secondary h-full
                         flex items-center justify-center">Generated config will appear here.</div>}
                    </div>
                </div>
            </div>
        </div>
    );
};


// ===== DeploymentPreview.tsx =====

import React, { useState, useEffect, useRef } from 'react';
import { getAllFiles, getFileByPath } from '../../services/dbService.ts';
import type { GeneratedFile } from '../../types.ts';
import { CloudIcon } from '../icons.tsx';
import { LoadingSpinner } from '../shared/index.tsx';

export const DeploymentPreview: React.FC = () => {
    const [files, setFiles] = useState<GeneratedFile[]>([]);
    const [isLoading, setIsLoading] = useState(true);
    const [error, setError] = useState('');
    const iframeRef = useRef<HTMLIFrameElement>(null);

    useEffect(() => {
        const loadAndRender = async () => {
            setIsLoading(true);
            setError('');
            try {
                const allFiles = await getAllFiles();
                if (allFiles.length === 0) {
                    setError('No files generated by AI Feature Builder found.');
                    setFiles([]);
```

```
                setIsLoading(false);
                return;
            }
            setFiles(allFiles);

            let indexHtmlFile = allFiles.find(f => f.filePath.endsWith('index.html'));
            if (!indexHtmlFile) {
                setError('No index.html file found in the generated files.');
                setIsLoading(false);
                return;
            }

            let content = indexHtmlFile.content;

            // Create blob URLs for all assets and replace relative paths
            const blobUrlMap = new Map<string, string>();
            for (const file of allFiles) {
                const mimeType = file.filePath.endsWith('.css') ? 'text/css' :
                'application/javascript';
                const blob = new Blob([file.content], { type: mimeType });
                blobUrlMap.set(file.filePath, URL.createObjectURL(blob));
            }

            // Replace relative paths in index.html
            content = content.replace(/(href|src)=["'](\.?\/)?([^"']+)["']/g, (match, attr,
            prefix, path) => {
                const blobUrl = blobUrlMap.get(path);
                return blobUrl ? `${attr}="${blobUrl}"` : match;
            });

            if (iframeRef.current) {
                iframeRef.current.srcdoc = content;
            }

        } catch (err) {
            setError(err instanceof Error ? err.message : 'Failed to load files.');
        } finally {
            setIsLoading(false);
        }
    };

    loadAndRender();

    // Cleanup blob URLs on unmount
    return () => {
         // This is a bit tricky since we don't have the map here, but the browser will
         clean them up.
    };
}, []);

return (
    <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
        <header className="mb-6">
            <h1 className="text-3xl font-bold flex items-center"><CloudIcon /><span
            className="ml-3">Static Deployment Previewer</span></h1>
            <p className="text-text-secondary mt-1">Live preview of the static site
            generated by the AI Feature Builder.</p>
        </header>
        <div className="flex-grow grid grid-cols-1 lg:grid-cols-3 gap-6 min-h-0">
            <div className="lg:col-span-1 bg-surface p-4 border border-border rounded-lg
            overflow-y-auto">
                <h3 className="font-bold mb-2">File List</h3>
```

```
                    <ul className="text-sm space-y-1">
                        {files.map(f => <li key={f.filePath} className="truncate p-1 bg-background
                        rounded">{f.filePath}</li>)}
                    </ul>
                </div>
                <div className="lg:col-span-2 bg-background border-2 border-dashed border-border
                rounded-lg overflow-hidden">
                    {isLoading && <div className="flex justify-center items-center
                    h-full"><LoadingSpinner/></div>}
                    {error && <div className="flex justify-center items-center h-full text-
                    red-500">{error}</div>}
                    {!isLoading && !error && <iframe ref={iframeRef} title="Deployment Preview"
                    className="w-full h-full bg-white"/>}
                </div>
            </div>
        </div>
    );
};


// ===== SecurityScanner.tsx =====

import React, { useState } from 'react';
import { analyzeCodeForVulnerabilities } from '../../services/geminiService.ts';
import { runStaticScan, SecurityIssue } from
'../../services/security/staticAnalysisService.ts';
import { ShieldCheckIcon, SparklesIcon } from '../icons.tsx';
import { LoadingSpinner } from '../shared/index.tsx';

const exampleCode = `function UserProfile({ user }) {
  // TODO: remove this temporary api key
  const API_KEY = "sk-xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx";
  const userContent = user.bio; // This might contain malicious scripts

  return (
    <div>
      <h2>{user.name}</h2>
      <div dangerouslySetInnerHTML={{ __html: userContent }} />
    </div>
  );
}`;

export const SecurityScanner: React.FC = () => {
    const [code, setCode] = useState(exampleCode);
    const [localIssues, setLocalIssues] = useState<SecurityIssue[]>([]);
    const [aiIssues, setAiIssues] = useState<any[]>([]);
    const [isLoading, setIsLoading] = useState(false);
    const [error, setError] = useState('');

    const handleScan = async () => {
        if (!code.trim()) {
            setError('Please enter code to scan.');
            return;
        }
        setIsLoading(true);
        setError('');
        setLocalIssues([]);
        setAiIssues([]);
        try {
            // Run local scan first
            const staticIssues = runStaticScan(code);
            setLocalIssues(staticIssues);
```

```
        // Then run AI scan
        const geminiIssues = await analyzeCodeForVulnerabilities(code);
        setAiIssues(geminiIssues);

    } catch (err) {
        setError(err instanceof Error ? err.message : 'An error occurred during
        scanning.');
    } finally {
        setIsLoading(false);
    }
};

const SeverityBadge: React.FC<{ severity: string }> = ({ severity }) => {
    const colors: Record<string, string> = {
        'Critical': 'bg-red-500 text-white',
        'High': 'bg-red-400 text-white',
        'Medium': 'bg-yellow-400 text-yellow-900',
        'Low': 'bg-blue-400 text-white',
        'Informational': 'bg-gray-400 text-gray-900',
    };
    return <span className={`px-2 py-0.5 text-xs font-bold rounded-full
    ${colors[severity] || 'bg-gray-300'}`}>{severity}</span>
}

return (
    <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
        <header className="mb-6">
            <h1 className="text-3xl font-bold flex items-center"><ShieldCheckIcon /><span
            className="ml-3">AI Security Co-Pilot</span></h1>
            <p className="text-text-secondary mt-1">Find vulnerabilities in your code with
            static analysis and AI.</p>
        </header>
        <div className="flex-grow grid grid-cols-1 lg:grid-cols-2 gap-6 min-h-0">
            <div className="flex flex-col">
                <label className="text-sm mb-2">Code to Scan</label>
                <textarea value={code} onChange={e => setCode(e.target.value)} className="w-full
                flex-grow p-2 bg-surface border rounded font-mono text-xs" />
                <button onClick={handleScan} disabled={isLoading} className="btn-primary w-full
                mt-4 py-2 flex justify-center items-center gap-2">{isLoading ? <LoadingSpinner/>
                : 'Scan Code'}</button>
            </div>
            <div className="flex flex-col bg-surface p-4 border rounded-lg">
                <h3 className="text-lg font-bold mb-2">Scan Results</h3>
                {error && <p className="text-red-500">{error}</p>}
                <div className="flex-grow overflow-y-auto pr-2 space-y-4">
                    {isLoading && <div className="flex justify-center items-center
                    h-full"><LoadingSpinner/></div>}
                    {!isLoading && localIssues.length === 0 && aiIssues.length === 0 && <p
                    className="text-text-secondary text-center mt-8">No issues found. Run a scan to
                    begin.</p>}

                    {localIssues.length > 0 && <div>
                        <h4 className="font-semibold text-sm mb-1">Static Analysis Findings</h4>
                        {localIssues.map((issue, i) => <div key={`local-${i}`} className="p-2 bg-
                        background border rounded mb-2"><p className="font-bold flex items-center
                        gap-2">{issue.type} <SeverityBadge severity={issue.severity} /></p><p
                        className="text-xs">Line {issue.line}: {issue.description}</p></div>)}
                    </div>}

                     {aiIssues.length > 0 && <div>
                        <h4 className="font-semibold text-sm mb-1 flex items-center
                        gap-1"><SparklesIcon/> AI-Powered Findings</h4>
```

```
                            {aiIssues.map((issue, i) => <div key={`ai-${i}`} className="p-2 bg-background
                            border rounded mb-2"><p className="font-bold flex items-center
                            gap-2">{issue.vulnerability} <SeverityBadge severity={issue.severity} /></p><p
                            className="text-xs mt-1"><strong>Description:</strong> {issue.description}</p><p
                            className="text-xs mt-1"><strong>Mitigation:</strong>
                            {issue.mitigation}</p></div>)}
                        </div>}
                    </div>
                </div>
            </div>
        );
};


// ===== ThemeDesigner_15.tsx =====

import React, { useState, useCallback, useEffect } from 'react';
import { SparklesIcon, ArrowDownTrayIcon, PhotoIcon } from '../icons.tsx';
import { generateSemanticTheme } from '../../services/geminiService.ts';
import { fileToBase64 } from '../../services/fileUtils.ts';
import type { SemanticColorTheme, ColorTheme } from '../../types.ts';
import { LoadingSpinner } from '../shared/index.tsx';
import { useTheme } from '../../hooks/useTheme.ts';

const ColorDisplay: React.FC<{ name: string; color: { name: string; value:
string; } }> = ({ name, color }) => (
    <div className="flex items-center justify-between p-2 bg-background rounded-md
    border border-border">
        <div className="flex items-center gap-3">
            <div className="w-6 h-6 rounded-full border border-border" style={{
            backgroundColor: color.value }} />
            <div>
                <p className="text-sm font-semibold text-text-primary capitalize">{name}</p>
                <p className="text-xs text-text-secondary">{color.name}</p>
            </div>
        </div>
        <span className="font-mono text-sm text-text-secondary">{color.value}</span>
    </div>
);

const AccessibilityCheck: React.FC<{ name: string, check: { ratio: number;
score: string; } }> = ({ name, check }) => {
    const scoreColor = check.score === 'AAA' ? 'text-green-600' : check.score ===
    'AA' ? 'text-emerald-600' : 'text-red-600';
    return (
        <div className="flex items-center justify-between p-2 bg-background rounded-md
        border border-border text-sm">
            <p className="text-text-secondary">{name}</p>
            <div className="flex items-center gap-2">
                <span className="font-mono">{check.ratio.toFixed(2)}</span>
                <span className={`font-bold px-2 py-0.5 rounded-full text-xs ${scoreColor}
                ${scoreColor.replace('text-', 'bg-')}/10`}>{check.score}</span>
            </div>
        </div>
    );
}

export const ThemeDesigner: React.FC = () => {
    const [theme, setTheme] = useState<SemanticColorTheme | null>(null);
    const [prompt, setPrompt] = useState('A calming, minimalist theme for a blog');
    const [image, setImage] = useState<{ base64: string, name: string } |
```

```
null>(null);
const [isLoading, setIsLoading] = useState(false);
const [error, setError] = useState('');
const [, , applyCustomTheme] = useTheme();

const handleGenerate = useCallback(async () => {
    const textPart = { text: `Generate a theme based on this description:
    "${prompt}"` };
    const imagePart = image ? { inlineData: { mimeType: 'image/png', data:
    image.base64 } } : null;
    const parts = imagePart ? [textPart, imagePart] : [textPart];

    setIsLoading(true); setError('');
    try {
        const newTheme = await generateSemanticTheme({ parts });
        setTheme(newTheme);
    } catch (err) {
        setError(err instanceof Error ? err.message : "An unknown error occurred.");
    } finally {
        setIsLoading(false);
    }
}, [prompt, image]);

const handleFileChange = async (e: React.ChangeEvent<HTMLInputElement>) => {
    const file = e.target.files?.[0];
    if (file) {
        const base64 = await fileToBase64(file);
        setImage({ base64, name: file.name });
        setPrompt(`A theme based on the uploaded image: ${file.name}`);
    }
};

useEffect(() => { handleGenerate(); }, []);

const handleApplyTheme = () => {
    if (!theme) return;
    const colorsToApply: ColorTheme = {
        primary: theme.palette.primary.value,
        background: theme.theme.background.value,
        surface: theme.theme.surface.value,
        textPrimary: theme.theme.textPrimary.value,
        textSecondary: theme.theme.textSecondary.value,
        textOnPrimary: theme.theme.textOnPrimary.value,
        border: theme.theme.border.value,
    };
    applyCustomTheme(colorsToApply, theme.mode);
};

return (
    <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
        <header className="mb-6">
            <h1 className="text-3xl font-bold flex items-center"><SparklesIcon /><span
            className="ml-3">AI Theme Designer</span></h1>
            <p className="text-text-secondary mt-1">Generate a full design system from a
            description or image.</p>
        </header>
        <div className="flex-grow grid grid-cols-1 md:grid-cols-2 gap-6 min-h-0">
            <div className="md:col-span-1 flex flex-col gap-4 bg-surface border border-
            border p-6 rounded-lg overflow-y-auto">
                <h3 className="text-xl font-bold">Describe or Upload</h3>
                <textarea value={prompt} onChange={e => setPrompt(e.target.value)}
                className="p-2 bg-background border border-border rounded-md resize-none text-sm
```

```
    h-24" placeholder="e.g., A light, airy theme for a blog" />
     <div className="relative border border-dashed border-border rounded-lg p-4 text-
     center">
        <input type="file" onChange={handleFileChange} className="absolute inset-0
        w-full h-full opacity-0 cursor-pointer" />
        <PhotoIcon/>
        <p className="text-sm mt-1">{image ? `Image: ${image.name}` : 'Upload an image
        (optional)'}</p>
    </div>
    <div className="flex gap-2">
        <button onClick={handleGenerate} disabled={isLoading} className="btn-primary
        flex-grow flex items-center justify-center gap-2 px-4 py-2">
            {isLoading ? <LoadingSpinner /> : 'Generate New Theme'}
        </button>
         <button onClick={handleApplyTheme} disabled={isLoading || !theme}
         className="px-4 py-2 bg-emerald-600 text-white font-bold rounded-md
         hover:opacity-90 transition-all disabled:opacity-50 shadow-md">
            Apply to App
        </button>
    </div>
    {error && <p className="text-red-500 text-xs text-center">{error}</p>}

    {theme && !isLoading && (
        <div className="mt-4 border-t border-border pt-4 space-y-4">
            <div><h3 className="text-lg font-bold mb-2">Palette</h3><div
            className="space-y-2"><ColorDisplay name="Primary"
            color={theme.palette.primary}/><ColorDisplay name="Secondary"
            color={theme.palette.secondary}/><ColorDisplay name="Accent"
            color={theme.palette.accent}/><ColorDisplay name="Neutral"
            color={theme.palette.neutral}/></div></div>
            <div><h3 className="text-lg font-bold mb-2">Theme Roles</h3><div
            className="space-y-2"><ColorDisplay name="Background"
            color={theme.theme.background}/><ColorDisplay name="Surface"
            color={theme.theme.surface}/><ColorDisplay name="Text Primary"
            color={theme.theme.textPrimary}/><ColorDisplay name="Text Secondary"
            color={theme.theme.textSecondary}/><ColorDisplay name="Text on Primary"
            color={theme.theme.textOnPrimary}/><ColorDisplay name="Border"
            color={theme.theme.border}/></div></div>
            <div><h3 className="text-lg font-bold mb-2">Accessibility (WCAG 2.1)</h3><div
            className="space-y-2"><AccessibilityCheck name="Primary on Surface"
            check={theme.accessibility.primaryOnSurface}/><AccessibilityCheck name="Text on
            Surface" check={theme.accessibility.textPrimaryOnSurface}/><AccessibilityCheck
            name="Subtle Text on Surface"
            check={theme.accessibility.textSecondaryOnSurface}/><AccessibilityCheck
            name="Text on Primary"
            check={theme.accessibility.textOnPrimaryOnPrimary}/></div></div>
        </div>
    )}
</div>
<div className="md:col-span-1 rounded-lg p-8 overflow-y-auto border border-
border" style={{ backgroundColor: theme?.theme.background.value, color:
theme?.theme.textPrimary.value }}>
     <h3 className="text-2xl font-bold mb-6">Live Preview</h3>
     {theme ? (
         <div className="p-6 rounded-lg grid grid-cols-1 md:grid-cols-2 gap-6" style={{
         backgroundColor: theme.theme.surface.value }}>
            <div className="space-y-4">
                <h4 className="text-lg font-bold">Sample Card</h4>
                <p className="text-sm" style={{color: theme.theme.textSecondary.value}}>This is
                a sample card to demonstrate the theme colors. It contains a primary button and
                some secondary text.</p>
                <button className="px-4 py-2 rounded-md font-bold transition-colors" style={{
```

```
                                backgroundColor: theme.palette.primary.value, color:
                                theme.theme.textOnPrimary.value }}>Primary Button</button>
                            </div>
                             <div className="space-y-4">
                                <input type="text" placeholder="Text input" className="w-full px-3 py-2 rounded-
                                md border" style={{backgroundColor: theme.theme.background.value, borderColor:
                                theme.theme.border.value, color: theme.theme.textPrimary.value}} />
                                <div className="p-3 border rounded" style={{borderColor:
                                theme.theme.border.value, color: theme.theme.textSecondary.value}}>
                                    <p>A bordered container.</p>
                                </div>
                             </div>
                        </div>
                    ) : <div className="flex items-center justify-center h-full text-text-
                    secondary">Theme preview will appear here.</div>}
                </div>
            </div>
        </div>
    );
};

// ===== CodeReviewBot_15.tsx =====

import React, { useState, useCallback } from 'react';
import { reviewCodeStream } from '../../services/index.ts';
import { useAiPersonalities } from '../../hooks/useAiPersonalities.ts';
import { formatSystemPromptToString } from '../../utils/promptUtils.ts';
import { CpuChipIcon } from '../icons.tsx';
import { LoadingSpinner } from '../shared/index.tsx';
import { MarkdownRenderer } from '../shared/index.tsx';

const exampleCode = `function UserList(users) {
  if (users.length = 0) {
    return "no users";
  } else {
    return (
      users.map(u => {
        return <li>{u.name}</li>
      })
    )
  }
}`;

export const CodeReviewBot: React.FC = () => {
    const [code, setCode] = useState<string>(exampleCode);
    const [review, setReview] = useState<string>('');
    const [isLoading, setIsLoading] = useState<boolean>(false);
    const [error, setError] = useState<string>('');
    const [personalities] = useAiPersonalities();
    const [selectedPersonalityId, setSelectedPersonalityId] =
    useState<string>('default');

    const handleGenerate = useCallback(async () => {
        if (!code.trim()) {
            setError('Please enter some code to review.');
            return;
        }
        setIsLoading(true);
        setError('');
        setReview('');

        let systemInstruction: string | undefined = undefined;
```

```
    if (selectedPersonalityId !== 'default') {
        const personality = personalities.find(p => p.id === selectedPersonalityId);
        if (personality) {
            systemInstruction = formatSystemPromptToString(personality);
        }
    }

    try {
        const stream = reviewCodeStream(code, systemInstruction);
        let fullResponse = '';
        for await (const chunk of stream) {
            fullResponse += chunk;
            setReview(fullResponse);
        }
    } catch (err) {
        const errorMessage = err instanceof Error ? err.message : 'An unknown error
        occurred.';
        setError(`Failed to get review: ${errorMessage}`);
    } finally {
        setIsLoading(false);
    }
}, [code, selectedPersonalityId, personalities]);

return (
    <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
        <header className="mb-6">
            <h1 className="text-3xl font-bold flex items-center">
                <CpuChipIcon />
                <span className="ml-3">AI Code Review Bot</span>
            </h1>
            <p className="text-text-secondary mt-1">Get an automated code review from
            Gemini.</p>
        </header>
        <div className="flex-grow flex flex-col gap-4 min-h-0">
            <div className="flex flex-col flex-1 min-h-0">
                <label htmlFor="code-input" className="text-sm font-medium text-text-secondary
                mb-2">Code to Review</label>
                <textarea
                    id="code-input"
                    value={code}
                    onChange={(e) => setCode(e.target.value)}
                    placeholder="Paste your code here..."
                    className="flex-grow p-4 bg-surface border border-border rounded-md resize-none
                    font-mono text-sm"
                />
            </div>
             <div className="flex-shrink-0 flex items-center justify-center gap-4">
                 <div className="w-full max-w-xs">
                    <label htmlFor="personality-select" className="text-sm font-medium text-text-
                    secondary">Reviewer Personality</label>
                    <select
                        id="personality-select"
                        value={selectedPersonalityId}
                        onChange={e => setSelectedPersonalityId(e.target.value)}
                        className="w-full mt-1 p-2 bg-surface border border-border rounded-md text-sm"
                    >
                        <option value="default">Default</option>
                        {personalities.map(p => (
                            <option key={p.id} value={p.id}>{p.name}</option>
                        ))}
                    </select>
                 </div>
```

```
                    <button
                        onClick={handleGenerate}
                        disabled={isLoading}
                        className="btn-primary self-end h-[42px] w-full max-w-xs flex items-center
                        justify-center px-6 py-3"
                    >
                        {isLoading ? <LoadingSpinner /> : 'Request Review'}
                    </button>
                </div>
                <div className="flex flex-col flex-1 min-h-0">
                    <label className="text-sm font-medium text-text-secondary mb-2">AI
                    Feedback</label>
                    <div className="flex-grow p-4 bg-background border border-border rounded-md
                    overflow-y-auto">
                        {isLoading && !review && <div className="flex items-center justify-center
                        h-full"><LoadingSpinner /></div>}
                        {error && <p className="text-red-500">{error}</p>}
                        {review && <MarkdownRenderer content={review} />}
                         {!isLoading && !review && !error && <div className="text-text-secondary h-full
                         flex items-center justify-center">Review will appear here.</div>}
                    </div>
                </div>
            </div>
        </div>
    );
};


// ===== TerraformGenerator.tsx =====

import React, { useState, useCallback } from 'react';
import { generateTerraformConfig } from '../../services/geminiService.ts';
import * as vaultService from '../../services/vaultService.ts';
import { useVaultModal } from '../../contexts/VaultModalContext.tsx';
import { useGlobalState } from '../../contexts/GlobalStateContext.tsx';
import { CpuChipIcon, SparklesIcon } from '../icons.tsx';
import { LoadingSpinner, MarkdownRenderer } from '../shared/index.tsx';

export const TerraformGenerator: React.FC = () => {
    const [description, setDescription] = useState('An S3 bucket for static website
    hosting');
    const [cloud, setCloud] = useState<'aws' | 'gcp'>('aws');
    const [config, setConfig] = useState('');
    const [isLoading, setIsLoading] = useState(false);
    const [error, setError] = useState('');
    const { state } = useGlobalState();
    const { requestUnlock } = useVaultModal();

    const handleGenerate = useCallback(async () => {
        if (!description.trim()) {
            setError('Please provide a description.');
            return;
        }
        setIsLoading(true);
        setError('');
        setConfig('');
        try {
            let context = '';
            if (cloud === 'aws' && state.vaultState.isUnlocked) {
                 const awsKey = await vaultService.getDecryptedCredential('aws_key');
                 const awsSecret = await vaultService.getDecryptedCredential('aws_secret');
                 if(awsKey && awsSecret) {
```

```
                context = 'Using stored AWS credentials to analyze environment.';
                // In a real app, you would make AWS SDK calls here to list S3 buckets, VPCs,
                etc.
                // For this demo, we'll just pass a context message.
             }
        } else if (cloud === 'aws' && !state.vaultState.isUnlocked) {
            const unlocked = await requestUnlock();
            if(!unlocked) {
                setError('Vault must be unlocked to use AWS context.');
                setIsLoading(false);
                return;
            }
        }

        const result = await generateTerraformConfig(cloud, description, context);
        setConfig(result);
    } catch (err) {
        setError(err instanceof Error ? err.message : 'Failed to generate config.');
    } finally {
        setIsLoading(false);
    }
}, [description, cloud, state.vaultState.isUnlocked, requestUnlock]);

return (
    <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
        <header className="mb-6">
            <h1 className="text-3xl font-bold flex items-center"><CpuChipIcon /><span
            className="ml-3">AI Terraform Generator</span></h1>
            <p className="text-text-secondary mt-1">Generate infrastructure-as-code from a
            description, with context from your cloud provider.</p>
        </header>
        <div className="flex-grow flex flex-col gap-4 min-h-0">
             <div className="flex flex-col flex-1 min-h-0">
                 <div className="grid grid-cols-1 md:grid-cols-3 gap-4 mb-4">
                    <div>
                        <label className="block text-sm">Cloud Provider</label>
                        <select value={cloud} onChange={e => setCloud(e.target.value as 'aws' | 'gcp')}
                        className="w-full mt-1 p-2 bg-surface border rounded">
                            <option value="aws">AWS</option>
                            <option value="gcp" disabled>GCP (coming soon)</option>
                        </select>
                    </div>
                    <div className="md:col-span-2">
                        <label className="block text-sm">Describe the infrastructure</label>
                        <input type="text" value={description} onChange={e =>
                        setDescription(e.target.value)} className="w-full mt-1 p-2 bg-surface border
                        rounded"/>
                    </div>
                </div>
                 <button onClick={handleGenerate} disabled={isLoading} className="btn-primary
                 w-full max-w-xs mx-auto flex items-center justify-center py-2"><SparklesIcon />
                 {isLoading ? 'Generating...' : 'Generate Configuration'}</button>
            </div>
             <div className="flex flex-col flex-grow min-h-0">
                <label className="text-sm font-medium text-text-secondary mb-2">Generated
                Terraform (.tf)</label>
                <div className="relative flex-grow p-1 bg-background border border-border
                rounded-md overflow-y-auto">
                    {isLoading && !config && <div className="flex items-center justify-center
                    h-full"><LoadingSpinner /></div>}
                    {error && <p className="p-4 text-red-500">{error}</p>}
                    {config && <MarkdownRenderer content={config} />}
```

```
                        {!isLoading && !config && !error && <div className="text-text-secondary h-full
                        flex items-center justify-center">Generated config will appear here.</div>}
                    </div>
                </div>
            </div>
        </div>
    );
};


// ===== AiPersonalityForge.tsx =====

import React, { useState, useEffect, useRef } from 'react';
import { SparklesIcon, PlusIcon, TrashIcon, ArrowDownTrayIcon,
ArrowUpOnSquareIcon } from '../icons.tsx';
import { useAiPersonalities } from '../../hooks/useAiPersonalities.ts';
import { formatSystemPromptToString } from '../../utils/promptUtils.ts';
import { streamContent } from '../../services/geminiService.ts';
import { downloadJson } from '../../services/fileUtils.ts';
import type { SystemPrompt } from '../../types.ts';
import { LoadingSpinner, MarkdownRenderer } from '../shared/index.tsx';
import { useNotification } from '../../contexts/NotificationContext.tsx';

const defaultNewPrompt: Omit<SystemPrompt, 'id' | 'name'> = {
    persona: 'You are a helpful assistant.',
    rules: [],
    outputFormat: 'markdown',
    exampleIO: [],
};

export const AiPersonalityForge: React.FC = () => {
    const [personalities, setPersonalities] = useAiPersonalities();
    const [activeId, setActiveId] = useState<string | null>(null);
    const { addNotification } = useNotification();
    const fileInputRef = useRef<HTMLInputElement>(null);

    // Testbed State
    const [testbedInput, setTestbedInput] = useState('');
    const [chatHistory, setChatHistory] = useState<{ role: 'user' | 'model';
    content: string }[]>([]);
    const [isStreaming, setIsStreaming] = useState(false);

    const activePersonality = personalities.find(p => p.id === activeId);

    useEffect(() => {
        if (!activeId && personalities.length > 0) {
            setActiveId(personalities[0].id);
        }
    }, [personalities, activeId]);

    const handleUpdate = (field: keyof SystemPrompt, value: any) => {
        if (!activePersonality) return;
        const updated = { ...activePersonality, [field]: value };
        setPersonalities(personalities.map(p => (p.id === activeId ? updated : p)));
    };

    const handleAddNew = () => {
        const newId = Date.now().toString();
        const newPersonality: SystemPrompt = { ...defaultNewPrompt, id: newId, name:
        'Untitled Personality' };
        setPersonalities([...personalities, newPersonality]);
        setActiveId(newId);
```

```
};

const handleDelete = (id: string) => {
    if (window.confirm('Are you sure you want to delete this personality?')) {
        setPersonalities(personalities.filter(p => p.id !== id));
        if (activeId === id) {
            setActiveId(personalities.length > 1 ? personalities[0].id : null);
        }
    }
};

const handleTestbedSend = async () => {
    if (!testbedInput.trim() || !activePersonality || isStreaming) return;

    const systemInstruction = formatSystemPromptToString(activePersonality);
    const newHistory = [...chatHistory, { role: 'user' as const, content:
    testbedInput }];
    setChatHistory(newHistory);
    setTestbedInput('');
    setIsStreaming(true);

    try {
        const stream = streamContent(testbedInput, systemInstruction, 0.7);
        let fullResponse = '';
        setChatHistory(prev => [...prev, { role: 'model', content: '' }]);
        for await (const chunk of stream) {
            fullResponse += chunk;
            setChatHistory(prev => {
                const last = prev[prev.length - 1];
                if (last.role === 'model') {
                    return [...prev.slice(0, -1), { role: 'model', content: fullResponse }];
                }
                return prev;
            });
        }
    } catch (e) {
        const errorMsg = e instanceof Error ? e.message : 'An error occurred';
        setChatHistory(prev => [...prev, { role: 'model', content: `**Error:**
        ${errorMsg}` }]);
    } finally {
        setIsStreaming(false);
    }
};

const handleExport = () => {
    if (!activePersonality) return;
    downloadJson(activePersonality, `${activePersonality.name.replace(/\s+/g,
    '_')}.json`);
    addNotification('Personality exported!', 'success');
};

const handleImport = (e: React.ChangeEvent<HTMLInputElement>) => {
    const file = e.target.files?.[0];
    if (!file) return;
    const reader = new FileReader();
    reader.onload = (event) => {
        try {
            const imported = JSON.parse(event.target?.result as string) as SystemPrompt;
            // Basic validation
            if (imported.id && imported.name && imported.persona) {
                setPersonalities(prev => [...prev.filter(p => p.id !== imported.id), imported]);
                setActiveId(imported.id);
```

```
                addNotification('Personality imported!', 'success');
            } else {
                addNotification('Invalid personality file.', 'error');
            }
        } catch {
            addNotification('Failed to parse JSON file.', 'error');
        }
    };
    reader.readAsText(file);
};

return (
    <div className="h-full flex text-text-primary">
        {/* Sidebar */}
        <aside className="w-64 bg-surface border-r border-border flex flex-col">
            <div className="p-4 border-b border-border">
                <h2 className="text-lg font-bold">Personalities</h2>
            </div>
            <div className="flex-grow overflow-y-auto">
                {personalities.map(p => (
                    <div key={p.id} onClick={() => setActiveId(p.id)} className={`group flex
                    justify-between items-center p-3 text-sm cursor-pointer ${activeId === p.id ?
                    'bg-primary/10 text-primary' : 'hover:bg-gray-100 dark:hover:bg-slate-700'}`}>
                        <span className="truncate">{p.name}</span>
                        <button onClick={(e) => { e.stopPropagation(); handleDelete(p.id)}}
                        className="opacity-0 group-hover:opacity-100 text-text-secondary hover:text-
                        red-500"><TrashIcon /></button>
                    </div>
                ))}
            </div>
            <div className="p-4 border-t border-border space-y-2">
                <button onClick={handleAddNew} className="btn-primary w-full py-2 text-sm flex
                items-center justify-center gap-2"><PlusIcon /> New</button>
                <div className="flex gap-2">
                    <button onClick={() => fileInputRef.current?.click()} className="flex-1 py-2
                    text-sm bg-gray-100 dark:bg-slate-700 rounded-md flex items-center justify-
                    center gap-2"><ArrowUpOnSquareIcon/> Import</button>
                    <button onClick={handleExport} className="flex-1 py-2 text-sm bg-gray-100
                    dark:bg-slate-700 rounded-md flex items-center justify-center
                    gap-2"><ArrowDownTrayIcon/> Export</button>
                    <input type="file" ref={fileInputRef} onChange={handleImport} accept=".json"
                    className="hidden"/>
                </div>
            </div>
        </aside>
        {/* Main Content */}
        {activePersonality ? (
            <div className="flex-1 grid grid-cols-2 gap-px bg-border">
                {/* Editor */}
                <div className="bg-background p-4 flex flex-col gap-4 overflow-y-auto">
                    <div><label className="font-bold">Name</label><input type="text"
                    value={activePersonality.name} onChange={e => handleUpdate('name',
                    e.target.value)} className="w-full mt-1 p-2 bg-surface border rounded"/></div>
                    <div><label className="font-bold">Persona</label><textarea
                    value={activePersonality.persona} onChange={e => handleUpdate('persona',
                    e.target.value)} className="w-full mt-1 p-2 bg-surface border rounded
                    h-24"/></div>
                    <div><label className="font-bold">Rules (one per line)</label><textarea
                    value={activePersonality.rules.join('\n')} onChange={e => handleUpdate('rules',
                    e.target.value.split('\n'))} className="w-full mt-1 p-2 bg-surface border
                    rounded h-32"/></div>
                    <div><label className="font-bold">Output Format</label><select
```

```
                        value={activePersonality.outputFormat} onChange={e =>
                        handleUpdate('outputFormat', e.target.value)} className="w-full mt-1 p-2 bg-
                        surface border rounded"><option>markdown</option><option>json</option><option>te
                        xt</option></select></div>
                        <div>
                            <h3 className="font-bold mb-2">Examples</h3>
                            {activePersonality.exampleIO.map((ex, i) => (
                                <div key={i} className="grid grid-cols-2 gap-2 mb-2 p-2 border rounded bg-
                                surface">
                                    <textarea placeholder="User Input" value={ex.input} onChange={e =>
                                    handleUpdate('exampleIO', activePersonality.exampleIO.map((item, idx) => idx ===
                                    i ? {...item, input: e.target.value} : item))} className="h-20 p-1 bg-background
                                    border rounded"/>
                                    <textarea placeholder="Model Output" value={ex.output} onChange={e =>
                                    handleUpdate('exampleIO', activePersonality.exampleIO.map((item, idx) => idx ===
                                    i ? {...item, output: e.target.value} : item))} className="h-20 p-1 bg-
                                    background border rounded"/>
                                </div>
                            ))}
                            <button onClick={() => handleUpdate('exampleIO',
                            [...activePersonality.exampleIO, {input: '', output: ''}])} className="text-sm
                            text-primary">+ Add Example</button>
                        </div>
                    </div>
                    {/* Testbed */}
                    <div className="bg-background p-4 flex flex-col">
                        <h2 className="text-lg font-bold mb-2 border-b pb-2">Live Testbed</h2>
                        <div className="flex-grow overflow-y-auto space-y-4 pr-2">
                           {chatHistory.map((msg, i) => (
                               <div key={i} className={`p-3 rounded-lg ${msg.role === 'user' ? 'bg-primary/10'
                               : 'bg-surface'}`}>
                                    <strong className="capitalize">{msg.role}</strong>
                                    <MarkdownRenderer content={msg.content} />
                               </div>
                           ))}
                           {isStreaming && <div className="flex justify-center"><LoadingSpinner/></div>}
                        </div>
                        <div className="flex gap-2 mt-4">
                            <input value={testbedInput} onChange={e => setTestbedInput(e.target.value)}
                            onKeyDown={e => e.key === 'Enter' && handleTestbedSend()} className="flex-grow
                            p-2 bg-surface border rounded" placeholder="Test your AI..."/>
                            <button onClick={handleTestbedSend} disabled={isStreaming} className="btn-
                            primary px-4">Send</button>
                        </div>
                    </div>
                </div>
            ) : (
                <div className="flex-1 flex items-center justify-center text-text-
                secondary">Select or create a personality to begin.</div>
            )}
        </div>
    );
};


// ===== ProjectExplorer_16.tsx =====

import React, { useState, useEffect, useCallback } from 'react';
import { useGlobalState } from '../../contexts/GlobalStateContext.tsx';
import { useNotification } from '../../contexts/NotificationContext.tsx';
import { initializeOctokit } from '../../services/authService.ts';
import { getUserToken } from '../../services/firebaseService.ts';
```

```
import { getRepos, getRepoTree, getFileContent, commitFiles } from
'../../services/githubService.ts';
import { generateCommitMessageStream } from '../../services/geminiService.ts';
import type { Repo, FileNode } from '../../types.ts';
import { FolderIcon, DocumentIcon } from '../icons.tsx';
import { LoadingSpinner } from '../shared/index.tsx';
import * as Diff from 'diff';

const FileTree: React.FC<{ node: FileNode, onFileSelect: (path: string, name:
string) => void, activePath: string | null }> = ({ node, onFileSelect,
activePath }) => {
    const [isOpen, setIsOpen] = useState(true);

    if (node.type === 'file') {
        const isActive = activePath === node.path;
        return (
            <div
                className={`flex items-center space-x-2 pl-4 py-1 cursor-pointer rounded
                ${isActive ? 'bg-primary/10 text-primary' : 'hover:bg-gray-100 dark:hover:bg-
                slate-700'}`}
                onClick={() => onFileSelect(node.path, node.name)}
            >
                <DocumentIcon />
                <span>{node.name}</span>
            </div>
        );
    }

    return (
        <div>
            <div
                className="flex items-center space-x-2 py-1 cursor-pointer hover:bg-gray-100
                dark:hover:bg-slate-700 rounded"
                onClick={() => setIsOpen(!isOpen)}
            >
                <div className={`transform transition-transform ${isOpen ? 'rotate-90' :
                ''}`}>■</div>
                <FolderIcon />
                <span className="font-semibold">{node.name}</span>
            </div>
            {isOpen && node.children && (
                <div className="pl-4 border-l border-border ml-3">
                    {node.children.map(child => <FileTree key={child.path} node={child}
                    onFileSelect={onFileSelect} activePath={activePath} />)}
                </div>
            )}
        </div>
    );
};

export const ProjectExplorer: React.FC = () => {
    const { state, dispatch } = useGlobalState();
    const { user, githubUser, selectedRepo, projectFiles } = state;
    const { addNotification } = useNotification();
    const [repos, setRepos] = useState<Repo[]>([]);
    const [isLoading, setIsLoading] = useState<'repos' | 'tree' | 'file' | 'commit'
    | null>(null);
    const [error, setError] = useState('');
    const [activeFile, setActiveFile] = useState<{ path: string; name: string;
    originalContent: string; editedContent: string} | null>(null);

    const getApiClient = useCallback(async () => {
```

```
    if (!user) {
        throw new Error("You must be logged in to use the Project Explorer.");
    }
    const token = await getUserToken(user.uid, 'github_pat');
    if (!token) {
        throw new Error("GitHub token not found. Please add it on the Connections
        page.");
    }
    return initializeOctokit(token);
}, [user]);


useEffect(() => {
    const loadRepos = async () => {
        if (user && githubUser) {
            setIsLoading('repos');
            setError('');
            try {
                const octokit = await getApiClient();
                const userRepos = await getRepos(octokit);
                setRepos(userRepos);
            } catch (err) {
                setError(err instanceof Error ? err.message : 'Failed to load repositories');
            } finally {
                setIsLoading(null);
            }
        } else {
            setRepos([]);
        }
    };
    loadRepos();
}, [user, githubUser, getApiClient]);

useEffect(() => {
    const loadTree = async () => {
         if (selectedRepo && user && githubUser) {
            setIsLoading('tree');
            setError('');
            setActiveFile(null);
            try {
                const octokit = await getApiClient();
                const tree = await getRepoTree(octokit, selectedRepo.owner, selectedRepo.repo);
                dispatch({ type: 'LOAD_PROJECT_FILES', payload: tree });
            } catch (err) {
                 setError(err instanceof Error ? err.message : 'Failed to load repository tree');
            } finally {
                setIsLoading(null);
            }
        }
    };
    loadTree();
}, [selectedRepo, user, githubUser, dispatch, getApiClient]);

const handleFileSelect = async (path: string, name: string) => {
    if (!selectedRepo) return;
    setIsLoading('file');
    try {
        const octokit = await getApiClient();
        const content = await getFileContent(octokit, selectedRepo.owner,
        selectedRepo.repo, path);
        setActiveFile({ path, name, originalContent: content, editedContent: content });
    } catch (err) {
```

```
        setError((err as Error).message);
    } finally {
        setIsLoading(null);
    }
};

const handleCommit = async () => {
    if (!activeFile || !selectedRepo || activeFile.originalContent ===
    activeFile.editedContent) return;

    setIsLoading('commit');
    setError('');
    try {
        const diff = Diff.createPatch(activeFile.path, activeFile.originalContent,
        activeFile.editedContent);

        const stream = generateCommitMessageStream(diff);
        let commitMessage = '';
        for await (const chunk of stream) { commitMessage += chunk; }

        const finalMessage = window.prompt("Confirm or edit commit message:",
        commitMessage);
        if (!finalMessage) {
            setIsLoading(null);
            return;
        }

        const octokit = await getApiClient();
        await commitFiles(
            octokit,
            selectedRepo.owner,
            selectedRepo.repo,
            [{ path: activeFile.path, content: activeFile.editedContent }],
            finalMessage
        );

        addNotification(`Successfully committed to ${selectedRepo.repo}`, 'success');
        setActiveFile(prev => prev ? { ...prev, originalContent: prev.editedContent } :
        null);

    } catch (err) {
        const message = err instanceof Error ? err.message : 'Failed to commit changes';
        setError(message);
        addNotification(message, 'error');
    } finally {
        setIsLoading(null);
    }
};

if (!user) {
    return (
        <div className="h-full flex flex-col items-center justify-center text-center
        text-text-secondary p-4">
            <FolderIcon />
            <h2 className="text-lg font-semibold mt-2">Please Sign In</h2>
            <p>Sign in via the "Connections" tab to explore your repositories.</p>
        </div>
    );
}

if (!githubUser) {
     return (
```

```
        <div className="h-full flex flex-col items-center justify-center text-center
        text-text-secondary p-4">
            <FolderIcon />
            <h2 className="text-lg font-semibold mt-2">Connect to GitHub</h2>
            <p>Please go to the "Connections" tab and provide a Personal Access Token to
            explore your repositories.</p>
        </div>
    );
}

const hasChanges = activeFile ? activeFile.originalContent !==
activeFile.editedContent : false;

return (
    <div className="h-full flex flex-col text-text-primary">
        <header className="p-4 border-b border-border flex-shrink-0">
            <h1 className="text-xl font-bold flex items-center"><FolderIcon /><span
            className="ml-3">Project Explorer</span></h1>
            <div className="mt-2">
                <select
                    value={selectedRepo ? `${selectedRepo.owner}/${selectedRepo.repo}` : ''}
                    onChange={e => {
                        const [owner, repo] = e.target.value.split('/');
                        dispatch({ type: 'SET_SELECTED_REPO', payload: { owner, repo } });
                    }}
                    className="w-full p-2 bg-surface border border-border rounded-md text-sm"
                >
                    <option value="" disabled>{isLoading === 'repos' ? 'Loading...' : 'Select a
                    repository'}</option>
                    {repos.map(r => <option key={r.id} value={r.full_name}>{r.full_name}</option>)}
                </select>
            </div>
            {error && <p className="text-red-500 text-xs mt-2">{error}</p>}
        </header>
        <div className="flex-grow flex min-h-0">
            <aside className="w-1/3 bg-background border-r border-border p-4 overflow-y-
            auto">
                {isLoading === 'tree' && <div className="flex justify-center"><LoadingSpinner
                /></div>}
                {projectFiles && <FileTree node={projectFiles} onFileSelect={handleFileSelect}
                activePath={activeFile?.path ?? null} />}
            </aside>
            <main className="flex-1 bg-surface flex flex-col">
                 <div className="flex justify-between items-center p-2 border-b border-border bg-
                 gray-50 dark:bg-slate-800">
                    <span className="text-sm font-semibold">{activeFile?.name || 'No file
                    selected'}</span>
                    <button onClick={handleCommit} disabled={!hasChanges || isLoading === 'commit'}
                    className="btn-primary px-4 py-1 text-sm flex items-center justify-center
                    min-w-[100px]">
                       {isLoading === 'commit' ? <LoadingSpinner/> : 'Commit'}
                    </button>
                 </div>
                 {isLoading === 'file' ? <div className="flex items-center justify-center
                 h-full"><LoadingSpinner /></div> :
                    <textarea
                        value={activeFile?.editedContent ?? 'Select a file to view its content.'}
                        onChange={e => setActiveFile(prev => prev ? { ...prev, editedContent:
                        e.target.value } : null)}
                        disabled={!activeFile}
                        className="w-full h-full p-4 text-sm font-mono bg-transparent resize-none
                        focus:outline-none"
```

```
                        />
                    }
                </main>
            </div>
        </div>
    );
};

// ===== Connections_16.tsx =====

import React, { useState, useEffect } from 'react';
import { GithubIcon } from '../icons.tsx';
import { useGlobalState } from '../../contexts/GlobalStateContext.tsx';
import { signInWithGoogle, signOutUser, saveUserToken, getUserToken } from
'../../services/firebaseService.ts';
import { validateToken } from '../../services/authService.ts';
import { initializeOctokit } from '../../services/authService.ts';
import { LoadingSpinner } from '../shared/LoadingSpinner.tsx';
import type { GitHubUser } from '../../types.ts';
import { useNotification } from '../../contexts/NotificationContext.tsx';

const GitHubConnection: React.FC = () => {
    const { state, dispatch } = useGlobalState();
    const { user, githubUser } = state;
    const { addNotification } = useNotification();
    const [tokenInput, setTokenInput] = useState('');
    const [isLoading, setIsLoading] = useState(false);
    const [error, setError] = useState('');

    useEffect(() => {
        const checkStoredToken = async () => {
            if (user && !githubUser) {
                setIsLoading(true);
                const token = await getUserToken(user.uid, 'github_pat');
                if (token) {
                    try {
                        const githubProfile = await validateToken(token);
                        dispatch({ type: 'SET_GITHUB_USER', payload: githubProfile });
                    } catch (e) {
                        addNotification('Failed to validate stored GitHub token.', 'error');
                    }
                }
                setIsLoading(false);
            }
        };
        checkStoredToken();
    }, [user, githubUser, dispatch, addNotification]);

    const handleConnectGitHub = async () => {
        if (!user || !tokenInput.trim()) {
            setError('Token is required.');
            return;
        }
        setIsLoading(true);
        setError('');
        try {
            const githubProfile = await validateToken(tokenInput);
            await saveUserToken(user.uid, 'github_pat', tokenInput);
            dispatch({ type: 'SET_GITHUB_USER', payload: githubProfile });
            addNotification('GitHub connected successfully!', 'success');
            setTokenInput('');
        } catch (err) {
```

```
        setError(err instanceof Error ? `Invalid Token: ${err.message}` : 'Could not
        connect to GitHub.');
    } finally {
        setIsLoading(false);
    }
};

const handleDisconnectGitHub = async () => {
    if (!user) return;
    setIsLoading(true);
    try {
        await saveUserToken(user.uid, 'github_pat', ''); // Clear the token
        dispatch({ type: 'SET_GITHUB_USER', payload: null });
        addNotification('GitHub disconnected.', 'info');
    } catch (e) {
        addNotification('Failed to disconnect GitHub.', 'error');
    } finally {
        setIsLoading(false);
    }
};

if (!user) {
    return null; // Don't show GitHub section if not logged in
}

return (
    <div className="bg-surface border border-border rounded-lg p-6">
        <div className="flex items-center justify-between">
            <div className="flex items-center gap-4">
                <div className="w-10 h-10"><GithubIcon /></div>
                <div>
                    <h3 className="text-lg font-bold text-text-primary">GitHub</h3>
                    {githubUser ? (
                         <p className="text-sm text-green-600">Connected as {githubUser.login}</p>
                    ) : (
                         <p className="text-sm text-text-secondary">Not Connected</p>
                    )}
                </div>
            </div>
             {githubUser && (
                <button onClick={handleDisconnectGitHub} className="px-4 py-2 bg-red-500/10
                text-red-600 font-semibold rounded-lg hover:bg-red-500/20">
                    Disconnect
                </button>
             )}
        </div>

        {!githubUser && (
            <div className="mt-4 pt-4 border-t border-border">
                <label htmlFor="github-pat" className="block text-sm font-medium text-text-
                secondary mb-1">Personal Access Token (Classic)</label>
                <div className="flex gap-2">
                    <input
                        id="github-pat"
                        type="password"
                        value={tokenInput}
                        onChange={(e) => setTokenInput(e.target.value)}
                        placeholder="ghp_..."
                        className="flex-grow p-2 bg-background border border-border rounded-md text-sm"
                    />
                     <button onClick={handleConnectGitHub} disabled={isLoading} className="btn-
                     primary px-6 py-2 flex items-center justify-center min-w-[100px]">
```

```
                            {isLoading ? <LoadingSpinner /> : 'Connect'}
                        </button>
                    </div>
                    {error && <p className="text-red-500 text-xs mt-2">{error}</p>}
                    <p className="text-xs text-text-secondary mt-2">
                        Your token is stored in your private, secure Firestore document. Required
                        scopes: `repo`, `read:user`.
                    </p>
                </div>
            )}
        </div>
    );
};


export const Connections: React.FC = () => {
    const { state } = useGlobalState();
    const { user } = state;
    const { addNotification } = useNotification();
    const [isLoading, setIsLoading] = useState(false);

    const handleSignIn = async () => {
        setIsLoading(true);
        try {
            await signInWithGoogle();
            // State will be updated by the onAuthStateChanged listener
            addNotification('Signed in successfully!', 'success');
        } catch (error) {
            addNotification('Failed to sign in.', 'error');
            console.error(error);
        }
        setIsLoading(false);
    };

    return (
        <div className="h-full p-4 sm:p-6 lg:p-8">
            <div className="max-w-4xl mx-auto">
                <header className="mb-8">
                    <h1 className="text-4xl font-extrabold tracking-tight">Connections</h1>
                    <p className="mt-2 text-lg text-text-secondary">Sign in and connect to external
                    services.</p>
                </header>

                {!user ? (
                    <div className="text-center bg-surface p-8 rounded-lg border border-border">
                        <h2 className="text-xl font-bold">Sign In Required</h2>
                        <p className="text-text-secondary my-4">Please sign in with your Google account
                        to manage connections and save your data.</p>
                        <button onClick={handleSignIn} disabled={isLoading} className="btn-primary px-6
                        py-3 flex items-center justify-center gap-2 mx-auto">
                            {isLoading ? <LoadingSpinner/> : 'Sign in with Google'}
                        </button>
                    </div>
                ) : (
                    <div className="space-y-6">
                        <GitHubConnection />
                    </div>
                )}
            </div>
        </div>
    );
};
```

```
// ===== AiFeatureBuilder_21.tsx =====


import React, { useState, useCallback, useEffect } from 'react';
import type { GeneratedFile } from '../../types.ts';
import { generateFeature, generateFullStackFeature, generateUnitTestsStream,
generateCommitMessageStream, generateDockerfile } from
'../../services/geminiService.ts';
import { deployCloudFunction, deployFirestoreRules } from
'../../services/gcpService.ts';
import { saveFile, getAllFiles, clearAllFiles } from
'../../services/dbService.ts';
import { useNotification } from '../../contexts/NotificationContext.tsx';
import { CpuChipIcon, DocumentTextIcon, BeakerIcon, GitBranchIcon, CloudIcon,
PaperAirplaneIcon } from '../icons.tsx';
import { LoadingSpinner, MarkdownRenderer } from '../shared/index.tsx';

type SupplementalTab = 'TESTS' | 'COMMIT' | 'DEPLOYMENT' | 'CODE';
type OutputTab = GeneratedFile | SupplementalTab;

export const AiFeatureBuilder: React.FC = () => {
    const [prompt, setPrompt] = useState<string>('A simple "Hello World" React
    component with a button that shows an alert.');
    const [framework] = useState('React');
    const [styling] = useState('Tailwind CSS');
    const [includeBackend, setIncludeBackend] = useState(false);

    const [generatedFiles, setGeneratedFiles] = useState<GeneratedFile[]>([]);
    const [unitTests, setUnitTests] = useState<string>('');
    const [commitMessage, setCommitMessage] = useState<string>('');
    const [dockerfile, setDockerfile] = useState<string>('');

    const [activeTab, setActiveTab] = useState<OutputTab>('CODE');
    const [isLoading, setIsLoading] = useState<boolean>(false);
    const [isDeploying, setIsDeploying] = useState(false);
    const [error, setError] = useState<string>('');
    const { addNotification } = useNotification();

    useEffect(() => {
        const loadFiles = async () => {
            const files = await getAllFiles();
            setGeneratedFiles(files);
            if (files.length > 0) setActiveTab(files[0]);
        };
        loadFiles();
    }, []);

    const handleGenerate = useCallback(async () => {
        if (!prompt.trim()) { setError('Please enter a feature description.'); return; }
        setIsLoading(true);
        setError('');
        await clearAllFiles();
        setGeneratedFiles([]); setUnitTests(''); setCommitMessage('');
        setDockerfile(''); setActiveTab('CODE');

        try {
            const resultFiles = includeBackend
                ? await generateFullStackFeature(prompt, framework, styling)
                : await generateFeature(prompt, framework, styling);

            for (const file of resultFiles) { await saveFile(file); }
            setGeneratedFiles(resultFiles);
```

```
        if (resultFiles.length > 0) {
            const componentFile = resultFiles.find(f => f.filePath.endsWith('.tsx') ||
            f.filePath.endsWith('.jsx'));
            setActiveTab(componentFile || resultFiles[0]);

            const testStream = generateUnitTestsStream(componentFile?.content ||
            resultFiles[0].content);
            const diffContext = resultFiles.map(f => `File:
            ${f.filePath}\n\n${f.content}`).join('\n---\n');
            const commitStream = generateCommitMessageStream(diffContext);

            let tests = ''; for await (const chunk of testStream) { tests += chunk;
            setUnitTests(tests); }
            let commit = ''; for await (const chunk of commitStream) { commit += chunk;
            setCommitMessage(commit); }

            if (!includeBackend) {
                const dockerfileStream = generateDockerfile(framework);
                let docker = ''; for await (const chunk of dockerfileStream) { docker += chunk;
                setDockerfile(docker); }
            }
        }
    } catch (err) {
        setError(err instanceof Error ? err.message : 'Failed to generate feature.');
    } finally {
        setIsLoading(false);
    }
}, [prompt, framework, styling, includeBackend]);

const handleDeploy = async () => {
    if (!includeBackend || generatedFiles.length === 0) return;
    setIsDeploying(true);
    try {
        const backendFile = generatedFiles.find(f =>
        f.filePath.includes('functions/index.js'));
        const rulesFile = generatedFiles.find(f =>
        f.filePath.includes('firestore.rules'));

        if (backendFile) {
            addNotification('Deploying Cloud Function...', 'info');
            await deployCloudFunction(backendFile.content, 'myGeneratedFunction');
            addNotification('Cloud Function deployment initiated.', 'success');
        }
        if (rulesFile) {
            addNotification('Deploying Firestore Rules...', 'info');
            await deployFirestoreRules(rulesFile.content);
            addNotification('Firestore Rules deployment initiated.', 'success');
        }
    } catch(e) {
        addNotification(e instanceof Error ? e.message : 'Deployment failed', 'error');
    } finally {
        setIsDeploying(false);
    }
}

const renderContent = () => {
    if (typeof activeTab === 'string') {
        switch (activeTab) {
            case 'TESTS': return <MarkdownRenderer content={unitTests} />;
            case 'COMMIT': return <pre className="w-full h-full p-4 whitespace-pre-wrap
            font-sans text-sm">{commitMessage}</pre>;
            case 'DEPLOYMENT': return <MarkdownRenderer content={dockerfile} />;
```

```
            default: return <div className="p-4">Select a file</div>;
        }
    }
    return <MarkdownRenderer content={'```tsx\n' + activeTab.content + '\n```'} />;
}

return (
    <div className="h-full flex flex-col text-text-primary bg-surface">
        <header className="p-4 border-b border-border flex-shrink-0">
            <h1 className="text-xl font-bold flex items-center"><CpuChipIcon /><span
            className="ml-3">AI Feature Builder</span></h1>
        </header>
        <div className="flex-grow flex min-h-0">
            <main className="flex-1 flex flex-col min-w-0">
                <div className="flex-grow flex flex-col bg-background">
                     <div className="border-b border-border flex items-center bg-surface overflow-x-
                     auto">
                        {generatedFiles.map(file => (
                            <button key={file.filePath} onClick={() => setActiveTab(file)} className={`flex-
                            shrink-0 flex items-center gap-2 px-4 py-2 text-sm ${activeTab === file ? 'bg-
                            background border-b-2 border-primary text-text-primary' : 'text-text-secondary
                            hover:bg-gray-50'}`}><DocumentTextIcon /> {file.filePath}</button>
                        ))}
                        {unitTests && <button onClick={() => setActiveTab('TESTS')} className={`flex-
                        shrink-0 flex items-center gap-2 px-4 py-2 text-sm ${activeTab === 'TESTS' ?
                        'bg-background border-b-2 border-primary text-text-primary' : 'text-text-
                        secondary hover:bg-gray-50'}`}><BeakerIcon /> Tests</button>}
                        {commitMessage && <button onClick={() => setActiveTab('COMMIT')}
                        className={`flex-shrink-0 flex items-center gap-2 px-4 py-2 text-sm ${activeTab
                        === 'COMMIT' ? 'bg-background border-b-2 border-primary text-text-primary' :
                        'text-text-secondary hover:bg-gray-50'}`}><GitBranchIcon /> Commit</button>}
                        {dockerfile && !includeBackend && <button onClick={() =>
                        setActiveTab('DEPLOYMENT')} className={`flex-shrink-0 flex items-center gap-2
                        px-4 py-2 text-sm ${activeTab === 'DEPLOYMENT' ? 'bg-background border-b-2
                        border-primary text-text-primary' : 'text-text-secondary hover:bg-
                        gray-50'}`}><CloudIcon /> Dockerfile</button>}
                    </div>
                    <div className="flex-grow p-2 overflow-auto">
                        {isLoading && !generatedFiles.length ? <div className="flex justify-center
                        items-center h-full"><LoadingSpinner/></div> : renderContent()}
                    </div>
                </div>

                <div className="flex-shrink-0 p-4 border-t border-border bg-surface">
                     <div className="flex items-center gap-2 mb-2">
                        <label className="flex items-center gap-2 text-sm"><input type="checkbox"
                        checked={includeBackend} onChange={e => setIncludeBackend(e.target.checked)} />
                        Include Backend (Cloud Function + Firestore)</label>
                    </div>
                    <textarea value={prompt} onChange={(e) => setPrompt(e.target.value)}
                    placeholder="e.g., A user profile card with an avatar, name, and bio."
                    className="w-full p-2 bg-background border border-border rounded-md resize-none
                    text-sm h-20"/>
                     <div className="flex gap-2 mt-2">
                         <button onClick={handleGenerate} disabled={isLoading || isDeploying}
                         className="btn-primary flex-grow flex items-center justify-center gap-2 px-4
                         py-2">
                            {isLoading ? <><LoadingSpinner /> Generating...</> : 'Generate Feature'}
                        </button>
                        {includeBackend && generatedFiles.length > 0 && (
                            <button onClick={handleDeploy} disabled={isLoading || isDeploying}
                            className="bg-blue-600 text-white font-bold rounded-md hover:opacity-90
```

```
                                transition-all disabled:opacity-50 shadow-md flex-grow flex items-center
                                justify-center gap-2 px-4 py-2">
                                    {isDeploying ? <><LoadingSpinner /> Deploying...</> : <><PaperAirplaneIcon/>
                                    Deploy to GCP</>}
                                </button>
                            )}
                        </div>
                        {error && <p className="text-red-600 text-xs mt-2 text-center">{error}</p>}
                    </div>
                </main>
            </div>
        </div>
    );
};


// ===== Connections_19.tsx =====


import React, { useState, useEffect } from 'react';
import { GithubIcon, MapIcon } from '../icons.tsx';
import { useGlobalState } from '../../contexts/GlobalStateContext.tsx';
import { signInWithGoogle } from '../../services/firebaseService.ts';
import { validateToken } from '../../services/authService.ts';
import * as vaultService from '../../services/vaultService.ts';
import { useVaultModal } from '../../contexts/VaultModalContext.tsx';
import { LoadingSpinner } from '../shared/LoadingSpinner.tsx';
import type { GitHubUser } from '../../types.ts';
import { useNotification } from '../../contexts/NotificationContext.tsx';

const GitHubConnection: React.FC = () => {
    const { state, dispatch } = useGlobalState();
    const { user, githubUser } = state;
    const { addNotification } = useNotification();
    const { requestUnlock } = useVaultModal();
    const [tokenInput, setTokenInput] = useState('');
    const [isLoading, setIsLoading] = useState(false);
    const [error, setError] = useState('');

    useEffect(() => {
        const checkStoredToken = async () => {
            if (user && !githubUser && state.vaultState.isUnlocked) {
                setIsLoading(true);
                const token = await vaultService.getDecryptedCredential('github_pat');
                if (token) {
                    try {
                        const githubProfile = await validateToken(token);
                        dispatch({ type: 'SET_GITHUB_USER', payload: githubProfile });
                    } catch (e) {
                        addNotification('Failed to validate stored GitHub token.', 'error');
                    }
                }
                setIsLoading(false);
            }
        };
        checkStoredToken();
    }, [user, githubUser, dispatch, addNotification, state.vaultState.isUnlocked]);

    const handleConnectGitHub = async () => {
        if (!user || !tokenInput.trim()) {
            setError('Token is required.');
            return;
```

```
    }
    if (!state.vaultState.isUnlocked) {
        const unlocked = await requestUnlock();
        if (!unlocked) return;
    }

    setIsLoading(true);
    setError('');
    try {
        const githubProfile = await validateToken(tokenInput);
        await vaultService.saveCredential('github_pat', tokenInput);
        dispatch({ type: 'SET_GITHUB_USER', payload: githubProfile });
        addNotification('GitHub connected successfully!', 'success');
        setTokenInput('');
    } catch (err) {
        setError(err instanceof Error ? `Invalid Token: ${err.message}` : 'Could not
        connect to GitHub.');
    } finally {
        setIsLoading(false);
    }
};

const handleDisconnectGitHub = async () => {
    if (!user) return;
    setIsLoading(true);
    try {
        await vaultService.saveCredential('github_pat', '');
        dispatch({ type: 'SET_GITHUB_USER', payload: null });
        addNotification('GitHub disconnected.', 'info');
    } catch (e) {
        addNotification('Failed to disconnect GitHub.', 'error');
    } finally {
        setIsLoading(false);
    }
};

if (!user) {
    return null;
}

return (
    <div className="bg-surface border border-border rounded-lg p-6">
        <div className="flex items-center justify-between">
            <div className="flex items-center gap-4">
                <div className="w-10 h-10"><GithubIcon /></div>
                <div>
                    <h3 className="text-lg font-bold text-text-primary">GitHub</h3>
                    {githubUser ? (
                         <p className="text-sm text-green-600">Connected as {githubUser.login}</p>
                    ) : (
                         <p className="text-sm text-text-secondary">Not Connected</p>
                    )}
                </div>
            </div>
             {githubUser && (
                <button onClick={handleDisconnectGitHub} className="px-4 py-2 bg-red-500/10
                text-red-600 font-semibold rounded-lg hover:bg-red-500/20">
                    Disconnect
                </button>
             )}
        </div>
        {!githubUser && (
```

```
                <div className="mt-4 pt-4 border-t border-border">
                    <label htmlFor="github-pat" className="block text-sm font-medium text-text-
                    secondary mb-1">Personal Access Token (Classic)</label>
                    <div className="flex gap-2">
                        <input id="github-pat" type="password" value={tokenInput} onChange={(e) =>
                        setTokenInput(e.target.value)} placeholder="ghp_..." className="flex-grow p-2
                        bg-background border border-border rounded-md text-sm" />
                         <button onClick={handleConnectGitHub} disabled={isLoading} className="btn-
                         primary px-6 py-2 flex items-center justify-center min-w-[100px]">
                            {isLoading ? <LoadingSpinner /> : 'Connect'}
                        </button>
                    </div>
                    {error && <p className="text-red-500 text-xs mt-2">{error}</p>}
                </div>
            )}
        </div>
    );
};

const GoogleMapsConnection: React.FC = () => {
    const { state } = useGlobalState();
    const { user } = state;
    const { addNotification } = useNotification();
    const { requestUnlock } = useVaultModal();
    const [apiKey, setApiKey] = useState('');
    const [isSaved, setIsSaved] = useState(false);
    const [isLoading, setIsLoading] = useState(false);

    useEffect(() => {
        const checkKey = async () => {
            if(user && state.vaultState.isUnlocked) {
                const key = await vaultService.getDecryptedCredential('google_maps_api_key');
                setIsSaved(!!key);
            }
        };
        checkKey();
    }, [user, state.vaultState.isUnlocked]);

    const handleSaveKey = async () => {
        if (!apiKey.trim()) return;
        if (!state.vaultState.isUnlocked) {
            const unlocked = await requestUnlock();
            if (!unlocked) return;
        }
        setIsLoading(true);
        try {
            await vaultService.saveCredential('google_maps_api_key', apiKey);
            addNotification('Google Maps API Key saved!', 'success');
            setIsSaved(true);
            setApiKey('');
        } catch(e) {
            addNotification('Failed to save key.', 'error');
        } finally {
            setIsLoading(false);
        }
    }

    if (!user) return null;

    return (
        <div className="bg-surface border border-border rounded-lg p-6">
            <div className="flex items-center gap-4">
```

```
                <div className="w-10 h-10"><MapIcon /></div>
                <div>
                    <h3 className="text-lg font-bold text-text-primary">Google Maps Platform</h3>
                    <p className={`text-sm ${isSaved ? 'text-green-600' : 'text-text-
                    secondary'}`}>{isSaved ? 'API Key is saved in vault' : 'Not Connected'}</p>
                </div>
            </div>
             <div className="mt-4 pt-4 border-t border-border">
                <label htmlFor="maps-key" className="block text-sm font-medium text-text-
                secondary mb-1">API Key</label>
                <div className="flex gap-2">
                    <input id="maps-key" type="password" value={apiKey} onChange={(e) =>
                    setApiKey(e.target.value)} placeholder="AIzaSy..." className="flex-grow p-2 bg-
                    background border border-border rounded-md text-sm" />
                     <button onClick={handleSaveKey} disabled={isLoading} className="btn-primary px-6
                     py-2 flex items-center justify-center min-w-[100px]">
                        {isLoading ? <LoadingSpinner /> : 'Save'}
                    </button>
                </div>
            </div>
        </div>
    )
}

export const Connections: React.FC = () => {
    const { state } = useGlobalState();
    const { user } = state;
    const { addNotification } = useNotification();
    const [isLoading, setIsLoading] = useState(false);

    const handleSignIn = async () => {
        setIsLoading(true);
        try {
            await signInWithGoogle();
            addNotification('Signed in successfully!', 'success');
        } catch (error) {
            addNotification('Failed to sign in.', 'error');
            console.error(error);
        }
        setIsLoading(false);
    };

    return (
        <div className="h-full p-4 sm:p-6 lg:p-8">
            <div className="max-w-4xl mx-auto">
                <header className="mb-8">
                    <h1 className="text-4xl font-extrabold tracking-tight">Connections</h1>
                    <p className="mt-2 text-lg text-text-secondary">Sign in and connect to external
                    services.</p>
                </header>

                {!user ? (
                    <div className="text-center bg-surface p-8 rounded-lg border border-border">
                        <h2 className="text-xl font-bold">Sign In Required</h2>
                        <p className="text-text-secondary my-4">Please sign in with your Google account
                        to manage connections and save your data.</p>
                        <button onClick={handleSignIn} disabled={isLoading} className="btn-primary px-6
                        py-3 flex items-center justify-center gap-2 mx-auto">
                            {isLoading ? <LoadingSpinner/> : 'Sign in with Google'}
                        </button>
                    </div>
                ) : (
```

```
                    <div className="space-y-6">
                        <GitHubConnection />
                        <GoogleMapsConnection />
                    </div>
                )}
            </div>
        </div>
    );
};

// ===== MarkdownSlides_21.tsx =====


import React, { useState, useMemo, useEffect, useRef, useCallback } from
'react';
import { marked } from 'marked';
import { PhotoIcon } from '../icons.tsx';
import { useNotification } from '../../contexts/NotificationContext.tsx';
import { useGlobalState } from '../../contexts/GlobalStateContext.tsx';
import { summarizeForSlides } from '../../services/geminiService.ts';
import { createPresentation, addSlide } from
'../../services/workspaceService.ts';
import { LoadingSpinner } from '../shared/index.tsx';

const exampleMarkdown = `# Slide 1: Welcome

This is a slide deck generated from Markdown.

- Use standard markdown syntax
- Like lists, headers, and **bold** text.

---

# Slide 2: Features

Navigate using the buttons below.

\`\`\`javascript
console.log("Code blocks work too!");
\`\`\`

---

# Slide 3: The End

Easy to create and present.
`;

export const MarkdownSlides: React.FC = () => {
    const [markdown, setMarkdown] = useState(exampleMarkdown);
    const [currentSlide, setCurrentSlide] = useState(0);
    const [slideHtml, setSlideHtml] = useState<string | TrustedHTML>('');
    const [isExporting, setIsExporting] = useState(false);
    const presentationRef = useRef<HTMLDivElement>(null);
    const { addNotification } = useNotification();
    const { state } = useGlobalState();
    const { user } = state;

    const slides = useMemo(() => markdown.split(/^-{3,}\s*$/m), [markdown]);

    useEffect(() => {
        const parse = async () => {
```

```
        const currentSlideContent = slides[currentSlide] || '';
        const html = await marked.parse(currentSlideContent);
        setSlideHtml(html);
    };
    parse();
}, [slides, currentSlide]);

const goToNext = useCallback(() => setCurrentSlide(s => Math.min(s + 1,
slides.length - 1)), [slides.length]);
const goToPrev = useCallback(() => setCurrentSlide(s => Math.max(s - 1, 0)),
[]);

const handleFullscreen = () => {
    presentationRef.current?.requestFullscreen();
};

const handleExportToSlides = async () => {
    if (!user) {
        addNotification('Please sign in with Google to export to Slides.', 'error');
        return;
    }
    setIsExporting(true);
    try {
        addNotification('Creating presentation...', 'info');
        const presentation = await createPresentation(`Presentation from DevCore: ${new
        Date().toLocaleDateString()}`);
        addNotification(`Presentation created: ${presentation.presentationId}`,
        'success');

        for (let i = 0; i < slides.length; i++) {
            addNotification(`Processing slide ${i + 1}/${slides.length}...`, 'info');
            const slideContent = slides[i];
            const summary = await summarizeForSlides(slideContent);
            await addSlide(presentation.presentationId, summary);
        }
        addNotification('All slides added successfully!', 'success');
        window.open(presentation.webViewLink, '_blank');

    } catch (e) {
        console.error(e);
        addNotification(e instanceof Error ? e.message : 'Failed to export
        presentation.', 'error');
    } finally {
        setIsExporting(false);
    }
};

useEffect(() => {
    const handleKeyDown = (e: KeyboardEvent) => {
        if (document.fullscreenElement === presentationRef.current) {
            if (e.key === 'ArrowRight' || e.key === ' ') goToNext();
            if (e.key === 'ArrowLeft') goToPrev();
        }
    };
    document.addEventListener('keydown', handleKeyDown);
    return () => document.removeEventListener('keydown', handleKeyDown);
}, [goToNext, goToPrev]);

return (
    <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
        <header className="mb-6">
            <h1 className="text-3xl font-bold flex items-center"><PhotoIcon /><span
```

```
                className="ml-3">Markdown to Slides</span></h1>
                <p className="text-text-secondary mt-1">Write markdown, present it as a
                slideshow. Use '---' to separate slides.</p>
            </header>
            <div className="flex-grow grid grid-cols-1 lg:grid-cols-2 gap-6 h-full overflow-
            hidden">
                <div className="flex flex-col h-full">
                     <label htmlFor="md-input" className="text-sm font-medium text-text-secondary
                     mb-2">Markdown Editor</label>
                     <textarea id="md-input" value={markdown} onChange={e =>
                     setMarkdown(e.target.value)} className="flex-grow p-4 bg-surface border border-
                     border rounded-md resize-none font-mono text-sm focus:ring-2 focus:ring-primary
                     focus:outline-none"/>
                </div>
                 <div ref={presentationRef} className="flex flex-col h-full bg-surface
                 fullscreen:bg-background border border-border rounded-md">
                    <div className="flex-shrink-0 flex justify-end items-center p-2 border-b border-
                    border gap-2">
                        <button onClick={handleExportToSlides} disabled={isExporting || !user}
                        className="btn-primary text-xs px-3 py-1 flex items-center gap-1 disabled:bg-
                        gray-400">
                           {isExporting ? <LoadingSpinner/> : 'Export to Google Slides'}
                        </button>
                        <button onClick={handleFullscreen} className="px-3 py-1 bg-gray-100 dark:bg-
                        slate-700 rounded-md text-xs hover:bg-gray-200 dark:hover:bg-
                        slate-600">Fullscreen</button>
                    </div>
                    <div className="relative flex-grow flex flex-col justify-center items-center p-8
                    overflow-y-auto">
                        <div className="prose prose-lg max-w-none w-full" dangerouslySetInnerHTML={{
                        __html: slideHtml }} />
                         <button onClick={goToPrev} disabled={currentSlide === 0} className="absolute
                         left-4 top-1/2 -translate-y-1/2 p-2 bg-gray-200/50 dark:bg-slate-700/50 rounded-
                         full disabled:opacity-30 hover:bg-gray-300/50 dark:hover:bg-
                         slate-600/50">■</button>
                         <button onClick={goToNext} disabled={currentSlide === slides.length - 1}
                         className="absolute right-4 top-1/2 -translate-y-1/2 p-2 bg-gray-200/50 dark:bg-
                         slate-700/50 rounded-full disabled:opacity-30 hover:bg-gray-300/50
                         dark:hover:bg-slate-600/50">■</button>
                         <div className="absolute bottom-4 right-4 text-xs bg-black/50 px-2 py-1 rounded-
                         md text-white">
                            {currentSlide + 1} / {slides.length}
                         </div>
                    </div>
                </div>
            </div>
        </div>
    );
};

// ===== WeeklyDigestGenerator.tsx =====


import React, { useState, useCallback } from 'react';
import { generateWeeklyDigest } from '../../services/geminiService.ts';
import { sendEmail } from '../../services/workspaceService.ts';
import { useGlobalState } from '../../contexts/GlobalStateContext.tsx';
import { useNotification } from '../../contexts/NotificationContext.tsx';
import { MailIcon, SparklesIcon } from '../icons.tsx';
import { LoadingSpinner } from '../shared/index.tsx';

// Dummy data for demonstration purposes
```

```
const dummyCommitLogs = `
feat: implement user authentication
fix: resolve issue with button alignment
feat: add dark mode toggle
chore: update dependencies
refactor: simplify data fetching logic
`;
const dummyTelemetry = {
    avgPageLoad: 120,
    errorRate: '0.5%',
    uptime: '99.98%'
};

export const WeeklyDigestGenerator: React.FC = () => {
    const { state } = useGlobalState();
    const { user } = state;
    const { addNotification } = useNotification();
    const [emailHtml, setEmailHtml] = useState('');
    const [isLoading, setIsLoading] = useState(false);
    const [isSending, setIsSending] = useState(false);

    const handleGenerate = useCallback(async () => {
        setIsLoading(true);
        setEmailHtml('');
        try {
            const html = await generateWeeklyDigest(dummyCommitLogs, dummyTelemetry);
            setEmailHtml(html);
            addNotification('Digest content generated!', 'success');
        } catch (e) {
            addNotification(e instanceof Error ? e.message : 'Failed to generate digest',
            'error');
        } finally {
            setIsLoading(false);
        }
    }, []);

    const handleSend = async () => {
        if (!emailHtml || !user?.email) {
            addNotification('Cannot send email. Generate content and ensure you are signed
            in.', 'error');
            return;
        }
        setIsSending(true);
        try {
            await sendEmail(user.email, 'Your Weekly Project Digest', emailHtml);
            addNotification(`Weekly digest sent to ${user.email}!`, 'success');
        } catch (e) {
             addNotification(e instanceof Error ? e.message : 'Failed to send email',
             'error');
        } finally {
            setIsSending(false);
        }
    };

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
            <header className="mb-6">
                <h1 className="text-3xl font-bold flex items-center"><MailIcon /><span
                className="ml-3">Weekly Digest Generator</span></h1>
                <p className="text-text-secondary mt-1">Generate an AI-powered weekly summary
                and send it via your Gmail.</p>
            </header>
```

```
            <div className="flex-grow grid grid-cols-1 lg:grid-cols-2 gap-6 min-h-0">
                <div className="bg-surface p-4 border border-border rounded-lg flex flex-col
                items-center justify-center text-center">
                    <h3 className="text-lg font-bold">Generate & Send</h3>
                    <p className="text-sm text-text-secondary my-4">This tool will use dummy data to
                    generate a project summary and send it to your signed-in email address
                    ({user?.email || 'N/A'}).</p>
                    <div className="flex flex-col gap-4 w-full max-w-xs">
                        <button onClick={handleGenerate} disabled={isLoading || isSending}
                        className="btn-primary flex items-center justify-center gap-2 py-3">
                            {isLoading ? <LoadingSpinner /> : <><SparklesIcon /> Generate Digest</>}
                        </button>
                        <button onClick={handleSend} disabled={!emailHtml || isSending || isLoading}
                        className="btn-primary flex items-center justify-center gap-2 py-3 bg-
                        emerald-600">
                           {isSending ? <LoadingSpinner /> : 'Send Email via Gmail'}
                        </button>
                    </div>
                </div>

                <div className="bg-surface p-4 border border-border rounded-lg flex flex-col">
                    <h3 className="text-lg font-bold mb-2">Email Preview</h3>
                    <div className="flex-grow bg-white border rounded overflow-hidden">
                        {isLoading && <div className="flex justify-center items-center
                        h-full"><LoadingSpinner /></div>}
                        {emailHtml && <iframe srcDoc={emailHtml} title="Email Preview" className="w-full
                        h-full" />}
                        {!isLoading && !emailHtml && <div className="flex justify-center items-center
                        h-full text-text-secondary">Preview will appear here.</div>}
                    </div>
                </div>
            </div>
        </div>
    );
};

// ===== AiPullRequestAssistant_22.tsx =====


import React, { useState, useMemo, useCallback } from 'react';
import * as Diff from 'diff';
import { generatePrSummaryStructured, generateTechnicalSpecFromDiff,
downloadFile } from '../../services/index.ts';
import { createDocument, insertText } from '../../services/workspaceService.ts';
import type { StructuredPrSummary } from '../../types.ts';
import { AiPullRequestAssistantIcon, DocumentIcon } from '../icons.tsx';
import { LoadingSpinner } from '../shared/index.tsx';
import { useNotification } from '../../contexts/NotificationContext.tsx';
import { useGlobalState } from '../../contexts/GlobalStateContext.tsx';

const exampleBefore = `function Greeter(props) {
  return <h1>Hello, {props.name}!</h1>;
}`;
const exampleAfter = `function Greeter({ name, enthusiasmLevel = 1 }) {
  const punctuation = '!'.repeat(enthusiasmLevel);
  return <h1>Hello, {name}{punctuation}</h1>;
}`;

export const AiPullRequestAssistant: React.FC = () => {
    const [beforeCode, setBeforeCode] = useState<string>(exampleBefore);
    const [afterCode, setAfterCode] = useState<string>(exampleAfter);
    const [isLoading, setIsLoading] = useState<boolean>(false);
```

```
const [isExporting, setIsExporting] = useState<boolean>(false);
const [error, setError] = useState<string>('');
const [summary, setSummary] = useState<StructuredPrSummary | null>(null);

const { addNotification } = useNotification();
const { state } = useGlobalState();
const { user } = state;

const diff = useMemo(() => Diff.createPatch('component.tsx', beforeCode,
afterCode), [beforeCode, afterCode]);

const handleGenerateSummary = useCallback(async () => {
    if (!beforeCode.trim() && !afterCode.trim()) {
        setError('Please provide code to generate a summary.');
        return;
    }
    setIsLoading(true);
    setError('');
    setSummary(null);

    try {
        const result: StructuredPrSummary = await generatePrSummaryStructured(diff);
        setSummary(result);
    } catch (err) {
        const errorMessage = err instanceof Error ? err.message : 'An unknown error
        occurred.';
        setError(`Failed to generate summary: ${errorMessage}`);
    } finally {
        setIsLoading(false);
    }
}, [diff, beforeCode, afterCode]);

const handleExportToDocs = async () => {
    if (!summary || !user) {
        addNotification('Please generate a summary first and ensure you are signed in.',
        'error');
        return;
    }
    setIsExporting(true);
    try {
        const specContent = await generateTechnicalSpecFromDiff(diff, summary);
        const doc = await createDocument(`Tech Spec: ${summary.title}`);
        await insertText(doc.documentId, specContent);
        addNotification('Successfully exported to Google Docs!', 'success');
        window.open(doc.webViewLink, '_blank');
    } catch (err) {
        const errorMessage = err instanceof Error ? err.message : 'An unknown error
        occurred.';
        addNotification(`Failed to export: ${errorMessage}`, 'error');
    } finally {
        setIsExporting(false);
    }
};

return (
    <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
        <header className="mb-6">
            <h1 className="text-3xl font-bold flex items-center">
                <AiPullRequestAssistantIcon />
                <span className="ml-3">AI Pull Request Assistant</span>
            </h1>
            <p className="text-text-secondary mt-1">Generate a PR summary from code changes
```

```
    and export a full tech spec.</p>
</header>
<div className="flex-grow grid grid-cols-1 lg:grid-cols-2 gap-6 min-h-0">
    {/* Left side: Inputs and Generator */}
    <div className="flex flex-col gap-4 min-h-0">
        <div className="flex flex-col flex-1 min-h-0">
            <label htmlFor="before-code" className="text-sm font-medium text-text-secondary
            mb-2">Before</label>
            <textarea id="before-code" value={beforeCode} onChange={e =>
            setBeforeCode(e.target.value)} className="flex-grow p-4 bg-surface border
            border-border rounded-md resize-none font-mono text-sm" />
        </div>
        <div className="flex flex-col flex-1 min-h-0">
            <label htmlFor="after-code" className="text-sm font-medium text-text-secondary
            mb-2">After</label>
            <textarea id="after-code" value={afterCode} onChange={e =>
            setAfterCode(e.target.value)} className="flex-grow p-4 bg-surface border border-
            border rounded-md resize-none font-mono text-sm" />
        </div>
        <button onClick={handleGenerateSummary} disabled={isLoading} className="btn-
        primary w-full flex items-center justify-center px-6 py-3">
            {isLoading ? <LoadingSpinner /> : 'Generate Summary'}
        </button>
        {error && <p className="text-red-500 text-xs text-center">{error}</p>}
    </div>

    {/* Right side: Summary and Export */}
    <div className="flex flex-col gap-4 min-h-0">
        <div className="flex flex-col bg-surface border border-border p-4 rounded-lg
        flex-grow min-h-0">
            <h3 className="text-lg font-bold mb-2">Generated Summary</h3>
            <div className="flex-grow overflow-y-auto pr-2 space-y-2">
                {summary ? (
                    <>
                        <input type="text" readOnly value={summary.title} className="w-full font-bold
                        p-2 bg-background rounded"/>
                        <textarea readOnly value={summary.summary} className="w-full h-24 p-2 bg-
                        background rounded resize-none"/>
                        <div>
                            <h4 className="font-semibold">Changes:</h4>
                            <ul className="list-disc list-inside text-sm">
                                {summary.changes.map((c, i) => <li key={i}>{c}</li>)}
                            </ul>
                        </div>
                    </>
                ) : (
                    <div className="text-text-secondary h-full flex items-center justify-center">
                        {isLoading ? <LoadingSpinner /> : 'PR summary will appear here.'}
                    </div>
                )}
            </div>
             {summary && user && (
                <div className="mt-4 pt-4 border-t border-border">
                    <button onClick={handleExportToDocs} disabled={isExporting} className="w-full
                    btn-primary bg-blue-600 flex items-center justify-center gap-2 py-2">
                        {isExporting ? <LoadingSpinner /> : <><DocumentIcon /> Export to Google Docs</>}
                    </button>
                </div>
             )}
        </div>
    </div>
</div>
```

```
        </div>
    );
};


// ===== Connections_22.tsx =====



import React, { useState, useEffect } from 'react';
import { GithubIcon, MapIcon } from '../icons.tsx';
import { useGlobalState } from '../../contexts/GlobalStateContext.tsx';
import { signInWithGoogle } from '../../services/firebaseService.ts';
import { validateToken } from '../../services/authService.ts';
import * as vaultService from '../../services/vaultService.ts';
import { useVaultModal } from '../../contexts/VaultModalContext.tsx';
import { LoadingSpinner } from '../shared/LoadingSpinner.tsx';
import type { GitHubUser } from '../../types.ts';
import { useNotification } from '../../contexts/NotificationContext.tsx';

const GitHubConnection: React.FC = () => {
    const { state, dispatch } = useGlobalState();
    const { user, githubUser } = state;
    const { addNotification } = useNotification();
    const { requestUnlock } = useVaultModal();
    const [tokenInput, setTokenInput] = useState('');
    const [isLoading, setIsLoading] = useState(false);
    const [error, setError] = useState('');

    useEffect(() => {
        const checkStoredToken = async () => {
            if (user && !githubUser && state.vaultState.isUnlocked) {
                setIsLoading(true);
                const token = await vaultService.getDecryptedCredential('github_pat');
                if (token) {
                    try {
                        const githubProfile = await validateToken(token);
                        dispatch({ type: 'SET_GITHUB_USER', payload: githubProfile });
                    } catch (e) {
                        addNotification('Failed to validate stored GitHub token.', 'error');
                    }
                }
                setIsLoading(false);
            }
        };
        checkStoredToken();
    }, [user, githubUser, dispatch, addNotification, state.vaultState.isUnlocked]);

    const handleConnectGitHub = async () => {
        if (!user || !tokenInput.trim()) {
            setError('Token is required.');
            return;
        }
        if (!state.vaultState.isUnlocked) {
            const unlocked = await requestUnlock();
            if (!unlocked) return;
        }

        setIsLoading(true);
        setError('');
        try {
            const githubProfile = await validateToken(tokenInput);
```

```
        await vaultService.saveCredential('github_pat', tokenInput);
        dispatch({ type: 'SET_GITHUB_USER', payload: githubProfile });
        addNotification('GitHub connected successfully!', 'success');
        setTokenInput('');
    } catch (err) {
        setError(err instanceof Error ? `Invalid Token: ${err.message}` : 'Could not
        connect to GitHub.');
    } finally {
        setIsLoading(false);
    }
};

const handleDisconnectGitHub = async () => {
    if (!user) return;
    setIsLoading(true);
    try {
        await vaultService.saveCredential('github_pat', '');
        dispatch({ type: 'SET_GITHUB_USER', payload: null });
        addNotification('GitHub disconnected.', 'info');
    } catch (e) {
        addNotification('Failed to disconnect GitHub.', 'error');
    } finally {
        setIsLoading(false);
    }
};

if (!user) {
    return null;
}

return (
    <div className="bg-surface border border-border rounded-lg p-6">
        <div className="flex items-center justify-between">
            <div className="flex items-center gap-4">
                <div className="w-10 h-10"><GithubIcon /></div>
                <div>
                    <h3 className="text-lg font-bold text-text-primary">GitHub</h3>
                    {githubUser ? (
                         <p className="text-sm text-green-600">Connected as {githubUser.login}</p>
                    ) : (
                         <p className="text-sm text-text-secondary">Not Connected</p>
                    )}
                </div>
            </div>
             {githubUser && (
                <button onClick={handleDisconnectGitHub} className="px-4 py-2 bg-red-500/10
                text-red-600 font-semibold rounded-lg hover:bg-red-500/20">
                    Disconnect
                </button>
             )}
        </div>

        {!githubUser && (
            <div className="mt-4 pt-4 border-t border-border">
                <label htmlFor="github-pat" className="block text-sm font-medium text-text-
                secondary mb-1">Personal Access Token (Classic)</label>
                <div className="flex gap-2">
                    <input id="github-pat" type="password" value={tokenInput} onChange={(e) =>
                    setTokenInput(e.target.value)} placeholder="ghp_..." className="flex-grow p-2
                    bg-background border border-border rounded-md text-sm" />
                     <button onClick={handleConnectGitHub} disabled={isLoading} className="btn-
                     primary px-6 py-2 flex items-center justify-center min-w-[100px]">
```

```
                            {isLoading ? <LoadingSpinner /> : 'Connect'}
                        </button>
                    </div>
                    {error && <p className="text-red-500 text-xs mt-2">{error}</p>}
                </div>
            )}
        </div>
    );
};

const GoogleMapsConnection: React.FC = () => {
    const { state } = useGlobalState();
    const { user } = state;
    const { addNotification } = useNotification();
    const { requestUnlock } = useVaultModal();
    const [apiKey, setApiKey] = useState('');
    const [isSaved, setIsSaved] = useState(false);
    const [isLoading, setIsLoading] = useState(false);

    useEffect(() => {
        const checkKey = async () => {
            if(user && state.vaultState.isUnlocked) {
                const key = await vaultService.getDecryptedCredential('google_maps_api_key');
                setIsSaved(!!key);
            }
        };
        checkKey();
    }, [user, state.vaultState.isUnlocked]);

    const handleSaveKey = async () => {
        if (!apiKey.trim()) return;
        if (!state.vaultState.isUnlocked) {
            const unlocked = await requestUnlock();
            if (!unlocked) return;
        }
        setIsLoading(true);
        try {
            await vaultService.saveCredential('google_maps_api_key', apiKey);
            addNotification('Google Maps API Key saved!', 'success');
            setIsSaved(true);
            setApiKey('');
        } catch(e) {
            addNotification('Failed to save key.', 'error');
        } finally {
            setIsLoading(false);
        }
    }

    if (!user) return null;

    return (
        <div className="bg-surface border border-border rounded-lg p-6">
            <div className="flex items-center gap-4">
                <div className="w-10 h-10"><MapIcon /></div>
                <div>
                    <h3 className="text-lg font-bold text-text-primary">Google Maps Platform</h3>
                    <p className={`text-sm ${isSaved ? 'text-green-600' : 'text-text-
                    secondary'}`}>{isSaved ? 'API Key is saved in vault' : 'Not Connected'}</p>
                </div>
            </div>
             <div className="mt-4 pt-4 border-t border-border">
                <label htmlFor="maps-key" className="block text-sm font-medium text-text-
```

```
                secondary mb-1">API Key</label>
                <div className="flex gap-2">
                    <input id="maps-key" type="password" value={apiKey} onChange={(e) =>
                    setApiKey(e.target.value)} placeholder="AIzaSy..." className="flex-grow p-2 bg-
                    background border border-border rounded-md text-sm" />
                     <button onClick={handleSaveKey} disabled={isLoading} className="btn-primary px-6
                     py-2 flex items-center justify-center min-w-[100px]">
                        {isLoading ? <LoadingSpinner /> : 'Save'}
                    </button>
                </div>
            </div>
        </div>
    )
}

export const Connections: React.FC = () => {
    const { state } = useGlobalState();
    const { user } = state;
    const { addNotification } = useNotification();
    const [isLoading, setIsLoading] = useState(false);

    const handleSignIn = async () => {
        setIsLoading(true);
        try {
            await signInWithGoogle();
            addNotification('Signed in successfully!', 'success');
        } catch (error) {
            addNotification('Failed to sign in.', 'error');
            console.error(error);
        }
        setIsLoading(false);
    };

    return (
        <div className="h-full p-4 sm:p-6 lg:p-8">
            <div className="max-w-4xl mx-auto">
                <header className="mb-8">
                    <h1 className="text-4xl font-extrabold tracking-tight">Connections</h1>
                    <p className="mt-2 text-lg text-text-secondary">Sign in and connect to external
                    services.</p>
                </header>

                {!user ? (
                    <div className="text-center bg-surface p-8 rounded-lg border border-border">
                        <h2 className="text-xl font-bold">Sign In Required</h2>
                        <p className="text-text-secondary my-4">Please sign in with your Google account
                        to manage connections and save your data.</p>
                        <button onClick={handleSignIn} disabled={isLoading} className="btn-primary px-6
                        py-3 flex items-center justify-center gap-2 mx-auto">
                           {isLoading ? <LoadingSpinner/> : 'Sign in with Google'}
                        </button>
                    </div>
                ) : (
                    <div className="space-y-6">
                        <GitHubConnection />
                        <GoogleMapsConnection />
                    </div>
                )}
            </div>
        </div>
    );
};
```

```tsx
// ===== LogicFlowBuilder_27.tsx =====

import React, { useState, useRef, useMemo, useCallback } from 'react';
import { ALL_FEATURES } from './index.ts';
import { FEATURE_TAXONOMY } from '../../services/taxonomyService.ts';
import { generatePipelineCode } from '../../services/aiService.ts';
import type { Feature } from '../../types.ts';
import { MapIcon, SparklesIcon, XMarkIcon } from '../icons.tsx';
import { LoadingSpinner, MarkdownRenderer } from '../shared/index.tsx';

interface Node {
    id: number;
    featureId: string;
    x: number;
    y: number;
}

interface Link {
    from: number;
    to: number;
}

const featuresMap = new Map(ALL_FEATURES.map(f => [f.id, f]));
const taxonomyMap = new Map(FEATURE_TAXONOMY.map(f => [f.id, f]));

const FeaturePaletteItem: React.FC<{ feature: Feature, onDragStart: (e:
React.DragEvent, featureId: string) => void }> = ({ feature, onDragStart }) => (
    <div
        draggable
        onDragStart={e => onDragStart(e, feature.id)}
        className="p-3 rounded-md bg-gray-50 border border-border flex items-center
        gap-3 cursor-grab hover:bg-gray-100 transition-colors"
    >
        <div className="text-primary flex-shrink-0">{feature.icon}</div>
        <div>
            <h4 className="font-bold text-sm text-text-primary">{feature.name}</h4>
            <p className="text-xs text-text-secondary">{feature.category}</p>
        </div>
    </div>
);

const NodeComponent: React.FC<{
    node: Node;
    feature: Feature;
    onMouseDown: (e: React.MouseEvent, id: number) => void;
    onLinkStart: (e: React.MouseEvent, id: number) => void;
    onLinkEnd: (e: React.MouseEvent, id: number) => void;
}> = ({ node, feature, onMouseDown, onLinkStart, onLinkEnd }) => (
    <div
        className="absolute w-52 bg-surface rounded-lg shadow-md border-2 border-border
        cursor-grab active:cursor-grabbing flex flex-col"
        style={{ left: node.x, top: node.y, transform: 'translate(-50%, -50%)' }}
        onMouseDown={e => onMouseDown(e, node.id)}
        onMouseUp={e => onLinkEnd(e, node.id)}
    >
        <div className="p-2 flex items-center gap-2 border-b border-border">
            <div className="w-5 h-5 text-primary">{feature.icon}</div>
            <span className="text-sm font-semibold truncate text-text-
            primary">{feature.name}</span>
        </div>
        <div className="relative p-3 text-xs text-text-secondary min-h-[40px] flex
        items-center justify-center">
```

```
            Workflow Node
            <div
                onMouseDown={e => onLinkStart(e, node.id)}
                className="absolute right-[-9px] top-1/2 -translate-y-1/2 w-4 h-4 bg-primary
                rounded-full border-2 border-surface cursor-crosshair hover:scale-125
                transition-transform"
                title="Drag to connect"
            />
        </div>
    </div>
);

const SVGGrid: React.FC = React.memo(() => (
    <svg width="100%" height="100%" className="absolute inset-0">
        <defs>
            <pattern id="smallGrid" width="10" height="10" patternUnits="userSpaceOnUse">
                <path d="M 10 0 L 0 0 0 10" fill="none" stroke="rgba(0, 0, 0, 0.05)"
                strokeWidth="0.5"/>
            </pattern>
            <pattern id="grid" width="50" height="50" patternUnits="userSpaceOnUse">
                <rect width="50" height="50" fill="url(#smallGrid)"/>
                <path d="M 50 0 L 0 0 0 50" fill="none" stroke="rgba(0, 0, 0, 0.1)"
                strokeWidth="1"/>
            </pattern>
        </defs>
        <rect width="100%" height="100%" fill="url(#grid)" />
    </svg>
));

export const LogicFlowBuilder: React.FC = () => {
    const [nodes, setNodes] = useState<Node[]>([]);
    const [links, setLinks] = useState<Link[]>([]);
    const [draggingNode, setDraggingNode] = useState<{ id: number; offsetX: number;
    offsetY: number } | null>(null);
    const [linking, setLinking] = useState<{ from: number; fromPos: { x: number; y:
    number }; toPos: { x: number; y: number } } | null>(null);
    const [generatedCode, setGeneratedCode] = useState('');
    const [isGenerating, setIsGenerating] = useState(false);
    const canvasRef = useRef<HTMLDivElement>(null);

    const handleGenerateCode = useCallback(async () => {
        setIsGenerating(true);
        setGeneratedCode('');

        const sortedNodeIds: number[] = [];
        const inDegree = new Map<number, number>();
        nodes.forEach(node => inDegree.set(node.id, 0));
        links.forEach(link => inDegree.set(link.to, (inDegree.get(link.to) || 0) + 1));

        const queue = nodes.filter(node => inDegree.get(node.id) === 0).map(n => n.id);

        while(queue.length > 0) {
            const u = queue.shift()!;
            sortedNodeIds.push(u);
            links.filter(l => l.from === u).forEach(l => {
                inDegree.set(l.to, (inDegree.get(l.to) || 0) - 1);
                if(inDegree.get(l.to) === 0) queue.push(l.to);
            })
        }

        const flowDescription = sortedNodeIds.map((id, index) => {
            const node = nodes.find(n => n.id === id)!;
```

```
        const featureInfo = taxonomyMap.get(node.featureId);
        return `Step ${index + 1}: Execute the '${featureInfo?.name}' tool. Description:
        ${featureInfo?.description}. Inputs: ${featureInfo?.inputs}.`;
    }).join('\n');

    try {
        const code = await generatePipelineCode(flowDescription);
        setGeneratedCode(code);
    } catch (e) {
        setGeneratedCode(`// Error generating code: ${e instanceof Error ? e.message :
        'Unknown error'}`);
    } finally {
        setIsGenerating(false);
    }

}, [nodes, links]);

const handleDragStart = (e: React.DragEvent, featureId: string) => {
    e.dataTransfer.setData('application/json', JSON.stringify({ featureId }));
};

const handleDrop = (e: React.DragEvent) => {
    e.preventDefault();
    if (!canvasRef.current) return;
    const { featureId } = JSON.parse(e.dataTransfer.getData('application/json'));
    const canvasRect = canvasRef.current.getBoundingClientRect();
    const newNode: Node = {
        id: Date.now(),
        featureId,
        x: e.clientX - canvasRect.left,
        y: e.clientY - canvasRect.top,
    };
    setNodes(prev => [...prev, newNode]);
};

const handleNodeMouseDown = (e: React.MouseEvent, id: number) => {
    const node = nodes.find(n => n.id === id);
    if (!node || (e.target as HTMLElement).title === 'Drag to connect') return;
    const canvasRect = canvasRef.current!.getBoundingClientRect();
    setDraggingNode({ id, offsetX: e.clientX - canvasRect.left - node.x, offsetY:
    e.clientY - canvasRect.top - node.y });
};

const handleCanvasMouseMove = (e: React.MouseEvent) => {
    if (!canvasRef.current) return;
    const canvasRect = canvasRef.current.getBoundingClientRect();
    const mouseX = e.clientX - canvasRect.left;
    const mouseY = e.clientY - canvasRect.top;

    if (draggingNode) {
        setNodes(nodes.map(n => n.id === draggingNode.id ? { ...n, x: mouseX -
        draggingNode.offsetX, y: mouseY - draggingNode.offsetY } : n));
    }
    if (linking) {
        setLinking({ ...linking, toPos: { x: mouseX, y: mouseY } });
    }
};

const handleCanvasMouseUp = () => {
    setDraggingNode(null);
    setLinking(null);
};
```

```
const handleLinkStart = (e: React.MouseEvent, id: number) => {
    e.stopPropagation();
    const fromNode = nodes.find(n => n.id === id);
    if (!fromNode) return;
    setLinking({ from: id, fromPos: { x: fromNode.x, y: fromNode.y }, toPos: { x:
    fromNode.x, y: fromNode.y } });
};

const handleLinkEnd = (e: React.MouseEvent, id: number) => {
    e.stopPropagation();
    if (linking && linking.from !== id) {
        setLinks(prev => [...prev, { from: linking.from, to: id }]);
    }
    setLinking(null);
};

const nodePositions = useMemo(() => new Map(nodes.map(n => [n.id, { x: n.x, y:
n.y }])), [nodes]);

return (
    <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
        <header className="mb-6 flex justify-between items-start">
            <div>
                <h1 className="text-3xl font-bold flex items-center"><MapIcon /><span
                className="ml-3">Logic Flow Builder</span></h1>
                <p className="text-text-secondary mt-1">Visually build application logic flows
                and generate pipeline code.</p>
            </div>
            <button onClick={handleGenerateCode} disabled={isGenerating || nodes.length ===
            0} className="btn-primary flex items-center gap-2 px-4 py-2">
                <SparklesIcon /> {isGenerating ? 'Generating...' : 'Generate Code'}
            </button>
        </header>
        <div className="flex-grow flex gap-6 min-h-0">
            <aside className="w-72 flex-shrink-0 bg-surface border border-border p-4
            rounded-lg flex flex-col">
                <h3 className="font-bold mb-3 text-lg">Features</h3>
                <div className="flex-grow overflow-y-auto space-y-3 pr-2">
                    {ALL_FEATURES.map(feature => <FeaturePaletteItem key={feature.id}
                    feature={feature} onDragStart={handleDragStart} />)}
                </div>
            </aside>
            <main
                ref={canvasRef}
                className="flex-grow relative bg-background border-2 border-dashed border-border
                rounded-lg overflow-hidden"
                onDrop={handleDrop}
                onDragOver={e => e.preventDefault()}
                onMouseMove={handleCanvasMouseMove}
                onMouseUp={handleCanvasMouseUp}
                onMouseLeave={handleCanvasMouseUp}
            >
                <SVGGrid />
                <svg width="100%" height="100%" className="absolute inset-0 pointer-events-
                none">
                    {links.map((link, i) => {
                        const fromNode = nodePositions.get(link.from);
                        const toNode = nodePositions.get(link.to);
                        if (!fromNode || !toNode) return null;
                        return <line key={i} x1={fromNode.x} y1={fromNode.y} x2={toNode.x} y2={toNode.y}
                        stroke="var(--color-primary)" strokeWidth="2" markerEnd="url(#arrow)" />;
                    })}
```

```
                        {linking && <line x1={linking.fromPos.x} y1={linking.fromPos.y}
                        x2={linking.toPos.x} y2={linking.toPos.y} stroke="var(--color-primary)"
                        strokeWidth="2" strokeDasharray="5,5" />}
                        <defs><marker id="arrow" viewBox="0 0 10 10" refX="8" refY="5" markerWidth="6"
                        markerHeight="6" orient="auto-start-reverse"><path d="M 0 0 L 10 5 L 0 10 z"
                        fill="var(--color-primary)" /></marker></defs>
                    </svg>
                    {nodes.map(node => {
                        const feature = featuresMap.get(node.featureId);
                        return feature ? <NodeComponent key={node.id} node={node} feature={feature}
                        onMouseDown={handleNodeMouseDown} onLinkStart={handleLinkStart}
                        onLinkEnd={handleLinkEnd} /> : null;
                    })}
                </main>
            </div>
            {(isGenerating || generatedCode) && (
                <div className="fixed inset-0 bg-gray-900/80 backdrop-blur-sm z-50 flex items-
                center justify-center" onClick={() => setGeneratedCode('')}>
                    <div className="w-full max-w-3xl h-3/4 bg-surface border border-border rounded-
                    lg shadow-2xl p-6 flex flex-col" onClick={e => e.stopPropagation()}>
                        <div className="flex justify-between items-center mb-4">
                            <h2 className="text-xl font-bold">Generated Pipeline Code</h2>
                            <button onClick={() => setGeneratedCode('')}><XMarkIcon/></button>
                        </div>
                        <div className="flex-grow bg-background border border-border rounded-md
                        overflow-auto">
                            {isGenerating && !generatedCode ? <div className="flex justify-center items-
                            center h-full"><LoadingSpinner /></div> : <MarkdownRenderer
                            content={'```javascript\n' + generatedCode + '\n```'} />}
                        </div>
                    </div>
                </div>
            )}
        </div>
    );
};

// ===== VisualGitTree_27.tsx =====

import React, { useState, useCallback, useEffect, useMemo } from 'react';
import { GitBranchIcon, ArrowDownTrayIcon } from '../icons.tsx';
import { generateChangelogFromLogStream } from '../../services/aiService.ts';
import { LoadingSpinner, MarkdownRenderer } from '../shared/index.tsx';
import { downloadFile } from '../../services/fileUtils.ts';

const exampleLog = `* commit 3a4b5c6d7e8f9g0h1i2j3k4l5m6n7o8p9q0r (HEAD -> main,
origin/main)
|\\  Merge: 1a2b3c4 2d3e4f5
| | Author: Dev One <dev.one@example.com>
| | Date:   Mon Jul 15 11:30:00 2024 -0400
| |
| |     feat: Implement collapsible sidebar navigation
| |
* | commit 2d3e4f5g6h7i8j9k0l1m2n3o4p5q6r7s8t9u (feature/new-sidebar)
| | Author: Dev Two <dev.two@example.com>
| | Date:   Mon Jul 15 10:00:00 2024 -0400
| |
| |     feat: Add icons to sidebar items
| |
* | commit 1a2b3c4d5e6f7g8h9i0j1k2l3m4n5o6p7q8r
|/  Author: Dev One <dev.one@example.com>
|   Date:   Fri Jul 12 16:45:00 2024 -0400
```

```
|
|       fix: Correct user authentication bug`;

const CommitGraph = ({ logInput }: { logInput: string }) => {
    const commits = useMemo(() => {
        const lines = logInput.split('\n');
        const parsedCommits: any[] = [];
        let currentCommit: any = null;

        lines.forEach(line => {
            const commitMatch = line.match(/^.?[\\|/ ]*\* commit (\w+)(.*)/);
            if (commitMatch) {
                if (currentCommit) parsedCommits.push(currentCommit);
                currentCommit = {
                    hash: commitMatch[1],
                    shortHash: commitMatch[1].substring(0, 7),
                    refs: commitMatch[2].trim(),
                    message: '',
                    author: '',
                };
            } else if (currentCommit) {
                 if (line.includes('Author:')) currentCommit.author =
                 line.split('Author:')[1].trim();
                 else if (line.trim().length > 0 && !line.match(/^[\\|/ ]*[\\|/ ]/)) {
                     currentCommit.message += line.trim() + ' ';
                 }
            }
        });
        if (currentCommit) parsedCommits.push(currentCommit);

        return parsedCommits.map((c, i) => ({ ...c, x: 50, y: 50 + i * 60 }));
    }, [logInput]);

    return (
         <svg width="100%" height={50 + commits.length * 60} className="min-h-[200px]">
            {commits.map((commit, i) => {
                const parent = commits[i + 1];
                return (
                    <g key={commit.hash}>
                        {parent && <line x1={commit.x} y1={commit.y} x2={parent.x} y2={parent.y}
                        stroke="var(--color-border)" strokeWidth="2" />}
                        <g className="group cursor-pointer">
                            <circle cx={commit.x} cy={commit.y} r="8" fill="var(--color-primary)"
                            stroke="var(--color-surface)" strokeWidth="3" />
                            <foreignObject x={commit.x + 20} y={commit.y - 25} width="350" height="50">
                                <div className="text-sm p-1">
                                    <p className="font-bold truncate text-text-primary">{commit.message}</p>
                                    <p className="text-xs text-text-secondary font-mono">{commit.shortHash} <span
                                    className="text-amber-600">{commit.refs}</span></p>
                                </div>
                            </foreignObject>
                            <title>{`Commit: ${commit.hash}\nAuthor:
                            ${commit.author}\n\n${commit.message}`}</title>
                        </g>
                    </g>
                );
            })}
        </svg>
    );
};

export const VisualGitTree: React.FC<{ logInput?: string }> = ({ logInput:
```

```
initialLogInput }) => {
    const [logInput, setLogInput] = useState(initialLogInput || exampleLog);
    const [analysis, setAnalysis] = useState('');
    const [isLoading, setIsLoading] = useState(false);
    const [error, setError] = useState('');

    const handleAnalyze = useCallback(async (logToAnalyze: string) => {
        if (!logToAnalyze.trim()) {
            setError('Please paste git log output.');
            return;
        }
        setIsLoading(true);
        setError('');
        setAnalysis('');
        try {
            const stream = generateChangelogFromLogStream(logToAnalyze);
            let fullResponse = '';
            for await (const chunk of stream) {
                fullResponse += chunk;
                setAnalysis(fullResponse);
            }
        } catch (err) {
            const errorMessage = err instanceof Error ? err.message : 'An unknown error
            occurred.';
            setError(`Failed to analyze log: ${errorMessage}`);
        } finally {
            setIsLoading(false);
        }
    }, []);

    useEffect(() => {
        if (initialLogInput) {
            setLogInput(initialLogInput);
            handleAnalyze(initialLogInput);
        }
    }, [initialLogInput, handleAnalyze]);

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
            <header className="mb-6">
                <h1 className="text-3xl font-bold flex items-center">
                    <GitBranchIcon />
                    <span className="ml-3">Visual Git Tree</span>
                </h1>
                <p className="text-text-secondary mt-1">Paste your `git log --graph` output to
                visualize the history and get an AI summary.</p>
            </header>
            <div className="flex-grow grid grid-cols-1 lg:grid-cols-2 gap-6 h-full overflow-
            hidden">
                <div className="flex flex-col h-full">
                    <label htmlFor="log-input" className="text-sm font-medium text-text-secondary
                    mb-2">Git Log Output</label>
                    <textarea
                        id="log-input"
                        value={logInput}
                        onChange={(e) => setLogInput(e.target.value)}
                        placeholder="Paste your git log output here..."
                        className="flex-grow p-4 bg-surface border border-border rounded-md resize-none
                        font-mono text-sm"
                    />
                    <button
                        onClick={() => handleAnalyze(logInput)}
```

```
                        disabled={isLoading}
                        className="btn-primary mt-4 w-full flex items-center justify-center px-6 py-3"
                    >
                        {isLoading ? <LoadingSpinner /> : 'Analyze & Summarize'}
                    </button>
                </div>
                <div className="flex flex-col h-full gap-4">
                    <div className="flex flex-col h-1/2">
                        <label className="text-sm font-medium text-text-secondary mb-2">Commit
                        Graph</label>
                        <div className="flex-grow p-2 bg-surface border border-border rounded-md
                        overflow-auto">
                            <CommitGraph logInput={logInput} />
                        </div>
                    </div>
                     <div className="flex flex-col h-1/2">
                        <div className="flex justify-between items-center mb-2">
                            <label className="text-sm font-medium text-text-secondary">AI Summary</label>
                            {analysis && !isLoading && (
                                <button onClick={() => downloadFile(analysis, 'summary.md', 'text/markdown')}
                                className="flex items-center gap-1 px-3 py-1 bg-gray-100 text-xs rounded-md
                                hover:bg-gray-200">
                                    <ArrowDownTrayIcon className="w-4 h-4"/> Download Summary
                                </button>
                            )}
                        </div>
                        <div className="flex-grow p-4 bg-background border border-border rounded-md
                        overflow-y-auto">
                            {isLoading && <div className="flex items-center justify-center
                            h-full"><LoadingSpinner /></div>}
                            {error && <p className="text-red-500">{error}</p>}
                            {analysis && !isLoading && <MarkdownRenderer content={analysis} />}
                            {!isLoading && !analysis && !error && <div className="text-text-secondary h-full
                            flex items-center justify-center">AI summary will appear here.</div>}
                        </div>
                    </div>
                </div>
            </div>
        </div>
    );
};

// ===== SnippetVault_27.tsx =====

import React, { useState, useEffect, useMemo } from 'react';
import { LockClosedIcon, SparklesIcon, TrashIcon, ClipboardDocumentIcon,
ArrowDownTrayIcon } from '../icons.tsx';
import { useLocalStorage } from '../../hooks/useLocalStorage.ts';
import { enhanceSnippetStream, generateTagsForCode } from
'../../services/aiService.ts';
import { LoadingSpinner } from '../shared/index.tsx';
import { downloadFile } from '../../services/fileUtils.ts';
import { useNotification } from '../../contexts/NotificationContext.tsx';

interface Snippet {
    id: number; name: string; code: string; language: string; tags: string[];
}

const langToExt: Record<string, string> = {
    javascript: 'js',
    typescript: 'ts',
    python: 'py',
```

```
    css: 'css',
    html: 'html',
    json: 'json',
    markdown: 'md',
    plaintext: 'txt',
};

export const SnippetVault: React.FC = () => {
    const [snippets, setSnippets] = useLocalStorage<Snippet[]>('devcore_snippets',
    [{ id: 1, name: 'React Hook Boilerplate', language: 'javascript', code: `import
    { useState } from 'react';\n\nconst useCustomHook = () => {\n  const [value,
    setValue] = useState(null);\n  return { value, setValue };\n};`, tags: ['react',
    'hook'] }]);
    const [activeSnippet, setActiveSnippet] = useState<Snippet | null>(null);
    const [isEnhancing, setIsEnhancing] = useState(false);
    const [searchTerm, setSearchTerm] = useState('');
    const [isEditingName, setIsEditingName] = useState(false);
    const { addNotification } = useNotification();

    const filteredSnippets = useMemo(() => {
        if (!searchTerm) return snippets;
        const lowerSearch = searchTerm.toLowerCase();
        return snippets.filter((s: Snippet) =>
            s.name.toLowerCase().includes(lowerSearch) ||
            s.code.toLowerCase().includes(lowerSearch) ||
            (s.tags && s.tags.some(t => t.toLowerCase().includes(lowerSearch)))
        );
    }, [snippets, searchTerm]);

    useEffect(() => {
        if (!activeSnippet && filteredSnippets.length > 0)
        setActiveSnippet(filteredSnippets[0]);
        if (activeSnippet) setActiveSnippet(snippets.find((s: Snippet) => s.id ===
        activeSnippet.id) || null);
    }, [snippets, activeSnippet, filteredSnippets]);

    const updateSnippet = (snippet: Snippet) => {
        setSnippets(snippets.map((s: Snippet) => s.id === snippet.id ? snippet : s));
        setActiveSnippet(snippet);
    };

    const handleEnhance = async () => {
        if (!activeSnippet) return;
        setIsEnhancing(true);
        try {
            const stream = enhanceSnippetStream(activeSnippet.code);
            let fullResponse = '';
            for await (const chunk of stream) {
                fullResponse += chunk;
                updateSnippet({ ...activeSnippet, code: fullResponse.replace(/^```(?:\w+\n)?/,
                '').replace(/```$/, '') });
            }
        } finally { setIsEnhancing(false); }
    };

    const handleAiTagging = async (snippet: Snippet) => {
        if (!snippet.code.trim()) return;
        try {
            const suggestedTags = await generateTagsForCode(snippet.code);
            const newTags = [...new Set([...(snippet.tags || []), ...suggestedTags])];
            updateSnippet({...snippet, tags: newTags});
            addNotification('AI tags added!', 'success');
```

```
    } catch(e) {
        console.error("AI tagging failed:", e);
        addNotification('AI tagging failed.', 'error');
    }
};

const handleAddNew = () => {
    const newSnippet: Snippet = { id: Date.now(), name: 'New Snippet', language:
    'plaintext', code: '', tags: [] };
    setSnippets([...snippets, newSnippet]);
    setActiveSnippet(newSnippet);
};

const handleDelete = (id: number) => {
    setSnippets(snippets.filter((s: Snippet) => s.id !== id));
    if(activeSnippet?.id === id) setActiveSnippet(filteredSnippets.length > 1 ?
    filteredSnippets[0] : null);
};

const handleDownload = () => {
    if(!activeSnippet) return;
    const extension = langToExt[activeSnippet.language] || 'txt';
    const filename = `${activeSnippet.name.replace(/\s/g, '_')}.${extension}`;
    downloadFile(activeSnippet.code, filename);
}

const handleNameChange = (e: React.ChangeEvent<HTMLInputElement>) => {
    if (activeSnippet) updateSnippet({...activeSnippet, name: e.target.value});
};

const handleTagsChange = (e: React.KeyboardEvent<HTMLInputElement>) => {
    if (e.key === 'Enter' && activeSnippet) {
        const newTag = e.currentTarget.value.trim();
        if (newTag && !activeSnippet.tags.includes(newTag)) {
            updateSnippet({...activeSnippet, tags: [...(activeSnippet.tags ?? []),
            newTag]});
        }
        e.currentTarget.value = '';
    }
};

return (
    <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
        <header className="mb-6"><h1 className="text-3xl font-bold flex items-
        center"><LockClosedIcon /><span className="ml-3">Snippet Vault</span></h1><p
        className="text-text-secondary mt-1">Store, search, tag, and enhance your
        reusable code snippets with AI.</p></header>
        <div className="flex-grow flex gap-6 min-h-0">
            <aside className="w-1/3 bg-surface border border-border p-4 rounded-lg flex
            flex-col">
                <input type="text" placeholder="Search snippets..." value={searchTerm}
                onChange={e => setSearchTerm(e.target.value)} className="w-full px-3 py-1.5 mb-3
                rounded-md bg-background border border-border text-sm"/>
                <ul className="space-y-2 flex-grow overflow-y-auto
                pr-2">{filteredSnippets.map((s: Snippet) => (<li key={s.id} className="group
                flex items-center justify-between"><button onClick={() => setActiveSnippet(s)}
                className={`w-full text-left px-3 py-2 rounded-md ${activeSnippet?.id === s.id ?
                'bg-primary/10 text-primary' : 'hover:bg-gray-100 dark:hover:bg-
                slate-700'}`}>{s.name}</button><div className="flex opacity-0 group-
                hover:opacity-100 transition-opacity"><button onClick={() =>
                {navigator.clipboard.writeText(s.code); addNotification("Copied snippet!",
                "success")}} className="ml-2 p-1 text-text-secondary hover:text-primary"
```

```
                    title="Copy"><ClipboardDocumentIcon /></button><button onClick={() =>
                    handleDelete(s.id)} className="ml-2 p-1 text-text-secondary hover:text-red-500"
                    title="Delete"><TrashIcon/></button></div></li>))}</ul>
                    <div className="mt-4 pt-4 border-t border-border"><button onClick={handleAddNew}
                    className="btn-primary w-full text-sm py-2">Add New Snippet</button></div>
                </aside>
                <main className="w-2/3 flex flex-col">
                    {activeSnippet ? (<>
                        <div className="flex justify-between items-center mb-2">
                            {isEditingName ? <input type="text" value={activeSnippet.name}
                            onChange={handleNameChange} onBlur={() => setIsEditingName(false)} autoFocus
                            className="text-lg font-bold bg-gray-100 dark:bg-slate-700 rounded px-2"/> : <h3
                            onDoubleClick={() => setIsEditingName(true)} className="text-lg font-bold
                            cursor-pointer">{activeSnippet.name}</h3>}
                            <div className="flex gap-2">
                                <button onClick={() => handleAiTagging(activeSnippet)} className="flex items-
                                center gap-2 px-3 py-1 bg-teal-500/80 text-white font-bold text-xs rounded-
                                md"><SparklesIcon /> AI Tag</button>
                                <button onClick={handleEnhance} disabled={isEnhancing} className="flex items-
                                center gap-2 px-3 py-1 bg-purple-500/80 text-white font-bold text-xs rounded-md
                                disabled:bg-gray-400"><SparklesIcon /> AI Enhance</button>
                                <button onClick={handleDownload} className="flex items-center gap-1 px-3 py-1
                                bg-gray-100 dark:bg-slate-700 text-xs rounded-md"><ArrowDownTrayIcon
                                className="w-4 h-4"/> Download</button>
                            </div>
                        </div>
                        <textarea value={activeSnippet.code} onChange={e =>
                        updateSnippet({...activeSnippet, code: e.target.value})} className="flex-grow
                        p-4 bg-surface border border-border rounded-md resize-none font-mono text-sm
                        focus:ring-2 focus:ring-primary focus:outline-none"/>
                        <div className="mt-2 text-xs text-text-secondary">
                           <div className="flex items-center gap-2 flex-wrap">
                             <span className="font-bold">Tags:</span> {(activeSnippet.tags ?? []).map(t =>
                             <span key={t} className="bg-gray-200 dark:bg-slate-700 px-2 py-0.5 rounded-
                             full">{t}</span>)}
                             <input type="text" placeholder="+ Add tag" onKeyDown={handleTagsChange}
                             className="bg-transparent border-b border-border focus:outline-none
                             focus:border-primary w-24 text-xs px-1"/>
                           </div>
                        </div>
                    </>) : (<div className="flex-grow flex items-center justify-center bg-background
                    border border-border rounded-lg text-text-secondary">Select a snippet or create
                    a new one.</div>)}
                </main>
            </div>
        </div>
    );
};

// ===== RegexSandbox_27.tsx =====

import React, { useState, useMemo, useCallback, useEffect } from 'react';
import { generateRegExStream } from '../../services/aiService.ts';
import { BeakerIcon } from '../icons.tsx';
import { LoadingSpinner } from '../shared/index.tsx';

const commonPatterns = [
    { name: 'Email', pattern: '/[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\\.[a-zA-Z]{2,}/g'
    },
    { name: 'URL', pattern: '/https?:\\/\\/(www\\.)?[-a-zA-Z0-
    9@:%._\\+~#=]{1,256}\\.[a-zA-Z0-9()]{1,6}\\b([-a-zA-Z0-9()@:%_\\+.~#?&//=]*)/g'
    },
```

```
    { name: 'IPv4 Address', pattern:
    '/((25[0-5]|(2[0-4]|1\\d|[1-9]|)\\d)\\.?\\b){4}/g' },
    { name: 'Date (YYYY-MM-DD)', pattern: '/\\d{4}-\\d{2}-\\d{2}/g' },
];

const CheatSheet = () => (
    <div className="bg-surface border border-border p-4 rounded-lg">
        <h3 className="text-lg font-bold mb-2">Regex Cheat Sheet</h3>
        <div className="grid grid-cols-2 gap-x-4 gap-y-1 text-xs font-mono">
            <p><span className="text-primary">.</span> - Any character</p>
            <p><span className="text-primary">\d</span> - Any digit</p>
            <p><span className="text-primary">\w</span> - Word character</p>
            <p><span className="text-primary">\s</span> - Whitespace</p>
            <p><span className="text-primary">[abc]</span> - a, b, or c</p>
            <p><span className="text-primary">[^abc]</span> - Not a, b, or c</p>
            <p><span className="text-primary">*</span> - 0 or more</p>
            <p><span className="text-primary">+</span> - 1 or more</p>
            <p><span className="text-primary">?</span> - 0 or one</p>
            <p><span className="text-primary">^</span> - Start of string</p>
            <p><span className="text-primary">$</span> - End of string</p>
            <p><span className="text-primary">\b</span> - Word boundary</p>
        </div>
    </div>
);

export const RegexSandbox: React.FC<{ initialPrompt?: string }> = ({
initialPrompt }) => {
    const [pattern, setPattern] =
    useState<string>('/\\b([A-Z][a-z]+)\\s(\\w+)\\b/g');
    const [testString, setTestString] = useState<string>('The quick Brown Fox jumps
    over the Lazy Dog.');
    const [aiPrompt, setAiPrompt] = useState<string>(initialPrompt || 'find
    capitalized words and the word after');
    const [isAiLoading, setIsAiLoading] = useState<boolean>(false);

    const { matches, error } = useMemo(() => {
        try {
            const patternParts = pattern.match(/^\/(.*)\/([gimyus]*)$/);
            if (!patternParts) return { matches: null, error: 'Invalid regex literal. Use
            /pattern/flags.' };
            const [, regexBody, regexFlags] = patternParts;
            const regex = new RegExp(regexBody, regexFlags);
            return { matches: [...testString.matchAll(regex)], error: null };
        } catch (e) { return { matches: null, error: e instanceof Error ? e.message :
        'Unknown error.' }; }
    }, [pattern, testString]);

    const handleGenerateRegex = useCallback(async (p: string) => {
        if (!p) return;
        setIsAiLoading(true);
        try {
            const stream = generateRegExStream(p);
            let fullResponse = '';
            for await (const chunk of stream) { fullResponse += chunk; }
            setPattern(fullResponse.trim().replace(/^`+|`+$/g, ''));
        } finally { setIsAiLoading(false); }
    }, []);

    useEffect(() => { if (initialPrompt) handleGenerateRegex(initialPrompt); },
    [initialPrompt, handleGenerateRegex]);

    const highlightedString = useMemo(() => {
```

```
    if (!matches || matches.length === 0 || error) return testString;
    let lastIndex = 0;
    const parts: (string | JSX.Element)[] = [];
    matches.forEach((match, i) => {
        if (match.index === undefined) return;
        parts.push(testString.substring(lastIndex, match.index));
        parts.push(<mark key={i} className="bg-primary/20 text-primary rounded
        px-1">{match[0]}</mark>);
        lastIndex = match.index + match[0].length;
    });
    parts.push(testString.substring(lastIndex));
    return parts;
}, [matches, testString, error]);

return (
    <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
        <header className="mb-6"><h1 className="text-3xl font-bold flex items-
        center"><BeakerIcon /><span className="ml-3">RegEx Sandbox</span></h1><p
        className="text-text-secondary mt-1">Test your regular expressions and generate
        them with AI.</p></header>
        <div className="flex-grow grid grid-cols-1 lg:grid-cols-3 gap-6 min-h-0">
            <div className="lg:col-span-2 flex flex-col gap-4">
                <div className="flex gap-2"><input type="text" value={aiPrompt} onChange={(e) =>
                setAiPrompt(e.target.value)} placeholder="Describe the pattern to find..."
                className="flex-grow px-3 py-1.5 rounded-md bg-surface border border-border
                text-sm focus:ring-2 focus:ring-primary" /><button onClick={() =>
                handleGenerateRegex(aiPrompt)} disabled={isAiLoading} className="btn-primary
                px-4 py-1.5 flex items-center">{isAiLoading ? <LoadingSpinner/> :
                'Generate'}</button></div>
                <div><label htmlFor="regex-pattern" className="text-sm font-medium text-text-
                secondary">Regular Expression</label><input id="regex-pattern" type="text"
                value={pattern} onChange={(e) => setPattern(e.target.value)} className={`w-full
                mt-1 px-3 py-2 rounded-md bg-surface border ${error ? 'border-red-500' :
                'border-border'} font-mono text-sm focus:ring-2 focus:ring-primary`} />{error &&
                <p className="text-red-500 text-xs mt-1">{error}</p>}</div>
                <div className="flex flex-col flex-grow min-h-0"><label htmlFor="test-string"
                className="text-sm font-medium text-text-secondary">Test String</label><textarea
                id="test-string" value={testString} onChange={(e) =>
                setTestString(e.target.value)} className="w-full mt-1 p-3 rounded-md bg-surface
                border border-border font-mono text-sm resize-y h-32" /><div className="mt-2 p-3
                bg-background rounded-md border border-border min-h-[50px] whitespace-pre-
                wrap">{highlightedString}</div></div>
                <div className="flex-shrink-0"><h3 className="text-lg font-bold">Match Groups
                ({matches?.length || 0})</h3><div className="mt-2 p-2 bg-surface rounded-md
                overflow-y-auto max-h-48 font-mono text-xs border border-border">{matches &&
                matches.length > 0 ? (matches.map((match, i) => (<details key={i} className="p-2
                border-b border-border"><summary className="cursor-pointer text-green-700">Match
                {i + 1}: "{match[0]}"</summary><div className="pl-4
                mt-1">{Array.from(match).map((group, gIndex) => <p key={gIndex} className="text-
                text-secondary">Group {gIndex}: <span className="text-
                amber-700">{String(group)}</span></p>)}</div></details>))) : (<p
                className="text-text-secondary text-sm p-2">No matches found.</p>)}</div></div>
            </div>
            <div className="lg:col-span-1 space-y-4">
                <CheatSheet />
                <div className="bg-surface border border-border p-4 rounded-lg">
                    <h3 className="text-lg font-bold mb-2">Common Patterns</h3>
                    <div className="flex flex-col items-start gap-2">
                        {commonPatterns.map(p => (
                            <button key={p.name} onClick={() => setPattern(p.pattern)} className="text-left
                            text-sm text-primary hover:underline">
                                {p.name}
```

```
                                </button>
                            ))}
                        </div>
                    </div>
                </div>
            </div>
        </div>
    );
};

// ===== AiFeatureBuilder_27.tsx =====



import React, { useState, useCallback, useEffect } from 'react';
import type { GeneratedFile } from '../../types.ts';
import { generateFeature, generateFullStackFeature, generateUnitTestsStream,
generateCommitMessageStream, generateDockerfile } from
'../../services/aiService.ts';
import { deployCloudFunction, deployFirestoreRules } from
'../../services/gcpService.ts';
import { saveFile, getAllFiles, clearAllFiles } from
'../../services/dbService.ts';
import { useNotification } from '../../contexts/NotificationContext.tsx';
import { CpuChipIcon, DocumentTextIcon, BeakerIcon, GitBranchIcon, CloudIcon,
PaperAirplaneIcon } from '../icons.tsx';
import { LoadingSpinner, MarkdownRenderer } from '../shared/index.tsx';

type SupplementalTab = 'TESTS' | 'COMMIT' | 'DEPLOYMENT' | 'CODE';
type OutputTab = GeneratedFile | SupplementalTab;

export const AiFeatureBuilder: React.FC = () => {
    const [prompt, setPrompt] = useState<string>('A simple "Hello World" React
    component with a button that shows an alert.');
    const [framework] = useState('React');
    const [styling] = useState('Tailwind CSS');
    const [includeBackend, setIncludeBackend] = useState(false);

    const [generatedFiles, setGeneratedFiles] = useState<GeneratedFile[]>([]);
    const [unitTests, setUnitTests] = useState<string>('');
    const [commitMessage, setCommitMessage] = useState<string>('');
    const [dockerfile, setDockerfile] = useState<string>('');

    const [activeTab, setActiveTab] = useState<OutputTab>('CODE');
    const [isLoading, setIsLoading] = useState<boolean>(false);
    const [isDeploying, setIsDeploying] = useState(false);
    const [error, setError] = useState<string>('');
    const { addNotification } = useNotification();

    useEffect(() => {
        const loadFiles = async () => {
            const files = await getAllFiles();
            setGeneratedFiles(files);
            if (files.length > 0) setActiveTab(files[0]);
        };
        loadFiles();
    }, []);

    const handleGenerate = useCallback(async () => {
        if (!prompt.trim()) { setError('Please enter a feature description.'); return; }
        setIsLoading(true);
        setError('');
```

```
    await clearAllFiles();
    setGeneratedFiles([]); setUnitTests(''); setCommitMessage('');
    setDockerfile(''); setActiveTab('CODE');

    try {
        const resultFiles = includeBackend
            ? await generateFullStackFeature(prompt, framework, styling)
            : await generateFeature(prompt, framework, styling);

        for (const file of resultFiles) { await saveFile(file); }
        setGeneratedFiles(resultFiles);

        if (resultFiles.length > 0) {
            const componentFile = resultFiles.find(f => f.filePath.endsWith('.tsx') ||
            f.filePath.endsWith('.jsx'));
            setActiveTab(componentFile || resultFiles[0]);

            const testStream = generateUnitTestsStream(componentFile?.content ||
            resultFiles[0].content);
            const diffContext = resultFiles.map(f => `File:
            ${f.filePath}\n\n${f.content}`).join('\n---\n');
            const commitStream = generateCommitMessageStream(diffContext);

            let tests = ''; for await (const chunk of testStream) { tests += chunk;
            setUnitTests(tests); }
            let commit = ''; for await (const chunk of commitStream) { commit += chunk;
            setCommitMessage(commit); }

            if (!includeBackend) {
                const dockerfileStream = generateDockerfile(framework);
                let docker = ''; for await (const chunk of dockerfileStream) { docker += chunk;
                setDockerfile(docker); }
            }
        }
    } catch (err) {
        setError(err instanceof Error ? err.message : 'Failed to generate feature.');
    } finally {
        setIsLoading(false);
    }
}, [prompt, framework, styling, includeBackend]);

const handleDeploy = async () => {
    if (!includeBackend || generatedFiles.length === 0) return;
    setIsDeploying(true);
    try {
        const backendFile = generatedFiles.find(f =>
        f.filePath.includes('functions/index.js'));
        const rulesFile = generatedFiles.find(f =>
        f.filePath.includes('firestore.rules'));

        if (backendFile) {
            addNotification('Deploying Cloud Function...', 'info');
            await deployCloudFunction(backendFile.content, 'myGeneratedFunction');
            addNotification('Cloud Function deployment initiated.', 'success');
        }
        if (rulesFile) {
            addNotification('Deploying Firestore Rules...', 'info');
            await deployFirestoreRules(rulesFile.content);
            addNotification('Firestore Rules deployment initiated.', 'success');
        }
    } catch(e) {
        addNotification(e instanceof Error ? e.message : 'Deployment failed', 'error');
```

```
    } finally {
        setIsDeploying(false);
    }
}

const renderContent = () => {
    if (typeof activeTab === 'string') {
        switch (activeTab) {
            case 'TESTS': return <MarkdownRenderer content={unitTests} />;
            case 'COMMIT': return <pre className="w-full h-full p-4 whitespace-pre-wrap
            font-sans text-sm">{commitMessage}</pre>;
            case 'DEPLOYMENT': return <MarkdownRenderer content={dockerfile} />;
            default: return <div className="p-4">Select a file</div>;
        }
    }
    return <MarkdownRenderer content={'```tsx\n' + activeTab.content + '\n```'} />;
}

return (
    <div className="h-full flex flex-col text-text-primary bg-surface">
        <header className="p-4 border-b border-border flex-shrink-0">
            <h1 className="text-xl font-bold flex items-center"><CpuChipIcon /><span
            className="ml-3">AI Feature Builder</span></h1>
        </header>
        <div className="flex-grow flex min-h-0">
            <main className="flex-1 flex flex-col min-w-0">
                <div className="flex-grow flex flex-col bg-background">
                    <div className="border-b border-border flex items-center bg-surface overflow-x-
                    auto">
                       {generatedFiles.map(file => (
                           <button key={file.filePath} onClick={() => setActiveTab(file)} className={`flex-
                           shrink-0 flex items-center gap-2 px-4 py-2 text-sm ${activeTab === file ? 'bg-
                           background border-b-2 border-primary text-text-primary' : 'text-text-secondary
                           hover:bg-gray-50'}`}><DocumentTextIcon /> {file.filePath}</button>
                       ))}
                       {unitTests && <button onClick={() => setActiveTab('TESTS')} className={`flex-
                       shrink-0 flex items-center gap-2 px-4 py-2 text-sm ${activeTab === 'TESTS' ?
                       'bg-background border-b-2 border-primary text-text-primary' : 'text-text-
                       secondary hover:bg-gray-50'}`}><BeakerIcon /> Tests</button>}
                       {commitMessage && <button onClick={() => setActiveTab('COMMIT')}
                       className={`flex-shrink-0 flex items-center gap-2 px-4 py-2 text-sm ${activeTab
                       === 'COMMIT' ? 'bg-background border-b-2 border-primary text-text-primary' :
                       'text-text-secondary hover:bg-gray-50'}`}><GitBranchIcon /> Commit</button>}
                       {dockerfile && !includeBackend && <button onClick={() =>
                       setActiveTab('DEPLOYMENT')} className={`flex-shrink-0 flex items-center gap-2
                       px-4 py-2 text-sm ${activeTab === 'DEPLOYMENT' ? 'bg-background border-b-2
                       border-primary text-text-primary' : 'text-text-secondary hover:bg-
                       gray-50'}`}><CloudIcon /> Dockerfile</button>}
                    </div>
                    <div className="flex-grow p-2 overflow-auto">
                        {isLoading && !generatedFiles.length ? <div className="flex justify-center
                        items-center h-full"><LoadingSpinner/></div> : renderContent()}
                    </div>
                </div>

                <div className="flex-shrink-0 p-4 border-t border-border bg-surface">
                     <div className="flex items-center gap-2 mb-2">
                       <label className="flex items-center gap-2 text-sm"><input type="checkbox"
                       checked={includeBackend} onChange={e => setIncludeBackend(e.target.checked)} />
                       Include Backend (Cloud Function + Firestore)</label>
                    </div>
                    <textarea value={prompt} onChange={(e) => setPrompt(e.target.value)}
```

```
                        placeholder="e.g., A user profile card with an avatar, name, and bio."
                        className="w-full p-2 bg-background border border-border rounded-md resize-none
                        text-sm h-20"/>
                         <div className="flex gap-2 mt-2">
                             <button onClick={handleGenerate} disabled={isLoading || isDeploying}
                             className="btn-primary flex-grow flex items-center justify-center gap-2 px-4
                             py-2">
                                {isLoading ? <><LoadingSpinner /> Generating...</> : 'Generate Feature'}
                            </button>
                            {includeBackend && generatedFiles.length > 0 && (
                                <button onClick={handleDeploy} disabled={isLoading || isDeploying}
                                className="bg-blue-600 text-white font-bold rounded-md hover:opacity-90
                                transition-all disabled:opacity-50 shadow-md flex-grow flex items-center
                                justify-center gap-2 px-4 py-2">
                                    {isDeploying ? <><LoadingSpinner /> Deploying...</> : <><PaperAirplaneIcon/>
                                    Deploy to GCP</>}
                                </button>
                            )}
                         </div>
                         {error && <p className="text-red-600 text-xs mt-2 text-center">{error}</p>}
                    </div>
                </main>
            </div>
        </div>
    );
};

// ===== AiCommandCenter_25.tsx =====

import React, { useState, useCallback } from 'react';
import { Type, FunctionDeclaration } from "@google/genai";
import { logError } from '../../services/telemetryService.ts';
import { getInferenceFunction, CommandResponse } from
'../../services/aiService.ts';
import { FEATURE_TAXONOMY } from '../../services/taxonomyService.ts';
import { useGlobalState } from '../../contexts/GlobalStateContext.tsx';
import { CommandLineIcon } from '../icons.tsx';
import { LoadingSpinner } from '../shared/index.tsx';
import { ALL_FEATURES } from './index.ts';

const functionDeclarations: FunctionDeclaration[] = [
    {
        name: 'navigateTo',
        description: 'Navigates to a specific feature page.',
        parameters: {
            type: Type.OBJECT,
            properties: {
                featureId: {
                    type: Type.STRING,
                    description: 'The ID of the feature to navigate to.',
                    enum: ALL_FEATURES.map(f => f.id)
                },
            },
            required: ['featureId'],
        },
    },
    {
        name: 'runFeatureWithInput',
        description: 'Navigates to a feature and passes initial data to it.',
        parameters: {
            type: Type.OBJECT,
            properties: {
```

```
                 featureId: {
                    type: Type.STRING,
                    description: 'The ID of the feature to run.',
                    enum: ALL_FEATURES.map(f => f.id)
                },
                props: {
                    type: Type.OBJECT,
                    description: 'An object containing the initial properties for the feature, based
                    on its required inputs.',
                    properties: {
                        initialCode: { type: Type.STRING },
                        initialPrompt: { type: Type.STRING },
                        beforeCode: { type: Type.STRING },
                        afterCode: { type: Type.STRING },
                        logInput: { type: Type.STRING },
                        diff: { type: Type.STRING },
                        codeInput: { type: Type.STRING },
                        jsonInput: { type: Type.STRING },
                    }
                }
            },
            required: ['featureId', 'props']
        }
    }
];

const knowledgeBase = FEATURE_TAXONOMY.map(f => `- ${f.name} (${f.id}):
${f.description} Inputs: ${f.inputs}`).join('\n');

const ExamplePromptButton: React.FC<{ text: string, onClick: (text: string) =>
void }> = ({ text, onClick }) => (
    <button
        onClick={() => onClick(text)}
        className="px-3 py-1.5 bg-surface border border-border rounded-full text-xs
        hover:bg-gray-100 transition-colors"
    >
        {text}
    </button>
)

export const AiCommandCenter: React.FC = () => {
    const { dispatch } = useGlobalState();
    const [prompt, setPrompt] = useState('');
    const [isLoading, setIsLoading] = useState(false);
    const [lastResponse, setLastResponse] = useState('');

    const handleCommand = useCallback(async () => {
        if (!prompt.trim()) return;

        setIsLoading(true);
        setLastResponse('');

        try {
            const response: CommandResponse = await getInferenceFunction(prompt,
            functionDeclarations, knowledgeBase);

            if (response.functionCalls && response.functionCalls.length > 0) {
                const call = response.functionCalls[0];
                const { name, args } = call;

                setLastResponse(`Understood! Executing command: ${name}`);
                switch (name) {
```

```
                case 'navigateTo':
                    dispatch({ type: 'SET_VIEW', payload: { view: args.featureId }});
                    break;
                case 'runFeatureWithInput':
                     dispatch({ type: 'SET_VIEW', payload: { view: args.featureId, props: args.props
                     } });
                    break;
                default:
                    setLastResponse(`Unknown command: ${name}`);
            }
             setPrompt('');
        } else {
             setLastResponse(response.text);
        }

    } catch (err) {
        logError(err as Error, { prompt });
        setLastResponse(err instanceof Error ? err.message : 'An unknown error
        occurred.');
    } finally {
        setIsLoading(false);
    }
}, [prompt, dispatch]);

const handleKeyDown = (e: React.KeyboardEvent) => {
    if (e.key === 'Enter' && !e.shiftKey) {
        e.preventDefault();
        handleCommand();
    }
};

const handleExampleClick = (text: string) => {
    setPrompt(text);
}

return (
    <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
        <header className="mb-6 text-center">
            <h1 className="text-4xl font-extrabold tracking-tight flex items-center justify-
            center">
                <CommandLineIcon />
                <span className="ml-3">AI Command Center</span>
            </h1>
            <p className="mt-2 text-lg text-text-secondary">What would you like to do?</p>
        </header>

        <div className="flex-grow flex flex-col justify-end max-w-3xl w-full mx-auto">
            {lastResponse && (
                <div className="mb-4 p-4 bg-surface rounded-lg text-text-primary border border-
                border">
                    <p><strong>AI:</strong> {lastResponse}</p>
                </div>
            )}
             <div className="relative">
                <textarea
                    value={prompt}
                    onChange={e => setPrompt(e.target.value)}
                    onKeyDown={handleKeyDown}
                    disabled={isLoading}
                    placeholder='Try "explain this code: const a = 1;" or "open the theme designer"'
                    className="w-full p-4 pr-28 rounded-lg bg-surface border border-border
                    focus:ring-2 focus:ring-primary focus:outline-none resize-none shadow-sm"
```

```
                        rows={2}
                    />
                    <button
                        onClick={handleCommand}
                        disabled={isLoading}
                        className="btn-primary absolute right-3 top-1/2 -translate-y-1/2 px-4 py-2"
                    >
                       {isLoading ? <LoadingSpinner/> : 'Send'}
                    </button>
                </div>
                 <div className="flex flex-wrap items-center justify-center gap-2 mt-4">
                    <ExamplePromptButton text="Open Theme Designer" onClick={handleExampleClick} />
                    <ExamplePromptButton text="Generate a commit for a bug fix"
                    onClick={handleExampleClick} />
                    <ExamplePromptButton text="Create a regex for email validation"
                    onClick={handleExampleClick} />
                </div>
                 <p className="text-xs text-text-secondary text-center mt-2">Press Enter to send,
                 Shift+Enter for new line.</p>
            </div>
        </div>
    );
};

// ===== AiImageGenerator_25.tsx =====

import React, { useState, useCallback, useRef } from 'react';
import { generateImage, generateImageFromImageAndText } from
'../../services/aiService.ts';
import { fileToBase64, blobToDataURL } from '../../services/fileUtils.ts';
import { ImageGeneratorIcon, SparklesIcon, ArrowDownTrayIcon, XMarkIcon } from
'../icons.tsx';
import { LoadingSpinner } from '../shared/index.tsx';

const surprisePrompts = [
    'A majestic lion wearing a crown, painted in the style of Van Gogh.',
    'A futuristic cityscape on another planet with two moons in the sky.',
    'A cozy, magical library inside a giant tree.',
    'A surreal image of a ship sailing on a sea of clouds.',
    'An astronaut riding a space-themed bicycle on the moon.',
];

interface UploadedImage {
    base64: string;
    dataUrl: string;
    mimeType: string;
}

export const AiImageGenerator: React.FC = () => {
    const [prompt, setPrompt] = useState<string>('A photorealistic image of a
    futuristic city at sunset, with flying cars.');
    const [uploadedImage, setUploadedImage] = useState<UploadedImage | null>(null);
    const [generatedImageUrl, setGeneratedImageUrl] = useState<string | null>(null);
    const [isLoading, setIsLoading] = useState<boolean>(false);
    const [error, setError] = useState<string>('');
    const fileInputRef = useRef<HTMLInputElement>(null);

    const handleGenerate = useCallback(async () => {
        if (!prompt.trim()) {
            setError('Please enter a prompt to generate an image.');
            return;
        }
```

```
    setIsLoading(true);
    setError('');
    setGeneratedImageUrl(null);
    try {
        let resultUrl: string;
        if (uploadedImage) {
            resultUrl = await generateImageFromImageAndText(prompt, uploadedImage.base64,
            uploadedImage.mimeType);
        } else {
            resultUrl = await generateImage(prompt);
        }
        setGeneratedImageUrl(resultUrl);
    } catch (err) {
        const errorMessage = err instanceof Error ? err.message : 'An unknown error
        occurred.';
        setError(`Failed to generate image: ${errorMessage}`);
    } finally {
        setIsLoading(false);
    }
}, [prompt, uploadedImage]);

const handleSurpriseMe = () => {
    const randomPrompt = surprisePrompts[Math.floor(Math.random() *
    surprisePrompts.length)];
    setPrompt(randomPrompt);
};

const processImageBlob = async (blob: Blob) => {
    try {
        const [dataUrl, base64] = await Promise.all([
            blobToDataURL(blob),
            fileToBase64(blob as File)
        ]);
        setUploadedImage({ dataUrl, base64, mimeType: blob.type });
    } catch (e) {
        setError('Could not process the image.');
    }
};

const handlePaste = useCallback(async (event: React.ClipboardEvent) => {
    const items = event.clipboardData.items;
    for (const item of items) {
        if (item.type.indexOf('image') !== -1) {
            const blob = item.getAsFile();
            if (blob) {
                await processImageBlob(blob);
                return;
            }
        }
    }
}, []);

const handleFileChange = async (event: React.ChangeEvent<HTMLInputElement>) => {
    const file = event.target.files?.[0];
    if (file) {
        await processImageBlob(file);
    }
};

const handleDownload = () => {
    if (!generatedImageUrl) return;
    const link = document.createElement('a');
```

```
    link.href = generatedImageUrl;
    link.download = `${prompt.slice(0, 30).replace(/\s/g, '_')}.png`;
    document.body.appendChild(link);
    link.click();
    document.body.removeChild(link);
}

return (
    <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
        <header className="mb-6">
            <h1 className="text-3xl font-bold flex items-center">
                <ImageGeneratorIcon />
                <span className="ml-3">AI Image Generator</span>
            </h1>
            <p className="text-text-secondary mt-1">Generate images from text, or provide an
            image for inspiration.</p>
        </header>

        <div className="flex-grow grid grid-cols-1 lg:grid-cols-2 gap-6 min-h-0">
            {/* Left Column: Inputs */}
            <div className="flex flex-col gap-4">
                <div>
                    <label htmlFor="prompt-input" className="text-sm font-medium text-text-
                    secondary">Your Prompt</label>
                    <textarea
                        id="prompt-input"
                        value={prompt}
                        onChange={(e) => setPrompt(e.target.value)}
                        placeholder="e.g., A cute cat wearing a wizard hat"
                        className="w-full p-3 mt-1 rounded-md bg-surface border border-border
                        focus:ring-2 focus:ring-primary focus:outline-none resize-y"
                        rows={3}
                    />
                </div>

                <div className="flex flex-col flex-grow min-h-[200px]">
                     <label className="text-sm font-medium text-text-secondary mb-1">Inspiration
                     Image (Optional)</label>
                     <div onPaste={handlePaste} className="relative flex-grow flex flex-col items-
                     center justify-center bg-surface p-4 rounded-lg border-2 border-dashed border-
                     border focus:outline-none focus:border-primary" tabIndex={0}>
                        {uploadedImage ? (
                            <>
                                <img src={uploadedImage.dataUrl} alt="Uploaded content" className="max-w-full
                                max-h-full object-contain rounded-md shadow-lg" />
                                <button onClick={() => setUploadedImage(null)} className="absolute top-2 right-2
                                p-1 bg-black/30 text-white rounded-full hover:bg-black/50"><XMarkIcon
                                /></button>
                            </>
                        ) : (
                            <div className="text-center text-text-secondary">
                                <h2 className="text-lg font-bold text-text-primary">Paste an image here</h2>
                                <p className="text-sm">(Cmd/Ctrl + V)</p>
                                <p className="text-xs my-1">or</p>
                                <button onClick={() => fileInputRef.current?.click()} className="text-sm font-
                                semibold text-primary hover:underline">Upload File</button>
                                <input type="file" ref={fileInputRef} onChange={handleFileChange}
                                accept="image/*" className="hidden"/>
                            </div>
                        )}
                     </div>
                </div>
```

```
                    <div className="flex gap-2">
                        <button
                            onClick={handleGenerate}
                            disabled={isLoading}
                            className="btn-primary w-full flex items-center justify-center px-6 py-3"
                        >
                            {isLoading ? <LoadingSpinner /> : 'Generate Image'}
                        </button>
                        <button
                            onClick={handleSurpriseMe}
                            disabled={isLoading}
                            className="px-4 py-3 bg-surface border border-border rounded-md hover:bg-
                            gray-100 transition-colors"
                            title="Surprise Me!"
                        >
                            <SparklesIcon />
                        </button>
                    </div>
                </div>

                {/* Right Column: Output */}
                <div className="flex flex-col h-full">
                    <label className="text-sm font-medium text-text-secondary mb-2">Generated
                    Image</label>
                    <div className="flex-grow flex items-center justify-center bg-background
                    border-2 border-dashed border-border rounded-lg p-4 relative overflow-auto">
                        {isLoading && <LoadingSpinner />}
                        {error && <p className="text-red-500 text-center">{error}</p>}
                        {generatedImageUrl && !isLoading && (
                            <>
                                <img src={generatedImageUrl} alt={prompt || "Generated by AI"} className="max-w-
                                full max-h-full object-contain rounded-md shadow-lg" />
                                <button
                                  onClick={handleDownload}
                                  className="absolute top-4 right-4 p-2 bg-black/30 text-white rounded-full
                                  hover:bg-black/50 backdrop-blur-sm"
                                  title="Download Image"
                                >
                                    <ArrowDownTrayIcon />
                                </button>
                            </>
                        )}
                        {!isLoading && !generatedImageUrl && !error && (
                            <div className="text-center text-text-secondary">
                                <p>Your generated image will appear here.</p>
                            </div>
                        )}
                    </div>
                </div>
            </div>
        </div>
    );
};

// ===== XbrlConverter_25.tsx =====

import React, { useState, useCallback } from 'react';
import { convertJsonToXbrlStream } from '../../services/aiService.ts';
import { XbrlConverterIcon } from '../icons.tsx';
import { LoadingSpinner, MarkdownRenderer } from '../shared/index.tsx';

const exampleJson = `{
```

```
  "company": "ExampleCorp",
  "year": 2024,
  "quarter": 2,
  "revenue": {
    "amount": 1500000,
    "currency": "USD"
  },
  "profit": {
    "amount": 250000,
    "currency": "USD"
  }
}`;

export const XbrlConverter: React.FC<{ jsonInput?: string }> = ({ jsonInput:
initialJsonInput }) => {
    const [jsonInput, setJsonInput] = useState<string>(initialJsonInput ||
    exampleJson);
    const [xbrlOutput, setXbrlOutput] = useState<string>('');
    const [isLoading, setIsLoading] = useState<boolean>(false);
    const [error, setError] = useState<string>('');

    const handleConvert = useCallback(async (jsonToConvert: string) => {
        if (!jsonToConvert.trim()) {
            setError('Please enter valid JSON to convert.');
            return;
        }
        setIsLoading(true);
        setError('');
        setXbrlOutput('');
        try {
            const stream = convertJsonToXbrlStream(jsonToConvert);
            let fullResponse = '';
            for await (const chunk of stream) {
                fullResponse += chunk;
                setXbrlOutput(fullResponse);
            }
        } catch (err) {
            const errorMessage = err instanceof Error ? err.message : 'An unknown error
            occurred.';
            setError(`Failed to convert: ${errorMessage}`);
        } finally {
            setIsLoading(false);
        }
    }, []);

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
            <header className="mb-6">
                <h1 className="text-3xl font-bold flex items-center">
                    <XbrlConverterIcon />
                    <span className="ml-3">JSON to XBRL Converter</span>
                </h1>
                <p className="text-text-secondary mt-1">Convert JSON data into a simplified
                XBRL-like XML format using AI.</p>
            </header>
            <div className="flex-grow flex flex-col gap-4 min-h-0">
                <div className="flex flex-col flex-1 min-h-0">
                    <label htmlFor="json-input" className="text-sm font-medium text-text-secondary
                    mb-2">JSON Input</label>
                    <textarea
                        id="json-input"
                        value={jsonInput}
```

```
                        onChange={(e) => setJsonInput(e.target.value)}
                        placeholder="Paste your JSON here..."
                        className="flex-grow p-4 bg-surface border border-border rounded-md resize-none
                        font-mono text-sm"
                    />
                </div>
                 <div className="flex-shrink-0">
                    <button
                        onClick={() => handleConvert(jsonInput)}
                        disabled={isLoading}
                        className="btn-primary w-full max-w-xs mx-auto flex items-center justify-center
                        px-6 py-3"
                    >
                        {isLoading ? <LoadingSpinner /> : 'Convert to XBRL'}
                    </button>
                </div>
                <div className="flex flex-col flex-1 min-h-0">
                    <label className="text-sm font-medium text-text-secondary mb-2">XBRL-like XML
                    Output</label>
                    <div className="relative flex-grow p-1 bg-background border border-border
                    rounded-md overflow-y-auto">
                        {isLoading && !xbrlOutput && <div className="flex items-center justify-center
                        h-full"><LoadingSpinner /></div>}
                        {error && <p className="p-4 text-red-500">{error}</p>}
                        {xbrlOutput && <MarkdownRenderer content={'```xml\n' +
                        xbrlOutput.replace(/```xml\n|```/g, '') + '\n```'} />}
                        {!isLoading && xbrlOutput && <button onClick={() =>
                        navigator.clipboard.writeText(xbrlOutput)} className="absolute top-2 right-2
                        px-2 py-1 bg-gray-100 hover:bg-gray-200 rounded-md text-xs">Copy XML</button>}
                        {!isLoading && !xbrlOutput && !error && <div className="text-text-secondary
                        h-full flex items-center justify-center">Output will appear here.</div>}
                    </div>
                </div>
            </div>
        </div>
    );
};

// ===== CodeReviewBot_27.tsx =====

import React, { useState, useCallback } from 'react';
import { reviewCodeStream } from '../../services/index.ts';
import { useAiPersonalities } from '../../hooks/useAiPersonalities.ts';
import { formatSystemPromptToString } from '../../utils/promptUtils.ts';
import { CpuChipIcon } from '../icons.tsx';
import { LoadingSpinner } from '../shared/index.tsx';
import { MarkdownRenderer } from '../shared/index.tsx';

const exampleCode = `function UserList(users) {
  if (users.length = 0) {
    return "no users";
  } else {
    return (
      users.map(u => {
        return <li>{u.name}</li>
      })
    )
  }
}`;

export const CodeReviewBot: React.FC = () => {
    const [code, setCode] = useState<string>(exampleCode);
```

```
const [review, setReview] = useState<string>('');
const [isLoading, setIsLoading] = useState<boolean>(false);
const [error, setError] = useState<string>('');
const [personalities] = useAiPersonalities();
const [selectedPersonalityId, setSelectedPersonalityId] =
useState<string>('default');

const handleGenerate = useCallback(async () => {
    if (!code.trim()) {
        setError('Please enter some code to review.');
        return;
    }
    setIsLoading(true);
    setError('');
    setReview('');

    let systemInstruction: string | undefined = undefined;
    if (selectedPersonalityId !== 'default') {
        const personality = personalities.find(p => p.id === selectedPersonalityId);
        if (personality) {
            systemInstruction = formatSystemPromptToString(personality);
        }
    }

    try {
        const stream = reviewCodeStream(code, systemInstruction);
        let fullResponse = '';
        for await (const chunk of stream) {
            fullResponse += chunk;
            setReview(fullResponse);
        }
    } catch (err) {
        const errorMessage = err instanceof Error ? err.message : 'An unknown error
        occurred.';
        setError(`Failed to get review: ${errorMessage}`);
    } finally {
        setIsLoading(false);
    }
}, [code, selectedPersonalityId, personalities]);

return (
    <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
        <header className="mb-6">
            <h1 className="text-3xl font-bold flex items-center">
                <CpuChipIcon />
                <span className="ml-3">AI Code Review Bot</span>
            </h1>
            <p className="text-text-secondary mt-1">Get an automated code review from
            Gemini.</p>
        </header>
        <div className="flex-grow flex flex-col gap-4 min-h-0">
            <div className="flex flex-col flex-1 min-h-0">
                <label htmlFor="code-input" className="text-sm font-medium text-text-secondary
                mb-2">Code to Review</label>
                <textarea
                    id="code-input"
                    value={code}
                    onChange={(e) => setCode(e.target.value)}
                    placeholder="Paste your code here..."
                    className="flex-grow p-4 bg-surface border border-border rounded-md resize-none
                    font-mono text-sm"
                />
```

```
                </div>
                 <div className="flex-shrink-0 flex items-center justify-center gap-4">
                     <div className="w-full max-w-xs">
                        <label htmlFor="personality-select" className="text-sm font-medium text-text-
                        secondary">Reviewer Personality</label>
                        <select
                            id="personality-select"
                            value={selectedPersonalityId}
                            onChange={e => setSelectedPersonalityId(e.target.value)}
                            className="w-full mt-1 p-2 bg-surface border border-border rounded-md text-sm"
                        >
                            <option value="default">Default</option>
                            {personalities.map(p => (
                                <option key={p.id} value={p.id}>{p.name}</option>
                            ))}
                        </select>
                     </div>
                    <button
                        onClick={handleGenerate}
                        disabled={isLoading}
                        className="btn-primary self-end h-[42px] w-full max-w-xs flex items-center
                        justify-center px-6 py-3"
                    >
                        {isLoading ? <LoadingSpinner /> : 'Request Review'}
                    </button>
                </div>
                <div className="flex flex-col flex-1 min-h-0">
                    <label className="text-sm font-medium text-text-secondary mb-2">AI
                    Feedback</label>
                    <div className="flex-grow p-4 bg-background border border-border rounded-md
                    overflow-y-auto">
                        {isLoading && !review && <div className="flex items-center justify-center
                        h-full"><LoadingSpinner /></div>}
                        {error && <p className="text-red-500">{error}</p>}
                        {review && <MarkdownRenderer content={review} />}
                         {!isLoading && !review && !error && <div className="text-text-secondary h-full
                         flex items-center justify-center">Review will appear here.</div>}
                    </div>
                </div>
            </div>
        </div>
    );
};

// ===== ChangelogGenerator_27.tsx =====

import React, { useState, useCallback } from 'react';
import { generateChangelogFromLogStream } from '../../services/aiService.ts';
import { GitBranchIcon } from '../icons.tsx';
import { LoadingSpinner } from '../shared/index.tsx';
import { MarkdownRenderer } from '../shared/index.tsx';

const exampleLog = `commit 3a4b5c...
Author: Dev One <dev.one@example.com>
Date:   Mon Jul 15 11:30:00 2024 -0400

    feat: add user login page

commit 1a2b3c...
Author: Dev Two <dev.two@example.com>
Date:   Mon Jul 15 10:00:00 2024 -0400
    fix: correct typo in header
```

```
`;

export const ChangelogGenerator: React.FC = () => {
    const [log, setLog] = useState(exampleLog);
    const [changelog, setChangelog] = useState('');
    const [isLoading, setIsLoading] = useState(false);
    const [error, setError] = useState('');

    const handleGenerate = useCallback(async () => {
        if (!log.trim()) {
            setError('Please paste your git log output.');
            return;
        }
        setIsLoading(true);
        setError('');
        setChangelog('');
        try {
            const stream = generateChangelogFromLogStream(log);
            let fullResponse = '';
            for await (const chunk of stream) {
                fullResponse += chunk;
                setChangelog(fullResponse);
            }
        } catch (err) {
            setError(err instanceof Error ? err.message : 'An unknown error occurred.');
        } finally {
            setIsLoading(false);
        }
    }, [log]);

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
            <header className="mb-6">
                <h1 className="text-3xl font-bold flex items-center">
                    <GitBranchIcon />
                    <span className="ml-3">AI Changelog Generator</span>
                </h1>
                <p className="text-text-secondary mt-1">Generate a markdown changelog from your
                raw git log.</p>
            </header>
            <div className="flex-grow flex flex-col gap-4 min-h-0">
                <div className="flex flex-col flex-1 min-h-0">
                    <label htmlFor="commit-input" className="text-sm font-medium text-text-secondary
                    mb-2">Raw Git Log</label>
                    <textarea
                        id="commit-input"
                        value={log}
                        onChange={(e) => setLog(e.target.value)}
                        className="flex-grow p-4 bg-surface border border-border rounded-md resize-none
                        font-mono text-sm"
                    />
                </div>
                <div className="flex-shrink-0">
                    <button onClick={handleGenerate} disabled={isLoading} className="btn-primary
                    w-full max-w-xs mx-auto flex items-center justify-center px-6 py-3">
                        {isLoading ? <LoadingSpinner /> : 'Generate Changelog'}
                    </button>
                </div>
                <div className="flex flex-col flex-1 min-h-0">
                    <label className="text-sm font-medium text-text-secondary mb-2">Generated
                    Changelog.md</label>
                    <div className="relative flex-grow p-4 bg-background border border-border
```

```
                    rounded-md overflow-y-auto">
                        {isLoading && !changelog && <div className="flex items-center justify-center
                        h-full"><LoadingSpinner /></div>}
                        {error && <p className="text-red-500">{error}</p>}
                        {changelog && <MarkdownRenderer content={changelog} />}
                        {!isLoading && changelog && <button onClick={() =>
                        navigator.clipboard.writeText(changelog)} className="absolute top-2 right-2 px-2
                        py-1 bg-gray-100 text-xs rounded-md hover:bg-gray-200">Copy</button>}
                    </div>
                </div>
            </div>
        </div>
    );
};

// ===== ApiMockGenerator_15.tsx =====

import React, { useState, useEffect } from 'react';
import { generateMockData } from '../../services/aiService.ts';
import { startMockServer, stopMockServer, setMockRoutes, isMockServerRunning }
from '../../services/mocking/mockServer.ts';
import { saveMockCollection, getAllMockCollections, deleteMockCollection } from
'../../services/mocking/db.ts';
import { ServerStackIcon, SparklesIcon, PlusIcon, TrashIcon } from
'../icons.tsx';
import { LoadingSpinner } from '../shared/index.tsx';

const exampleSchema = "a user with an id, name, email, and a nested address
object containing a city and country";

export const ApiMockGenerator: React.FC = () => {
    const [schema, setSchema] = useState(exampleSchema);
    const [count, setCount] = useState(5);
    const [collectionName, setCollectionName] = useState('users');
    const [collections, setCollections] = useState<any[]>([]);
    const [generatedData, setGeneratedData] = useState<any[] | null>(null);
    const [isLoading, setIsLoading] = useState(false);
    const [error, setError] = useState('');
    const [isServerRunning, setIsServerRunning] = useState(isMockServerRunning());
    const [routes, setRoutes] = useState([{ path: '/api/users', method: 'GET' }]);

    useEffect(() => {
        const loadCollections = async () => {
            const storedCollections = await getAllMockCollections();
            setCollections(storedCollections);
        };
        loadCollections();
    }, []);

    const handleGenerate = async () => {
        if (!schema.trim() || !collectionName.trim()) {
            setError('Schema description and collection name are required.');
            return;
        }
        setIsLoading(true);
        setError('');
        try {
            const data = await generateMockData(schema, count);
            setGeneratedData(data);
            const collectionId = collectionName.toLowerCase().replace(/\s/g, '-');
            await saveMockCollection({ id: collectionId, schemaDescription: schema, data });
            setCollections(await getAllMockCollections());
```

```
    } catch (err) {
        setError(err instanceof Error ? err.message : 'Failed to generate data.');
    } finally {
        setIsLoading(false);
    }
};

const handleServerToggle = async () => {
    if (isServerRunning) {
        await stopMockServer();
        setIsServerRunning(false);
    } else {
        try {
            await startMockServer();
            setIsServerRunning(true);
            updateRoutes();
        } catch (err) {
            setError(err instanceof Error ? err.message : 'Could not start server.');
        }
    }
};

const updateRoutes = () => {
    const mockRoutes = routes.map(route => {
        // A simple implementation: find first matching collection for path
        const matchingCollection = collections.find(c => route.path.includes(c.id));
        return {
            ...route,
            response: {
                status: 200,
                body: matchingCollection ? matchingCollection.data : { message: 'No data found
                for this route.' }
            }
        };
    });
    setMockRoutes(mockRoutes as any);
};

useEffect(() => {
    if (isServerRunning) {
        updateRoutes();
    }
}, [routes, collections, isServerRunning]);

return (
    <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
        <header className="mb-6 flex justify-between items-start">
            <div>
                <h1 className="text-3xl font-bold flex items-center"><ServerStackIcon /><span
                className="ml-3">AI API Mock Server</span></h1>
                <p className="text-text-secondary mt-1">Generate and serve mock API data locally
                using a service worker.</p>
            </div>
            <button onClick={handleServerToggle} className={`px-4 py-2 rounded-md font-
            semibold flex items-center gap-2 ${isServerRunning ? 'bg-green-100 text-
            green-700' : 'bg-gray-100'}`}>
                <span className={`w-3 h-3 rounded-full ${isServerRunning ? 'bg-green-500' : 'bg-
                gray-400'}`}></span>
                {isServerRunning ? 'Server Running' : 'Server Stopped'}
            </button>
        </header>
        <div className="flex-grow grid grid-cols-1 lg:grid-cols-3 gap-6 min-h-0">
```

```
                {/* Left: Generator */}
                <div className="lg:col-span-1 flex flex-col gap-4 bg-surface p-4 border border-
                border rounded-lg">
                    <h3 className="text-lg font-bold">1. Generate Data</h3>
                    <div><label className="text-sm">Describe the data schema</label><textarea
                    value={schema} onChange={e => setSchema(e.target.value)} className="w-full mt-1
                    p-2 bg-background border border-border rounded" rows={4}/></div>
                    <div className="flex gap-2">
                        <div className="flex-grow"><label className="text-sm">Collection
                        Name</label><input type="text" value={collectionName} onChange={e =>
                        setCollectionName(e.target.value)} className="w-full mt-1 p-2 bg-background
                        border border-border rounded"/></div>
                        <div><label className="text-sm">Count</label><input type="number" value={count}
                        onChange={e => setCount(Number(e.target.value))} className="w-20 mt-1 p-2 bg-
                        background border border-border rounded"/></div>
                    </div>
                    <button onClick={handleGenerate} disabled={isLoading} className="btn-primary
                    py-2 flex items-center justify-center gap-2">{isLoading ? <LoadingSpinner/> :
                    <><SparklesIcon/> Generate & Save</>}</button>
                    {error && <p className="text-red-500 text-xs">{error}</p>}
                </div>

                {/* Middle: Data & Routes */}
                <div className="lg:col-span-2 flex flex-col gap-4 min-h-0">
                   <div className="bg-surface p-4 border border-border rounded-lg flex-grow flex
                   flex-col min-h-0">
                        <h3 className="text-lg font-bold mb-2">2. View Data & Configure Routes</h3>
                        <div className="flex-grow grid grid-cols-2 gap-4 min-h-0">
                            <div className="overflow-y-auto">
                                <h4 className="font-semibold text-sm mb-1">Saved Collections</h4>
                                {collections.map(c => <div key={c.id} className="text-xs p-2 bg-background
                                rounded border border-border mb-1">{c.id} ({c.data.length} items)</div>)}
                                <h4 className="font-semibold text-sm mb-1 mt-2">Last Generated Data</h4>
                                <pre className="text-xs p-2 bg-background rounded border border-border
                                whitespace-pre-wrap">{generatedData ? JSON.stringify(generatedData, null, 2) :
                                'No data generated yet.'}</pre>
                            </div>
                            <div className="overflow-y-auto">
                                <h4 className="font-semibold text-sm mb-1">Mock Routes</h4>
                                {routes.map((r, i) => <div key={i} className="flex gap-1 items-center
                                mb-1"><select value={r.method} className="p-1 text-xs bg-background border
                                rounded"><option>GET</option><option>POST</option></select><input type="text"
                                value={r.path} className="flex-grow p-1 text-xs bg-background border rounded"
                                /></div>)}
                                 <p className="text-xs text-text-secondary mt-2">Routes are automatically mapped
                                 to collections by name (e.g., `/api/users` maps to `users` collection).</p>
                            </div>
                        </div>
                   </div>
                </div>
            </div>
        </div>
    );
};

// ===== SecurityScanner_15.tsx =====

import React, { useState } from 'react';
import { analyzeCodeForVulnerabilities } from '../../services/aiService.ts';
import { runStaticScan, SecurityIssue } from
'../../services/security/staticAnalysisService.ts';
import type { SecurityVulnerability } from '../../types.ts';
```

```
import { ShieldCheckIcon, SparklesIcon } from '../icons.tsx';
import { LoadingSpinner, MarkdownRenderer } from '../shared/index.tsx';

const exampleCode = `function UserProfile({ user }) {
  // TODO: remove this temporary api key
  const API_KEY = "sk-xxxxxxxxxxxxxxxxxxxxxxxxxxxx";
  const userContent = user.bio; // This might contain malicious scripts

  return (
    <div>
      <h2>{user.name}</h2>
      <div dangerouslySetInnerHTML={{ __html: userContent }} />
    </div>
  );
}`;

export const SecurityScanner: React.FC = () => {
    const [code, setCode] = useState(exampleCode);
    const [localIssues, setLocalIssues] = useState<SecurityIssue[]>([]);
    const [aiIssues, setAiIssues] = useState<SecurityVulnerability[]>([]);
    const [isLoading, setIsLoading] = useState(false);
    const [error, setError] = useState('');

    const handleScan = async () => {
        if (!code.trim()) {
            setError('Please enter code to scan.');
            return;
        }
        setIsLoading(true);
        setError('');
        setLocalIssues([]);
        setAiIssues([]);
        try {
            // Run local scan first
            const staticIssues = runStaticScan(code);
            setLocalIssues(staticIssues);

            // Then run AI scan
            const geminiIssues = await analyzeCodeForVulnerabilities(code);
            setAiIssues(geminiIssues);

        } catch (err) {
            setError(err instanceof Error ? err.message : 'An error occurred during
            scanning.');
        } finally {
            setIsLoading(false);
        }
    };

    const SeverityBadge: React.FC<{ severity: string }> = ({ severity }) => {
        const colors: Record<string, string> = {
            'Critical': 'bg-red-500 text-white',
            'High': 'bg-red-400 text-white',
            'Medium': 'bg-yellow-400 text-yellow-900',
            'Low': 'bg-blue-400 text-white',
            'Informational': 'bg-gray-400 text-gray-900',
        };
        return <span className={`px-2 py-0.5 text-xs font-bold rounded-full
        ${colors[severity] || 'bg-gray-300'}`}>{severity}</span>
    }

    return (
```

```
    <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
        <header className="mb-6">
            <h1 className="text-3xl font-bold flex items-center"><ShieldCheckIcon /><span
            className="ml-3">AI Security Co-Pilot</span></h1>
            <p className="text-text-secondary mt-1">Find vulnerabilities in your code with
            static analysis and AI.</p>
        </header>
        <div className="flex-grow grid grid-cols-1 lg:grid-cols-2 gap-6 min-h-0">
            <div className="flex flex-col">
                <label className="text-sm mb-2">Code to Scan</label>
                <textarea value={code} onChange={e => setCode(e.target.value)} className="w-full
                flex-grow p-2 bg-surface border rounded font-mono text-xs" />
                <button onClick={handleScan} disabled={isLoading} className="btn-primary w-full
                mt-4 py-2 flex justify-center items-center gap-2">{isLoading ? <LoadingSpinner/>
                : 'Scan Code'}</button>
            </div>
            <div className="flex flex-col bg-surface p-4 border rounded-lg">
                <h3 className="text-lg font-bold mb-2">Scan Results</h3>
                {error && <p className="text-red-500">{error}</p>}
                <div className="flex-grow overflow-y-auto pr-2 space-y-4">
                    {isLoading && <div className="flex justify-center items-center
                    h-full"><LoadingSpinner/></div>}
                    {!isLoading && localIssues.length === 0 && aiIssues.length === 0 && <p
                    className="text-text-secondary text-center mt-8">No issues found. Run a scan to
                    begin.</p>}

                    {localIssues.length > 0 && <div>
                        <h4 className="font-semibold text-sm mb-1">Static Analysis Findings</h4>
                        {localIssues.map((issue, i) => <div key={`local-${i}`} className="p-2 bg-
                        background border rounded mb-2"><p className="font-bold flex items-center
                        gap-2">{issue.type} <SeverityBadge severity={issue.severity} /></p><p
                        className="text-xs">Line {issue.line}: {issue.description}</p></div>)}
                    </div>}

                     {aiIssues.length > 0 && <div>
                        <h4 className="font-semibold text-sm mb-1 flex items-center
                        gap-1"><SparklesIcon/> AI-Powered Findings</h4>
                        {aiIssues.map((issue, i) => (
                            <details key={`ai-${i}`} className="p-2 bg-background border rounded mb-2">
                                <summary className="cursor-pointer font-bold flex items-center
                                gap-2">{issue.vulnerability} <SeverityBadge severity={issue.severity}
                                /></summary>
                                <div className="mt-2 pt-2 border-t text-xs space-y-2">
                                    <p><strong>Description:</strong> {issue.description}</p>
                                    <p><strong>Mitigation:</strong> {issue.mitigation}</p>
                                    {issue.exploitSuggestion && (
                                        <div>
                                            <strong>Exploit Simulation:</strong>
                                            <div className="mt-1 p-2 bg-gray-50 rounded">
                                                <MarkdownRenderer content={'```bash\n' + issue.exploitSuggestion + '\n```'}/>
                                            </div>
                                        </div>
                                    )}
                                </div>
                            </details>
                        ))}
                    </div>}
                </div>
            </div>
        </div>
    </div>
);
```

```
};

// ===== TerraformGenerator_12.tsx =====

import React, { useState, useCallback } from 'react';
import { generateTerraformConfig } from '../../services/geminiService.ts';
import { CpuChipIcon, SparklesIcon } from '../icons.tsx';
import { LoadingSpinner, MarkdownRenderer } from '../shared/index.tsx';

export const TerraformGenerator: React.FC = () => {
    const [description, setDescription] = useState('An S3 bucket for static website
    hosting');
    const [cloud, setCloud] = useState<'aws' | 'gcp'>('aws');
    const [config, setConfig] = useState('');
    const [isLoading, setIsLoading] = useState(false);
    const [error, setError] = useState('');

    const handleGenerate = useCallback(async () => {
        if (!description.trim()) {
            setError('Please provide a description.');
            return;
        }
        setIsLoading(true);
        setError('');
        setConfig('');
        try {
            // Context is stubbed for now but demonstrates future capability
            const context = 'User might have existing VPCs. Check before creating new
            ones.';
            const result = await generateTerraformConfig(cloud, description, context);
            setConfig(result);
        } catch (err) {
            setError(err instanceof Error ? err.message : 'Failed to generate config.');
        } finally {
            setIsLoading(false);
        }
    }, [description, cloud]);

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
            <header className="mb-6">
                <h1 className="text-3xl font-bold flex items-center"><CpuChipIcon /><span
                className="ml-3">AI Terraform Generator</span></h1>
                <p className="text-text-secondary mt-1">Generate infrastructure-as-code from a
                description, with context from your cloud provider.</p>
            </header>
            <div className="flex-grow flex flex-col gap-4 min-h-0">
                 <div className="flex flex-col flex-1 min-h-0">
                     <div className="grid grid-cols-1 md:grid-cols-3 gap-4 mb-4">
                        <div>
                            <label className="block text-sm">Cloud Provider</label>
                            <select value={cloud} onChange={e => setCloud(e.target.value as 'aws' | 'gcp')}
                            className="w-full mt-1 p-2 bg-surface border rounded">
                                <option value="aws">AWS</option>
                                <option value="gcp">GCP</option>
                            </select>
                        </div>
                        <div className="md:col-span-2">
                            <label className="block text-sm">Describe the infrastructure</label>
                            <input type="text" value={description} onChange={e =>
                            setDescription(e.target.value)} className="w-full mt-1 p-2 bg-surface border
                            rounded"/>
```

```
                        </div>
                    </div>
                     <button onClick={handleGenerate} disabled={isLoading} className="btn-primary
                     w-full max-w-xs mx-auto flex items-center justify-center py-2"><SparklesIcon />
                     {isLoading ? 'Generating...' : 'Generate Configuration'}</button>
                </div>
                 <div className="flex flex-col flex-grow min-h-0">
                    <label className="text-sm font-medium text-text-secondary mb-2">Generated
                    Terraform (.tf)</label>
                    <div className="relative flex-grow p-1 bg-background border border-border
                    rounded-md overflow-y-auto">
                        {isLoading && !config && <div className="flex items-center justify-center
                        h-full"><LoadingSpinner /></div>}
                        {error && <p className="p-4 text-red-500">{error}</p>}
                        {config && <MarkdownRenderer content={config} />}
                         {!isLoading && !config && !error && <div className="text-text-secondary h-full
                         flex items-center justify-center">Generated config will appear here.</div>}
                    </div>
                </div>
            </div>
        </div>
    );
};

// ===== AiPersonalityForge_12.tsx =====

import React, { useState, useEffect, useRef } from 'react';
import { SparklesIcon, PlusIcon, TrashIcon, ArrowDownTrayIcon,
ArrowUpOnSquareIcon } from '../icons.tsx';
import { useAiPersonalities } from '../../hooks/useAiPersonalities.ts';
import { formatSystemPromptToString } from '../../utils/promptUtils.ts';
import { streamContent } from '../../services/geminiService.ts';
import { downloadJson } from '../../services/fileUtils.ts';
import type { SystemPrompt } from '../../types.ts';
import { LoadingSpinner, MarkdownRenderer } from '../shared/index.tsx';
import { useNotification } from '../../contexts/NotificationContext.tsx';

const defaultNewPrompt: Omit<SystemPrompt, 'id' | 'name'> = {
    persona: 'You are a helpful assistant.',
    rules: [],
    outputFormat: 'markdown',
    exampleIO: [],
};

export const AiPersonalityForge: React.FC = () => {
    const [personalities, setPersonalities] = useAiPersonalities();
    const [activeId, setActiveId] = useState<string | null>(null);
    const { addNotification } = useNotification();
    const fileInputRef = useRef<HTMLInputElement>(null);

    // Testbed State
    const [testbedInput, setTestbedInput] = useState('');
    const [chatHistory, setChatHistory] = useState<{ role: 'user' | 'model';
    content: string }[]>([]);
    const [isStreaming, setIsStreaming] = useState(false);

    const activePersonality = personalities.find(p => p.id === activeId);

    useEffect(() => {
        if (!activeId && personalities.length > 0) {
            setActiveId(personalities[0].id);
        }
```

```
}, [personalities, activeId]);

const handleUpdate = (field: keyof SystemPrompt, value: any) => {
    if (!activePersonality) return;
    const updated = { ...activePersonality, [field]: value };
    setPersonalities(personalities.map(p => (p.id === activeId ? updated : p)));
};

const handleAddNew = () => {
    const newId = Date.now().toString();
    const newPersonality: SystemPrompt = { ...defaultNewPrompt, id: newId, name:
    'Untitled Personality' };
    setPersonalities([...personalities, newPersonality]);
    setActiveId(newId);
};

const handleDelete = (id: string) => {
    if (window.confirm('Are you sure you want to delete this personality?')) {
        setPersonalities(personalities.filter(p => p.id !== id));
        if (activeId === id) {
            setActiveId(personalities.length > 1 ? personalities[0].id : null);
        }
    }
};

const handleTestbedSend = async () => {
    if (!testbedInput.trim() || !activePersonality || isStreaming) return;

    const systemInstruction = formatSystemPromptToString(activePersonality);
    const newHistory = [...chatHistory, { role: 'user' as const, content:
    testbedInput }];
    setChatHistory(newHistory);
    setTestbedInput('');
    setIsStreaming(true);

    try {
        const stream = streamContent(testbedInput, systemInstruction, 0.7);
        let fullResponse = '';
        setChatHistory(prev => [...prev, { role: 'model', content: '' }]);
        for await (const chunk of stream) {
            fullResponse += chunk;
            setChatHistory(prev => {
                const last = prev[prev.length - 1];
                if (last.role === 'model') {
                    return [...prev.slice(0, -1), { role: 'model', content: fullResponse }];
                }
                return prev;
            });
        }
    } catch (e) {
        const errorMsg = e instanceof Error ? e.message : 'An error occurred';
        setChatHistory(prev => [...prev, { role: 'model', content: `**Error:**
        ${errorMsg}` }]);
    } finally {
        setIsStreaming(false);
    }
};

const handleExport = () => {
    if (!activePersonality) return;
    downloadJson(activePersonality, `${activePersonality.name.replace(/\s+/g,
    '_')}.json`);
```

```
    addNotification('Personality exported!', 'success');
};

const handleImport = (e: React.ChangeEvent<HTMLInputElement>) => {
    const file = e.target.files?.[0];
    if (!file) return;
    const reader = new FileReader();
    reader.onload = (event) => {
        try {
            const imported = JSON.parse(event.target?.result as string) as SystemPrompt;
            // Basic validation
            if (imported.id && imported.name && imported.persona) {
                setPersonalities(prev => [...prev.filter(p => p.id !== imported.id), imported]);
                setActiveId(imported.id);
                addNotification('Personality imported!', 'success');
            } else {
                 addNotification('Invalid personality file.', 'error');
            }
        } catch {
             addNotification('Failed to parse JSON file.', 'error');
        }
    };
    reader.readAsText(file);
};

return (
    <div className="h-full flex text-text-primary">
        {/* Sidebar */}
        <aside className="w-64 bg-surface border-r border-border flex flex-col">
            <div className="p-4 border-b border-border">
                <h2 className="text-lg font-bold">Personalities</h2>
            </div>
            <div className="flex-grow overflow-y-auto">
                {personalities.map(p => (
                    <div key={p.id} onClick={() => setActiveId(p.id)} className={`group flex
                    justify-between items-center p-3 text-sm cursor-pointer ${activeId === p.id ?
                    'bg-primary/10 text-primary' : 'hover:bg-gray-100 dark:hover:bg-slate-700'}`}>
                        <span className="truncate">{p.name}</span>
                        <button onClick={(e) => { e.stopPropagation(); handleDelete(p.id)}}
                        className="opacity-0 group-hover:opacity-100 text-text-secondary hover:text-
                        red-500"><TrashIcon /></button>
                    </div>
                ))}
            </div>
            <div className="p-4 border-t border-border space-y-2">
                <button onClick={handleAddNew} className="btn-primary w-full py-2 text-sm flex
                items-center justify-center gap-2"><PlusIcon /> New</button>
                <div className="flex gap-2">
                     <button onClick={() => fileInputRef.current?.click()} className="flex-1 py-2
                     text-sm bg-gray-100 dark:bg-slate-700 rounded-md flex items-center justify-
                     center gap-2"><ArrowUpOnSquareIcon/> Import</button>
                     <button onClick={handleExport} className="flex-1 py-2 text-sm bg-gray-100
                     dark:bg-slate-700 rounded-md flex items-center justify-center
                     gap-2"><ArrowDownTrayIcon/> Export</button>
                     <input type="file" ref={fileInputRef} onChange={handleImport} accept=".json"
                     className="hidden"/>
                </div>
            </div>
        </aside>
        {/* Main Content */}
        {activePersonality ? (
             <div className="flex-1 grid grid-cols-2 gap-px bg-border">
```

```
        {/* Editor */}
        <div className="bg-background p-4 flex flex-col gap-4 overflow-y-auto">
            <div><label className="font-bold">Name</label><input type="text"
            value={activePersonality.name} onChange={e => handleUpdate('name',
            e.target.value)} className="w-full mt-1 p-2 bg-surface border rounded"/></div>
            <div><label className="font-bold">Persona</label><textarea
            value={activePersonality.persona} onChange={e => handleUpdate('persona',
            e.target.value)} className="w-full mt-1 p-2 bg-surface border rounded
            h-24"/></div>
            <div><label className="font-bold">Rules (one per line)</label><textarea
            value={activePersonality.rules.join('\n')} onChange={e => handleUpdate('rules',
            e.target.value.split('\n'))} className="w-full mt-1 p-2 bg-surface border
            rounded h-32"/></div>
            <div><label className="font-bold">Output Format</label><select
            value={activePersonality.outputFormat} onChange={e =>
            handleUpdate('outputFormat', e.target.value)} className="w-full mt-1 p-2 bg-
            surface border rounded"><option>markdown</option><option>json</option><option>te
            xt</option></select></div>
            <div>
                <h3 className="font-bold mb-2">Examples</h3>
                {activePersonality.exampleIO.map((ex, i) => (
                    <div key={i} className="grid grid-cols-2 gap-2 mb-2 p-2 border rounded bg-
                    surface">
                        <textarea placeholder="User Input" value={ex.input} onChange={e =>
                        handleUpdate('exampleIO', activePersonality.exampleIO.map((item, idx) => idx ===
                        i ? {...item, input: e.target.value} : item))} className="h-20 p-1 bg-background
                        border rounded"/>
                        <textarea placeholder="Model Output" value={ex.output} onChange={e =>
                        handleUpdate('exampleIO', activePersonality.exampleIO.map((item, idx) => idx ===
                        i ? {...item, output: e.target.value} : item))} className="h-20 p-1 bg-
                        background border rounded"/>
                    </div>
                ))}
                <button onClick={() => handleUpdate('exampleIO',
                [...activePersonality.exampleIO, {input: '', output: ''}])} className="text-sm
                text-primary">+ Add Example</button>
            </div>
        </div>
        {/* Testbed */}
        <div className="bg-background p-4 flex flex-col">
            <h2 className="text-lg font-bold mb-2 border-b pb-2">Live Testbed</h2>
            <div className="flex-grow overflow-y-auto space-y-4 pr-2">
               {chatHistory.map((msg, i) => (
                   <div key={i} className={`p-3 rounded-lg ${msg.role === 'user' ? 'bg-primary/10'
                   : 'bg-surface'}`}>
                        <strong className="capitalize">{msg.role}</strong>
                        <MarkdownRenderer content={msg.content} />
                   </div>
               ))}
               {isStreaming && <div className="flex justify-center"><LoadingSpinner/></div>}
            </div>
            <div className="flex gap-2 mt-4">
                <input value={testbedInput} onChange={e => setTestbedInput(e.target.value)}
                onKeyDown={e => e.key === 'Enter' && handleTestbedSend()} className="flex-grow
                p-2 bg-surface border rounded" placeholder="Test your AI..."/>
                <button onClick={handleTestbedSend} disabled={isStreaming} className="btn-
                primary px-4">Send</button>
            </div>
        </div>
    </div>
) : (
    <div className="flex-1 flex items-center justify-center text-text-
```

```
                secondary">Select or create a personality to begin.</div>
            )}
        </div>
    );
};

// ===== WeeklyDigestGenerator_6.tsx =====

import React, { useState, useCallback } from 'react';
import { generateWeeklyDigest } from '../../services/geminiService.ts';
import { sendEmail } from '../../services/workspaceService.ts';
import { useGlobalState } from '../../contexts/GlobalStateContext.tsx';
import { useNotification } from '../../contexts/NotificationContext.tsx';
import { MailIcon, SparklesIcon } from '../icons.tsx';
import { LoadingSpinner } from '../shared/index.tsx';

// Dummy data for demonstration purposes
const dummyCommitLogs = `
feat: implement user authentication
fix: resolve issue with button alignment
feat: add dark mode toggle
chore: update dependencies
refactor: simplify data fetching logic
`;
const dummyTelemetry = {
    avgPageLoad: 120,
    errorRate: '0.5%',
    uptime: '99.98%'
};

export const WeeklyDigestGenerator: React.FC = () => {
    const { state } = useGlobalState();
    const { user } = state;
    const { addNotification } = useNotification();
    const [emailHtml, setEmailHtml] = useState('');
    const [isLoading, setIsLoading] = useState(false);
    const [isSending, setIsSending] = useState(false);

    const handleGenerate = useCallback(async () => {
        setIsLoading(true);
        setEmailHtml('');
        try {
            const html = await generateWeeklyDigest(dummyCommitLogs, dummyTelemetry);
            setEmailHtml(html);
            addNotification('Digest content generated!', 'success');
        } catch (e) {
            addNotification(e instanceof Error ? e.message : 'Failed to generate digest',
            'error');
        } finally {
            setIsLoading(false);
        }
    }, []);

    const handleSend = async () => {
        if (!emailHtml || !user?.email) {
            addNotification('Cannot send email. Generate content and ensure you are signed
            in.', 'error');
            return;
        }
        setIsSending(true);
        try {
            await sendEmail(user.email, 'Your Weekly Project Digest', emailHtml);
```

```
            addNotification(`Weekly digest sent to ${user.email}!`, 'success');
        } catch (e) {
             addNotification(e instanceof Error ? e.message : 'Failed to send email',
             'error');
        } finally {
            setIsSending(false);
        }
    };

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
            <header className="mb-6">
                <h1 className="text-3xl font-bold flex items-center"><MailIcon /><span
                className="ml-3">Weekly Digest Generator</span></h1>
                <p className="text-text-secondary mt-1">Generate an AI-powered weekly summary
                and send it via your Gmail.</p>
            </header>

            <div className="flex-grow grid grid-cols-1 lg:grid-cols-2 gap-6 min-h-0">
                <div className="bg-surface p-4 border border-border rounded-lg flex flex-col
                items-center justify-center text-center">
                    <h3 className="text-lg font-bold">Generate & Send</h3>
                    <p className="text-sm text-text-secondary my-4">This tool will use dummy data to
                    generate a project summary and send it to your signed-in email address
                    ({user?.email || 'N/A'}).</p>
                    <div className="flex flex-col gap-4 w-full max-w-xs">
                        <button onClick={handleGenerate} disabled={isLoading || isSending}
                        className="btn-primary flex items-center justify-center gap-2 py-3">
                            {isLoading ? <LoadingSpinner /> : <><SparklesIcon /> Generate Digest</>}
                        </button>
                        <button onClick={handleSend} disabled={!emailHtml || isSending || isLoading ||
                        !user} className="btn-primary flex items-center justify-center gap-2 py-3 bg-
                        emerald-600">
                           {isSending ? <LoadingSpinner /> : 'Send Email via Gmail'}
                        </button>
                    </div>
                </div>

                <div className="bg-surface p-4 border border-border rounded-lg flex flex-col">
                    <h3 className="text-lg font-bold mb-2">Email Preview</h3>
                    <div className="flex-grow bg-white border rounded overflow-hidden">
                        {isLoading && <div className="flex justify-center items-center
                        h-full"><LoadingSpinner /></div>}
                        {emailHtml && <iframe srcDoc={emailHtml} title="Email Preview" className="w-full
                        h-full" />}
                        {!isLoading && !emailHtml && <div className="flex justify-center items-center
                        h-full text-text-secondary">Preview will appear here.</div>}
                    </div>
                </div>
            </div>
        </div>
    );
};

// ===== OneClickRefactor.tsx =====

import React, { useState, useCallback } from 'react';
import * as Diff from 'diff';
import { applySpecificRefactor, refactorForPerformance, refactorForReadability,
generateJsDoc, convertToFunctionalComponent } from
'../../services/aiService.ts';
import { SparklesIcon } from '../icons.tsx';
```

```
import { LoadingSpinner } from '../shared/index.tsx';

type RefactorAction = 'readability' | 'performance' | 'jsdoc' | 'functional' |
'custom';

const exampleCode = `const MyComponent = ({ data }) => {
  // A less readable component
  let transformedData = [];
  for (let i = 0; i < data.length; i++) {
    if (data[i].value > 50) {
      let item = { ...data[i], status: 'high' };
      transformedData.push(item);
    }
  }
  return (
    <div>
      {transformedData.map(d => <p key={d.id}>{d.name}</p>)}
    </div>
  );
}`;

const DiffViewer: React.FC<{ oldCode: string, newCode: string }> = ({ oldCode,
newCode }) => {
    const diff = Diff.diffLines(oldCode, newCode);

    return (
        <pre className="whitespace-pre-wrap font-mono text-xs">
            {diff.map((part, index) => {
                const color = part.added ? 'bg-green-500/20' : part.removed ? 'bg-red-500/20' :
                'bg-transparent';
                return <div key={index} className={color}>{part.value}</div>;
            })}
        </pre>
    );
};


export const OneClickRefactor: React.FC = () => {
    const [code, setCode] = useState(exampleCode);
    const [refactoredCode, setRefactoredCode] = useState('');
    const [loadingAction, setLoadingAction] = useState<RefactorAction | null>(null);

    const handleRefactor = useCallback(async (action: RefactorAction) => {
        if (!code.trim()) return;
        setLoadingAction(action);
        setRefactoredCode('');

        let stream;
        switch(action) {
            case 'readability':
                stream = refactorForReadability(code);
                break;
            case 'performance':
                stream = refactorForPerformance(code);
                break;
            case 'jsdoc':
                stream = generateJsDoc(code);
                break;
            case 'functional':
                stream = convertToFunctionalComponent(code);
                break;
            default:
```

```
            setLoadingAction(null);
            return;
        }

        try {
            let fullResponse = '';
            for await (const chunk of stream) {
                fullResponse += chunk;
                setRefactoredCode(fullResponse.replace(/^```(?:\w+\n)?/, '').replace(/```$/,
                ''));
            }
        } catch (e) {
            console.error(e);
            setRefactoredCode(`// Error during refactoring: ${e instanceof Error ? e.message
            : 'Unknown error'}`);
        } finally {
            setLoadingAction(null);
        }
    }, [code]);

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
            <header className="mb-6">
                <h1 className="text-3xl font-bold flex items-center">
                    <SparklesIcon />
                    <span className="ml-3">One-Click Refactor</span>
                </h1>
                <p className="text-text-secondary mt-1">Apply common refactoring patterns to
                your code with a single click.</p>
            </header>
            <div className="flex items-center justify-center flex-wrap gap-2 mb-4 p-4 bg-
            surface rounded-lg border border-border">
                <button onClick={() => handleRefactor('readability')} disabled={!!loadingAction}
                className="btn-primary px-3 py-1.5 text-sm">{loadingAction === 'readability' ?
                <LoadingSpinner/> : 'Improve Readability'}</button>
                <button onClick={() => handleRefactor('performance')} disabled={!!loadingAction}
                className="btn-primary px-3 py-1.5 text-sm">{loadingAction === 'performance' ?
                <LoadingSpinner/> : 'Boost Performance'}</button>
                <button onClick={() => handleRefactor('jsdoc')} disabled={!!loadingAction}
                className="btn-primary px-3 py-1.5 text-sm">{loadingAction === 'jsdoc' ?
                <LoadingSpinner/> : 'Add JSDoc'}</button>
                <button onClick={() => handleRefactor('functional')} disabled={!!loadingAction}
                className="btn-primary px-3 py-1.5 text-sm">{loadingAction === 'functional' ?
                <LoadingSpinner/> : 'To Functional Component'}</button>
            </div>
            <div className="flex-grow grid grid-cols-1 lg:grid-cols-2 gap-6 min-h-0">
                <div className="flex flex-col">
                    <label className="text-sm font-medium mb-2">Original Code</label>
                    <textarea value={code} onChange={e => setCode(e.target.value)} className="flex-
                    grow p-2 bg-surface border rounded font-mono text-xs"/>
                </div>
                 <div className="flex flex-col">
                    <label className="text-sm font-medium mb-2">Refactored Code</label>
                    <div className="flex-grow p-2 bg-background border rounded overflow-auto">
                        {loadingAction ? <div className="flex justify-center items-center
                        h-full"><LoadingSpinner/></div> : <DiffViewer oldCode={code}
                        newCode={refactoredCode} />}
                    </div>
                </div>
            </div>
        </div>
    );
```

```
};


// ===== BugReproducer.tsx =====

import React, { useState, useCallback } from 'react';
import { generateBugReproductionTestStream } from '../../services/aiService.ts';
import { BugAntIcon } from '../icons.tsx';
import { LoadingSpinner, MarkdownRenderer } from '../shared/index.tsx';

const exampleStackTrace = `TypeError: Cannot read properties of undefined
(reading 'name')
    at UserProfile (UserProfile.jsx:5:21)
    at renderWithHooks (react-dom.development.js:14985:18)
    at mountIndeterminateComponent (react-dom.development.js:17811:13)
    at beginWork (react-dom.development.js:19049:16)`;

export const BugReproducer: React.FC = () => {
    const [stackTrace, setStackTrace] = useState(exampleStackTrace);
    const [context, setContext] = useState('// The UserProfile component
    code:\nconst UserProfile = ({ user }) => <div>{user.name}</div>;');
    const [generatedTest, setGeneratedTest] = useState('');
    const [isLoading, setIsLoading] = useState(false);
    const [error, setError] = useState('');

    const handleGenerate = useCallback(async () => {
        if (!stackTrace.trim()) {
            setError('Please provide a stack trace.');
            return;
        }
        setIsLoading(true);
        setError('');
        setGeneratedTest('');
        try {
            const stream = generateBugReproductionTestStream(stackTrace, context);
            let fullResponse = '';
            for await (const chunk of stream) {
                fullResponse += chunk;
                setGeneratedTest(fullResponse);
            }
        } catch (err) {
            setError(err instanceof Error ? err.message : 'An unknown error occurred.');
        } finally {
            setIsLoading(false);
        }
    }, [stackTrace, context]);

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
            <header className="mb-6">
                <h1 className="text-3xl font-bold flex items-center">
                    <BugAntIcon />
                    <span className="ml-3">Automated Bug Reproducer</span>
                </h1>
                <p className="text-text-secondary mt-1">Paste a stack trace to automatically
                generate a failing unit test.</p>
            </header>
            <div className="flex-grow grid grid-cols-1 lg:grid-cols-2 gap-6 min-h-0">
                <div className="flex flex-col gap-4">
                    <div className="flex flex-col flex-1 min-h-0">
                        <label htmlFor="stack-trace" className="text-sm font-medium mb-2">Stack
                        Trace</label>
```

```
                        <textarea id="stack-trace" value={stackTrace} onChange={e =>
                        setStackTrace(e.target.value)} className="flex-grow p-2 bg-surface border
                        rounded font-mono text-xs"/>
                    </div>
                     <div className="flex flex-col flex-1 min-h-0">
                        <label htmlFor="context" className="text-sm font-medium mb-2">Relevant Code /
                        Context (Optional)</label>
                        <textarea id="context" value={context} onChange={e =>
                        setContext(e.target.value)} className="flex-grow p-2 bg-surface border rounded
                        font-mono text-xs"/>
                    </div>
                    <button onClick={handleGenerate} disabled={isLoading} className="btn-primary
                    w-full py-3">{isLoading ? <LoadingSpinner/> : 'Generate Test'}</button>
                </div>
                <div className="flex flex-col">
                    <label className="text-sm font-medium mb-2">Generated Test File</label>
                    <div className="flex-grow p-1 bg-background border rounded overflow-auto">
                        {isLoading && !generatedTest && <div className="flex justify-center items-center
                        h-full"><LoadingSpinner /></div>}
                        {error && <p className="text-red-500 p-4">{error}</p>}
                        {generatedTest && <MarkdownRenderer content={generatedTest} />}
                    </div>
                </div>
            </div>
        </div>
    );
};


// ===== TechDebtSonar.tsx =====

import React, { useState, useCallback } from 'react';
import { detectCodeSmells } from '../../services/aiService.ts';
import type { CodeSmell } from '../../types.ts';
import { MagnifyingGlassIcon } from '../icons.tsx';
import { LoadingSpinner } from '../shared/index.tsx';

const exampleCode = `class DataProcessor {
    process(data) {
        // Long method with multiple responsibilities
        if (data.type === 'A') {
            const results = [];
            for (let i = 0; i < data.items.length; i++) {
                // complex logic
                const item = data.items[i];
                if(item.value > 100) {
                   results.push({ ...item, status: 'processed' });
                }
            }
            return results;
        } else {
            // Duplicated logic
            const results = [];
            for (let i = 0; i < data.items.length; i++) {
                const item = data.items[i];
                 if(item.value > 100) {
                   results.push({ ...item, status: 'processed_special' });
                }
            }
            return results;
        }
    }
```

```
}`;

export const TechDebtSonar: React.FC = () => {
    const [code, setCode] = useState(exampleCode);
    const [smells, setSmells] = useState<CodeSmell[]>([]);
    const [isLoading, setIsLoading] = useState(false);
    const [error, setError] = useState('');

    const handleScan = useCallback(async () => {
        if (!code.trim()) {
            setError('Please provide code to scan.');
            return;
        }
        setIsLoading(true);
        setError('');
        setSmells([]);
        try {
            const result = await detectCodeSmells(code);
            setSmells(result);
        } catch (err) {
            setError(err instanceof Error ? err.message : 'An unknown error occurred.');
        } finally {
            setIsLoading(false);
        }
    }, [code]);

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
            <header className="mb-6">
                <h1 className="text-3xl font-bold flex items-center">
                    <MagnifyingGlassIcon />
                    <span className="ml-3">Tech Debt Sonar</span>
                </h1>
                <p className="text-text-secondary mt-1">Scan code to find code smells and areas
                with high complexity.</p>
            </header>
            <div className="flex-grow grid grid-cols-1 lg:grid-cols-2 gap-6 min-h-0">
                <div className="flex flex-col">
                    <label className="text-sm font-medium mb-2">Code to Analyze</label>
                    <textarea value={code} onChange={e => setCode(e.target.value)} className="flex-
                    grow p-2 bg-surface border rounded font-mono text-xs"/>
                    <button onClick={handleScan} disabled={isLoading} className="btn-primary w-full
                    mt-4 py-3">{isLoading ? <LoadingSpinner/> : 'Scan for Code Smells'}</button>
                </div>
                <div className="flex flex-col">
                    <label className="text-sm font-medium mb-2">Detected Smells</label>
                    <div className="flex-grow p-2 bg-background border rounded overflow-auto">
                        {isLoading && <div className="flex justify-center items-center
                        h-full"><LoadingSpinner /></div>}
                        {error && <p className="text-red-500 p-4">{error}</p>}
                        {!isLoading && smells.length === 0 && <p className="text-text-secondary text-
                        center pt-8">No smells detected, or scan not run.</p>}
                        {smells.length > 0 && (
                            <div className="space-y-3">
                                {smells.map((smell, i) => (
                                    <div key={i} className="p-3 bg-surface border border-border rounded-lg">
                                        <div className="flex justify-between items-center">
                                            <h4 className="font-bold text-primary">{smell.smell}</h4>
                                            <span className="text-xs font-mono bg-gray-100 dark:bg-slate-700 px-2 py-1
                                            rounded">Line: {smell.line}</span>
                                        </div>
                                        <p className="text-sm mt-1">{smell.explanation}</p>
```

```
                                    </div>
                                ))}
                            </div>
                        )}
                    </div>
                </div>
            </div>
        </div>
    );
};


// ===== IamPolicyGenerator.tsx =====

import React, { useState, useCallback } from 'react';
import { generateIamPolicyStream } from '../../services/aiService.ts';
import { ShieldCheckIcon } from '../icons.tsx';
import { LoadingSpinner, MarkdownRenderer } from '../shared/index.tsx';

export const IamPolicyGenerator: React.FC = () => {
    const [description, setDescription] = useState('A user role that can read from
    S3 buckets but not write or delete.');
    const [platform, setPlatform] = useState<'aws' | 'gcp'>('aws');
    const [policy, setPolicy] = useState('');
    const [isLoading, setIsLoading] = useState(false);
    const [error, setError] = useState('');

    const handleGenerate = useCallback(async () => {
        if (!description.trim()) {
            setError('Please provide a description.');
            return;
        }
        setIsLoading(true);
        setError('');
        setPolicy('');
        try {
            const stream = generateIamPolicyStream(description, platform);
            let fullResponse = '';
            for await (const chunk of stream) {
                fullResponse += chunk;
                setPolicy(fullResponse);
            }
        } catch (err) {
            setError(err instanceof Error ? err.message : 'An unknown error occurred.');
        } finally {
            setIsLoading(false);
        }
    }, [description, platform]);

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
            <header className="mb-6">
                <h1 className="text-3xl font-bold flex items-center">
                    <ShieldCheckIcon />
                    <span className="ml-3">IAM Policy Generator</span>
                </h1>
                <p className="text-text-secondary mt-1">Generate AWS or GCP IAM policies from a
                natural language description.</p>
            </header>
            <div className="flex-grow grid grid-cols-1 lg:grid-cols-2 gap-6 min-h-0">
                <div className="flex flex-col gap-4">
                     <div>
```

```
                        <label htmlFor="platform" className="text-sm font-medium mb-2 block">Cloud
                        Platform</label>
                        <div className="flex gap-2 p-1 bg-surface rounded-lg border">
                            <button onClick={() => setPlatform('aws')} className={`flex-1 py-2 rounded-md
                            text-sm ${platform === 'aws' ? 'bg-primary text-text-on-primary' :
                            ''}`}>AWS</button>
                            <button onClick={() => setPlatform('gcp')} className={`flex-1 py-2 rounded-md
                            text-sm ${platform === 'gcp' ? 'bg-primary text-text-on-primary' :
                            ''}`}>GCP</button>
                        </div>
                    </div>
                    <div className="flex flex-col flex-1 min-h-0">
                        <label htmlFor="description" className="text-sm font-medium mb-2">Describe the
                        desired permissions</label>
                        <textarea id="description" value={description} onChange={e =>
                        setDescription(e.target.value)} className="flex-grow p-2 bg-surface border
                        rounded text-sm"/>
                    </div>
                    <button onClick={handleGenerate} disabled={isLoading} className="btn-primary
                    w-full py-3">{isLoading ? <LoadingSpinner/> : 'Generate Policy'}</button>
                </div>
                <div className="flex flex-col">
                    <label className="text-sm font-medium mb-2">Generated Policy (JSON)</label>
                    <div className="flex-grow p-1 bg-background border rounded overflow-auto">
                        {isLoading && !policy && <div className="flex justify-center items-center
                        h-full"><LoadingSpinner /></div>}
                        {error && <p className="text-red-500 p-4">{error}</p>}
                        {policy && <MarkdownRenderer content={policy} />}
                    </div>
                </div>
            </div>
        </div>
    );
};


// ===== ThemeDesigner_30.tsx =====

import React, { useState, useCallback, useEffect } from 'react';
import { SparklesIcon, ArrowDownTrayIcon, PhotoIcon } from '../icons.tsx';
import { generateSemanticTheme } from '../../services/index.ts';
import { fileToBase64 } from '../../services/fileUtils.ts';
import type { SemanticColorTheme, ColorTheme } from '../../types.ts';
import { LoadingSpinner } from '../shared/index.tsx';
import { useTheme } from '../../hooks/useTheme.ts';

const ColorDisplay: React.FC<{ name: string; color: { name: string; value:
string; } }> = ({ name, color }) => (
    <div className="flex items-center justify-between p-2 bg-background rounded-md
    border border-border">
        <div className="flex items-center gap-3">
            <div className="w-6 h-6 rounded-full border border-border" style={{
            backgroundColor: color.value }} />
            <div>
                <p className="text-sm font-semibold text-text-primary capitalize">{name}</p>
                <p className="text-xs text-text-secondary">{color.name}</p>
            </div>
        </div>
        <span className="font-mono text-sm text-text-secondary">{color.value}</span>
    </div>
);
const AccessibilityCheck: React.FC<{ name: string, check: { ratio: number;
```

```
score: string; } }> = ({ name, check }) => {
    const scoreColor = check.score === 'AAA' ? 'text-green-600' : check.score ===
    'AA' ? 'text-emerald-600' : 'text-red-600';
    return (
        <div className="flex items-center justify-between p-2 bg-background rounded-md
        border border-border text-sm">
            <p className="text-text-secondary">{name}</p>
            <div className="flex items-center gap-2">
                <span className="font-mono">{check.ratio.toFixed(2)}</span>
                <span className={`font-bold px-2 py-0.5 rounded-full text-xs ${scoreColor}
                ${scoreColor.replace('text-', 'bg-')}/10`}>{check.score}</span>
            </div>
        </div>
    );
}

export const ThemeDesigner: React.FC = () => {
    const [theme, setTheme] = useState<SemanticColorTheme | null>(null);
    const [prompt, setPrompt] = useState('A calming, minimalist theme for a blog');
    const [image, setImage] = useState<{ base64: string, name: string } |
    null>(null);
    const [isLoading, setIsLoading] = useState(false);
    const [error, setError] = useState('');
    const [, , applyCustomTheme] = useTheme();

    const handleGenerate = useCallback(async () => {
        const textPart = { text: `Generate a theme based on this description:
        "${prompt}"` };
        const imagePart = image ? { inlineData: { mimeType: 'image/png', data:
        image.base64 } } : null;
        const parts = imagePart ? [textPart, imagePart] : [textPart];

        setIsLoading(true); setError('');
        try {
            const newTheme = await generateSemanticTheme({ parts });
            setTheme(newTheme);
        } catch (err) {
            setError(err instanceof Error ? err.message : "An unknown error occurred.");
        } finally {
            setIsLoading(false);
        }
    }, [prompt, image]);

    const handleFileChange = async (e: React.ChangeEvent<HTMLInputElement>) => {
        const file = e.target.files?.[0];
        if (file) {
            const base64 = await fileToBase64(file);
            setImage({ base64, name: file.name });
            setPrompt(`A theme based on the uploaded image: ${file.name}`);
        }
    };

    useEffect(() => { handleGenerate(); }, []);

    const handleApplyTheme = () => {
        if (!theme) return;
        const colorsToApply: ColorTheme = {
            primary: theme.palette.primary.value,
            background: theme.theme.background.value,
            surface: theme.theme.surface.value,
            textPrimary: theme.theme.textPrimary.value,
            textSecondary: theme.theme.textSecondary.value,
```

```
        textOnPrimary: theme.theme.textOnPrimary.value,
        border: theme.theme.border.value,
    };
    applyCustomTheme(colorsToApply, theme.mode);
};

return (
    <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
        <header className="mb-6">
            <h1 className="text-3xl font-bold flex items-center"><SparklesIcon /><span
            className="ml-3">AI Theme Designer</span></h1>
            <p className="text-text-secondary mt-1">Generate a full design system from a
            description or image.</p>
        </header>
        <div className="flex-grow grid grid-cols-1 md:grid-cols-2 gap-6 min-h-0">
            <div className="md:col-span-1 flex flex-col gap-4 bg-surface border border-
            border p-6 rounded-lg overflow-y-auto">
                <h3 className="text-xl font-bold">Describe or Upload</h3>
                <textarea value={prompt} onChange={e => setPrompt(e.target.value)}
                className="p-2 bg-background border border-border rounded-md resize-none text-sm
                h-24" placeholder="e.g., A light, airy theme for a blog" />
                 <div className="relative border border-dashed border-border rounded-lg p-4 text-
                 center">
                    <input type="file" onChange={handleFileChange} className="absolute inset-0
                    w-full h-full opacity-0 cursor-pointer" />
                    <PhotoIcon/>
                    <p className="text-sm mt-1">{image ? `Image: ${image.name}` : 'Upload an image
                    (optional)'}</p>
                </div>
                <div className="flex gap-2">
                    <button onClick={handleGenerate} disabled={isLoading} className="btn-primary
                    flex-grow flex items-center justify-center gap-2 px-4 py-2">
                        {isLoading ? <LoadingSpinner /> : 'Generate New Theme'}
                    </button>
                     <button onClick={handleApplyTheme} disabled={isLoading || !theme}
                     className="px-4 py-2 bg-emerald-600 text-white font-bold rounded-md
                     hover:opacity-90 transition-all disabled:opacity-50 shadow-md">
                        Apply to App
                    </button>
                </div>
                {error && <p className="text-red-500 text-xs text-center">{error}</p>}

                {theme && !isLoading && (
                    <div className="mt-4 border-t border-border pt-4 space-y-4">
                        <div><h3 className="text-lg font-bold mb-2">Palette</h3><div
                        className="space-y-2"><ColorDisplay name="Primary"
                        color={theme.palette.primary}/><ColorDisplay name="Secondary"
                        color={theme.palette.secondary}/><ColorDisplay name="Accent"
                        color={theme.palette.accent}/><ColorDisplay name="Neutral"
                        color={theme.palette.neutral}/></div></div>
                        <div><h3 className="text-lg font-bold mb-2">Theme Roles</h3><div
                        className="space-y-2"><ColorDisplay name="Background"
                        color={theme.theme.background}/><ColorDisplay name="Surface"
                        color={theme.theme.surface}/><ColorDisplay name="Text Primary"
                        color={theme.theme.textPrimary}/><ColorDisplay name="Text Secondary"
                        color={theme.theme.textSecondary}/><ColorDisplay name="Text on Primary"
                        color={theme.theme.textOnPrimary}/><ColorDisplay name="Border"
                        color={theme.theme.border}/></div></div>
                        <div><h3 className="text-lg font-bold mb-2">Accessibility (WCAG 2.1)</h3><div
                        className="space-y-2"><AccessibilityCheck name="Primary on Surface"
                        check={theme.accessibility.primaryOnSurface}/><AccessibilityCheck name="Text on
                        Surface" check={theme.accessibility.textPrimaryOnSurface}/><AccessibilityCheck
```

```
                            name="Subtle Text on Surface"
                            check={theme.accessibility.textSecondaryOnSurface}/><AccessibilityCheck
                            name="Text on Primary"
                            check={theme.accessibility.textOnPrimaryOnPrimary}/></div></div>
                        </div>
                    )}
                </div>
                <div className="md:col-span-1 rounded-lg p-8 overflow-y-auto border border-
                border" style={{ backgroundColor: theme?.theme.background.value, color:
                theme?.theme.textPrimary.value }}>
                     <h3 className="text-2xl font-bold mb-6">Live Preview</h3>
                     {theme ? (
                         <div className="p-6 rounded-lg grid grid-cols-1 md:grid-cols-2 gap-6" style={{
                         backgroundColor: theme.theme.surface.value }}>
                            <div className="space-y-4">
                                <h4 className="text-lg font-bold">Sample Card</h4>
                                <p className="text-sm" style={{color: theme.theme.textSecondary.value}}>This is
                                a sample card to demonstrate the theme colors. It contains a primary button and
                                some secondary text.</p>
                                <button className="px-4 py-2 rounded-md font-bold transition-colors" style={{
                                backgroundColor: theme.palette.primary.value, color:
                                theme.theme.textOnPrimary.value }}>Primary Button</button>
                            </div>
                             <div className="space-y-4">
                                <input type="text" placeholder="Text input" className="w-full px-3 py-2 rounded-
                                md border" style={{backgroundColor: theme.theme.background.value, borderColor:
                                theme.theme.border.value, color: theme.theme.textPrimary.value}} />
                                <div className="p-3 border rounded" style={{borderColor:
                                theme.theme.border.value, color: theme.theme.textSecondary.value}}>
                                    <p>A bordered container.</p>
                                </div>
                             </div>
                         </div>
                     ) : <div className="flex items-center justify-center h-full text-text-
                     secondary">Theme preview will appear here.</div>}
                </div>
            </div>
        </div>
    );
};

// ===== AiCodeExplainer_28.tsx =====

import React, { useState, useCallback, useEffect, useMemo, useRef } from
'react';
import mermaid from 'mermaid';
import { explainCodeStructured, generateMermaidJs } from
'../../services/index.ts';
import type { StructuredExplanation } from '../../types.ts';
import { CpuChipIcon } from '../icons.tsx';
import { MarkdownRenderer, LoadingSpinner } from '../shared/index.tsx';

const exampleCode = `const bubbleSort = (arr) => {
  for (let i = 0; i < arr.length; i++) {
    for (let j = 0; j < arr.length - i - 1; j++) {
      if (arr[j] > arr[j + 1]) {
        [arr[j], arr[j + 1]] = [arr[j + 1], arr[j]];
      }
    }
  }
  return arr;
};`;
```

```
type ExplanationTab = 'summary' | 'lineByLine' | 'complexity' | 'suggestions' |
'flowchart';

const simpleSyntaxHighlight = (code: string) => {
    const escapedCode = code
        .replace(/&/g, '&amp;')
        .replace(/</g, '&lt;')
        .replace(/>/g, '&gt;');

    return escapedCode
        .replace(/\b(const|let|var|function|return|if|for|=>|import|from|export|default)
        \b/g, '<span class="text-indigo-400 font-semibold">$1</span>')
        .replace(/(\`|'|")(.*?)(\`|'|")/g, '<span class="text-
        emerald-400">$1$2$3</span>')
        .replace(/(\/\/.*)/g, '<span class="text-gray-400 italic">$1</span>')
        .replace(/(\{|\}|\(|\)|\[|\])/g, '<span class="text-gray-400">$1</span>');
};

mermaid.initialize({ startOnLoad: false, theme: 'neutral', securityLevel:
'loose' });

export const AiCodeExplainer: React.FC<{ initialCode?: string }> = ({
initialCode }) => {
    const [code, setCode] = useState<string>(initialCode || exampleCode);
    const [explanation, setExplanation] = useState<StructuredExplanation |
    null>(null);
    const [mermaidCode, setMermaidCode] = useState<string>('');
    const [isLoading, setIsLoading] = useState<boolean>(false);
    const [error, setError] = useState<string>('');
    const [activeTab, setActiveTab] = useState<ExplanationTab>('summary');
    const textareaRef = useRef<HTMLTextAreaElement>(null);
    const preRef = useRef<HTMLPreElement>(null);
    const mermaidContainerRef = useRef<HTMLDivElement>(null);

    const handleExplain = useCallback(async (codeToExplain: string) => {
        if (!codeToExplain.trim()) {
            setError('Please enter some code to explain.');
            return;
        }
        setIsLoading(true);
        setError('');
        setExplanation(null);
        setMermaidCode('');
        setActiveTab('summary');
        try {
            const [explanationResult, mermaidResult] = await Promise.all([
                explainCodeStructured(codeToExplain),
                generateMermaidJs(codeToExplain)
            ]);
            setExplanation(explanationResult);
            setMermaidCode(mermaidResult.replace(/```mermaid\n|```/g, ''));

        } catch (err) {
            const errorMessage = err instanceof Error ? err.message : 'An unknown error
            occurred.';
            setError(`Failed to get explanation: ${errorMessage}`);
        } finally {
            setIsLoading(false);
        }
    }, []);

    useEffect(() => {
```

```
    if (initialCode) {
        setCode(initialCode);
        handleExplain(initialCode);
    }
}, [initialCode, handleExplain]);

useEffect(() => {
    const renderMermaid = async () => {
         if (activeTab === 'flowchart' && mermaidCode && mermaidContainerRef.current) {
            try {
                mermaidContainerRef.current.innerHTML = ''; // Clear previous
                const { svg } = await mermaid.render(`mermaid-graph-${Date.now()}`,
                mermaidCode);
                mermaidContainerRef.current.innerHTML = svg;
            } catch (e) {
                console.error("Mermaid rendering error:", e);
                mermaidContainerRef.current.innerHTML = `<p class="text-red-500">Error rendering
                flowchart.</p>`;
            }
        }
    }
    renderMermaid();
}, [activeTab, mermaidCode]);


const handleScroll = () => {
    if (preRef.current && textareaRef.current) {
        preRef.current.scrollTop = textareaRef.current.scrollTop;
        preRef.current.scrollLeft = textareaRef.current.scrollLeft;
    }
};

const highlightedCode = useMemo(() => simpleSyntaxHighlight(code), [code]);

const renderTabContent = () => {
    if (!explanation) return null;
    switch(activeTab) {
        case 'summary':
            return <MarkdownRenderer content={explanation.summary} />;
        case 'lineByLine':
            return (
                <div className="space-y-3">
                    {explanation.lineByLine.map((item, index) => (
                        <div key={index} className="p-3 bg-background rounded-md border border-border">
                            <p className="font-mono text-xs text-primary mb-1">Lines: {item.lines}</p>
                            <p className="text-sm">{item.explanation}</p>
                        </div>
                    ))}
                </div>
            );
        case 'complexity':
            return (
                <div>
                    <p><strong>Time Complexity:</strong> <span className="font-mono text-
                    amber-600">{explanation.complexity.time}</span></p>
                    <p><strong>Space Complexity:</strong> <span className="font-mono text-
                    amber-600">{explanation.complexity.space}</span></p>
                </div>
            );
        case 'suggestions':
            return (
                 <ul className="list-disc list-inside space-y-2">
```

```
                    {explanation.suggestions.map((item, index) => <li key={index}>{item}</li>)}
                </ul>
            );
        case 'flowchart':
            return (
                <div ref={mermaidContainerRef} className="w-full h-full flex items-center
                justify-center">
                    <LoadingSpinner />
                </div>
            );
    }
}

return (
    <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
        <header className="mb-6 flex-shrink-0">
            <h1 className="text-3xl font-bold flex items-center">
                <CpuChipIcon />
                <span className="ml-3">AI Code Explainer</span>
            </h1>
            <p className="text-text-secondary mt-1">Get a detailed, structured analysis of
            any code snippet.</p>
        </header>
        <div className="flex-grow grid grid-cols-1 md:grid-cols-2 gap-6 min-h-0">

            {/* Left Column: Code Input */}
            <div className="flex flex-col min-h-0 md:col-span-1">
                <label htmlFor="code-input" className="text-sm font-medium text-text-secondary
                mb-2">Your Code</label>
                <div className="relative flex-grow bg-surface border border-border rounded-md
                focus-within:ring-2 focus-within:ring-primary overflow-hidden">
                    <textarea
                        ref={textareaRef}
                        id="code-input"
                        value={code}
                        onChange={(e) => setCode(e.target.value)}
                        onScroll={handleScroll}
                        placeholder="Paste your code here..."
                        spellCheck="false"
                        className="absolute inset-0 w-full h-full p-4 bg-transparent resize-none font-
                        mono text-sm text-transparent caret-primary outline-none z-10"
                    />
                    <pre
                        ref={preRef}
                        aria-hidden="true"
                        className="absolute inset-0 w-full h-full p-4 font-mono text-sm text-text-
                        primary pointer-events-none z-0 whitespace-pre-wrap overflow-auto no-scrollbar"
                        dangerouslySetInnerHTML={{ __html: highlightedCode + '\n' }}
                    />
                </div>
                <div className="mt-4 flex-shrink-0">
                    <button
                        onClick={() => handleExplain(code)}
                        disabled={isLoading}
                        className="btn-primary w-full flex items-center justify-center px-6 py-3"
                    >
                        {isLoading ? <LoadingSpinner/> : 'Analyze Code'}
                    </button>
                </div>
            </div>

            {/* Right Column: AI Analysis */}
```

```
                <div className="flex flex-col min-h-0 md:col-span-1">
                    <label className="text-sm font-medium text-text-secondary mb-2">AI
                    Analysis</label>
                    <div className="relative flex-grow flex flex-col bg-surface border border-border
                    rounded-md overflow-hidden">
                        <div className="flex-shrink-0 flex border-b border-border">
                           {(['summary', 'lineByLine', 'complexity', 'suggestions', 'flowchart'] as
                           ExplanationTab[]).map(tab => (
                               <button key={tab} onClick={() => setActiveTab(tab)} disabled={!explanation}
                                className={`px-4 py-2 text-sm font-medium capitalize transition-colors
                                ${activeTab === tab ? 'bg-background text-primary font-semibold' : 'text-text-
                                secondary hover:bg-gray-100 dark:hover:bg-slate-700 disabled:text-gray-400
                                dark:disabled:text-slate-500'}`}>
                                   {tab.replace(/([A-Z])/g, ' $1')}
                               </button>
                           ))}
                        </div>
                        <div className="p-4 flex-grow overflow-y-auto">
                            {isLoading && <div className="flex items-center justify-center
                            h-full"><LoadingSpinner /></div>}
                            {error && <p className="text-red-500">{error}</p>}
                            {explanation && !isLoading && renderTabContent()}
                            {!isLoading && !explanation && !error && <div className="text-text-secondary
                            h-full flex items-center justify-center">The analysis will appear here.</div>}
                        </div>
                    </div>
                </div>
            </div>
        );
};

// ===== AiFeatureBuilder_30.tsx =====

import React, { useState, useCallback, useEffect } from 'react';
import type { GeneratedFile } from '../../types.ts';
import { generateFeature, generateFullStackFeature, generateUnitTestsStream,
generateCommitMessageStream, generateDockerfile } from
'../../services/aiService.ts';
import { saveFile, getAllFiles, clearAllFiles } from
'../../services/dbService.ts';
import { useNotification } from '../../contexts/NotificationContext.tsx';
import { CpuChipIcon, DocumentTextIcon, BeakerIcon, GitBranchIcon, CloudIcon }
from '../icons.tsx';
import { LoadingSpinner, MarkdownRenderer } from '../shared/index.tsx';

type SupplementalTab = 'TESTS' | 'COMMIT' | 'DEPLOYMENT' | 'CODE';
type OutputTab = GeneratedFile | SupplementalTab;

export const AiFeatureBuilder: React.FC = () => {
    const [prompt, setPrompt] = useState<string>('A simple "Hello World" React
    component with a button that shows an alert.');
    const [framework] = useState('React');
    const [styling] = useState('Tailwind CSS');
    const [includeBackend, setIncludeBackend] = useState(false);

    const [generatedFiles, setGeneratedFiles] = useState<GeneratedFile[]>([]);
    const [unitTests, setUnitTests] = useState<string>('');
    const [commitMessage, setCommitMessage] = useState<string>('');
    const [dockerfile, setDockerfile] = useState<string>('');

    const [activeTab, setActiveTab] = useState<OutputTab>('CODE');
```

```
const [isLoading, setIsLoading] = useState<boolean>(false);
const [error, setError] = useState<string>('');

useEffect(() => {
    const loadFiles = async () => {
        const files = await getAllFiles();
        setGeneratedFiles(files);
        if (files.length > 0) setActiveTab(files[0]);
    };
    loadFiles();
}, []);

const handleGenerate = useCallback(async () => {
    if (!prompt.trim()) { setError('Please enter a feature description.'); return; }
    setIsLoading(true);
    setError('');
    await clearAllFiles();
    setGeneratedFiles([]); setUnitTests(''); setCommitMessage('');
    setDockerfile(''); setActiveTab('CODE');

    try {
        const resultFiles = includeBackend
            ? await generateFullStackFeature(prompt, framework, styling)
            : await generateFeature(prompt, framework, styling);

        for (const file of resultFiles) { await saveFile(file); }
        setGeneratedFiles(resultFiles);

        if (resultFiles.length > 0) {
            const componentFile = resultFiles.find(f => f.filePath.endsWith('.tsx') ||
            f.filePath.endsWith('.jsx'));
            setActiveTab(componentFile || resultFiles[0]);

            const testStream = generateUnitTestsStream(componentFile?.content ||
            resultFiles[0].content);
            const diffContext = resultFiles.map(f => `File:
            ${f.filePath}\n\n${f.content}`).join('\n---\n');
            const commitStream = generateCommitMessageStream(diffContext);

            let tests = ''; for await (const chunk of testStream) { tests += chunk;
            setUnitTests(tests); }
            let commit = ''; for await (const chunk of commitStream) { commit += chunk;
            setCommitMessage(commit); }

            if (!includeBackend) {
                const dockerfileStream = generateDockerfile(framework);
                let docker = ''; for await (const chunk of dockerfileStream) { docker += chunk;
                setDockerfile(docker); }
            }
        }
    } catch (err) {
        setError(err instanceof Error ? err.message : 'Failed to generate feature.');
    } finally {
        setIsLoading(false);
    }
}, [prompt, framework, styling, includeBackend]);

const renderContent = () => {
    if (typeof activeTab === 'string') {
        switch (activeTab) {
            case 'TESTS': return <MarkdownRenderer content={unitTests} />;
            case 'COMMIT': return <pre className="w-full h-full p-4 whitespace-pre-wrap
```

```
            font-sans text-sm">{commitMessage}</pre>;
            case 'DEPLOYMENT': return <MarkdownRenderer content={dockerfile} />;
            default: return <div className="p-4">Select a file</div>;
        }
    }
    return <MarkdownRenderer content={'```tsx\n' + activeTab.content + '\n```'} />;
}

return (
    <div className="h-full flex flex-col text-text-primary bg-surface">
        <header className="p-4 border-b border-border flex-shrink-0">
            <h1 className="text-xl font-bold flex items-center"><CpuChipIcon /><span
            className="ml-3">AI Feature Builder</span></h1>
        </header>
        <div className="flex-grow flex min-h-0">
            <main className="flex-1 flex flex-col min-w-0">
                <div className="flex-grow flex flex-col bg-background">
                     <div className="border-b border-border flex items-center bg-surface overflow-x-
                     auto">
                        {generatedFiles.map(file => (
                            <button key={file.filePath} onClick={() => setActiveTab(file)} className={`flex-
                            shrink-0 flex items-center gap-2 px-4 py-2 text-sm ${activeTab === file ? 'bg-
                            background border-b-2 border-primary text-text-primary' : 'text-text-secondary
                            hover:bg-gray-50'}`}><DocumentTextIcon /> {file.filePath}</button>
                        ))}
                        {unitTests && <button onClick={() => setActiveTab('TESTS')} className={`flex-
                        shrink-0 flex items-center gap-2 px-4 py-2 text-sm ${activeTab === 'TESTS' ?
                        'bg-background border-b-2 border-primary text-text-primary' : 'text-text-
                        secondary hover:bg-gray-50'}`}><BeakerIcon /> Tests</button>}
                        {commitMessage && <button onClick={() => setActiveTab('COMMIT')}
                        className={`flex-shrink-0 flex items-center gap-2 px-4 py-2 text-sm ${activeTab
                        === 'COMMIT' ? 'bg-background border-b-2 border-primary text-text-primary' :
                        'text-text-secondary hover:bg-gray-50'}`}><GitBranchIcon /> Commit</button>}
                        {dockerfile && !includeBackend && <button onClick={() =>
                        setActiveTab('DEPLOYMENT')} className={`flex-shrink-0 flex items-center gap-2
                        px-4 py-2 text-sm ${activeTab === 'DEPLOYMENT' ? 'bg-background border-b-2
                        border-primary text-text-primary' : 'text-text-secondary hover:bg-
                        gray-50'}`}><CloudIcon /> Dockerfile</button>}
                    </div>
                    <div className="flex-grow p-2 overflow-auto">
                        {isLoading && !generatedFiles.length ? <div className="flex justify-center
                        items-center h-full"><LoadingSpinner/></div> : renderContent()}
                    </div>
                </div>

                <div className="flex-shrink-0 p-4 border-t border-border bg-surface">
                     <div className="flex items-center gap-2 mb-2">
                        <label className="flex items-center gap-2 text-sm"><input type="checkbox"
                        checked={includeBackend} onChange={e => setIncludeBackend(e.target.checked)} />
                        Include Backend (Cloud Function + Firestore)</label>
                    </div>
                    <textarea value={prompt} onChange={(e) => setPrompt(e.target.value)}
                    placeholder="e.g., A user profile card with an avatar, name, and bio."
                    className="w-full p-2 bg-background border border-border rounded-md resize-none
                    text-sm h-20"/>
                     <div className="flex gap-2 mt-2">
                         <button onClick={handleGenerate} disabled={isLoading} className="btn-primary
                         flex-grow flex items-center justify-center gap-2 px-4 py-2">
                            {isLoading ? <><LoadingSpinner /> Generating...</> : 'Generate Feature'}
                        </button>
                     </div>
                     {error && <p className="text-red-600 text-xs mt-2 text-center">{error}</p>}
```

```
                    </div>
                </main>
            </div>
        </div>
    );
};


// ===== ProjectExplorer_28.tsx =====

import React, { useState, useEffect, useCallback } from 'react';
import { useGlobalState } from '../../contexts/GlobalStateContext.tsx';
import { useNotification } from '../../contexts/NotificationContext.tsx';
import { initializeOctokit } from '../../services/authService.ts';
import { getUserToken } from '../../services/firebaseService.ts';
import { getRepos, getRepoTree, getFileContent, commitFiles } from
'../../services/githubService.ts';
import { generateCommitMessageStream } from '../../services/index.ts';
import type { Repo, FileNode } from '../../types.ts';
import { FolderIcon, DocumentIcon } from '../icons.tsx';
import { LoadingSpinner } from '../shared/index.tsx';
import * as Diff from 'diff';

const FileTree: React.FC<{ node: FileNode, onFileSelect: (path: string, name:
string) => void, activePath: string | null }> = ({ node, onFileSelect,
activePath }) => {
    const [isOpen, setIsOpen] = useState(true);

    if (node.type === 'file') {
        const isActive = activePath === node.path;
        return (
            <div
                className={`flex items-center space-x-2 pl-4 py-1 cursor-pointer rounded
                ${isActive ? 'bg-primary/10 text-primary' : 'hover:bg-gray-100 dark:hover:bg-
                slate-700'}`}
                onClick={() => onFileSelect(node.path, node.name)}
            >
                <DocumentIcon />
                <span>{node.name}</span>
            </div>
        );
    }

    return (
        <div>
            <div
                className="flex items-center space-x-2 py-1 cursor-pointer hover:bg-gray-100
                dark:hover:bg-slate-700 rounded"
                onClick={() => setIsOpen(!isOpen)}
            >
                <div className={`transform transition-transform ${isOpen ? 'rotate-90' :
                ''}`}>■</div>
                <FolderIcon />
                <span className="font-semibold">{node.name}</span>
            </div>
            {isOpen && node.children && (
                <div className="pl-4 border-l border-border ml-3">
                    {node.children.map(child => <FileTree key={child.path} node={child}
                    onFileSelect={onFileSelect} activePath={activePath} />)}
                </div>
            )}
        </div>
```

```
    );
};

export const ProjectExplorer: React.FC = () => {
    const { state, dispatch } = useGlobalState();
    const { user, githubUser, selectedRepo, projectFiles } = state;
    const { addNotification } = useNotification();
    const [repos, setRepos] = useState<Repo[]>([]);
    const [isLoading, setIsLoading] = useState<'repos' | 'tree' | 'file' | 'commit'
    | null>(null);
    const [error, setError] = useState('');
    const [activeFile, setActiveFile] = useState<{ path: string; name: string;
    originalContent: string; editedContent: string} | null>(null);

    const getApiClient = useCallback(async () => {
        if (!user) {
            throw new Error("You must be logged in to use the Project Explorer.");
        }
        const token = await getUserToken(user.uid, 'github_pat');
        if (!token) {
            throw new Error("GitHub token not found. Please add it on the Connections
            page.");
        }
        return initializeOctokit(token);
    }, [user]);


    useEffect(() => {
        const loadRepos = async () => {
            if (user && githubUser) {
                setIsLoading('repos');
                setError('');
                try {
                    const octokit = await getApiClient();
                    const userRepos = await getRepos(octokit);
                    setRepos(userRepos);
                } catch (err) {
                    setError(err instanceof Error ? err.message : 'Failed to load repositories');
                } finally {
                    setIsLoading(null);
                }
            } else {
                setRepos([]);
            }
        };
        loadRepos();
    }, [user, githubUser, getApiClient]);

    useEffect(() => {
        const loadTree = async () => {
             if (selectedRepo && user && githubUser) {
                setIsLoading('tree');
                setError('');
                setActiveFile(null);
                try {
                    const octokit = await getApiClient();
                    const tree = await getRepoTree(octokit, selectedRepo.owner, selectedRepo.repo);
                    dispatch({ type: 'LOAD_PROJECT_FILES', payload: tree });
                } catch (err) {
                     setError(err instanceof Error ? err.message : 'Failed to load repository tree');
                } finally {
                    setIsLoading(null);
```

```
            }
        }
    };
    loadTree();
}, [selectedRepo, user, githubUser, dispatch, getApiClient]);

const handleFileSelect = async (path: string, name: string) => {
    if (!selectedRepo) return;
    setIsLoading('file');
    try {
        const octokit = await getApiClient();
        const content = await getFileContent(octokit, selectedRepo.owner,
        selectedRepo.repo, path);
        setActiveFile({ path, name, originalContent: content, editedContent: content });
    } catch (err) {
        setError((err as Error).message);
    } finally {
        setIsLoading(null);
    }
};

const handleCommit = async () => {
    if (!activeFile || !selectedRepo || activeFile.originalContent ===
    activeFile.editedContent) return;

    setIsLoading('commit');
    setError('');
    try {
        const diff = Diff.createPatch(activeFile.path, activeFile.originalContent,
        activeFile.editedContent);

        const stream = generateCommitMessageStream(diff);
        let commitMessage = '';
        for await (const chunk of stream) { commitMessage += chunk; }

        const finalMessage = window.prompt("Confirm or edit commit message:",
        commitMessage);
        if (!finalMessage) {
            setIsLoading(null);
            return;
        }

        const octokit = await getApiClient();
        await commitFiles(
            octokit,
            selectedRepo.owner,
            selectedRepo.repo,
            [{ path: activeFile.path, content: activeFile.editedContent }],
            finalMessage
        );

        addNotification(`Successfully committed to ${selectedRepo.repo}`, 'success');
        setActiveFile(prev => prev ? { ...prev, originalContent: prev.editedContent } :
        null);

    } catch (err) {
        const message = err instanceof Error ? err.message : 'Failed to commit changes';
        setError(message);
        addNotification(message, 'error');
    } finally {
        setIsLoading(null);
    }
```

```
};

if (!user) {
    return (
        <div className="h-full flex flex-col items-center justify-center text-center
        text-text-secondary p-4">
            <FolderIcon />
            <h2 className="text-lg font-semibold mt-2">Please Sign In</h2>
            <p>Sign in via the "Connections" tab to explore your repositories.</p>
        </div>
    );
}

if (!githubUser) {
     return (
        <div className="h-full flex flex-col items-center justify-center text-center
        text-text-secondary p-4">
            <FolderIcon />
            <h2 className="text-lg font-semibold mt-2">Connect to GitHub</h2>
            <p>Please go to the "Connections" tab and provide a Personal Access Token to
            explore your repositories.</p>
        </div>
    );
}

const hasChanges = activeFile ? activeFile.originalContent !==
activeFile.editedContent : false;

return (
    <div className="h-full flex flex-col text-text-primary">
        <header className="p-4 border-b border-border flex-shrink-0">
            <h1 className="text-xl font-bold flex items-center"><FolderIcon /><span
            className="ml-3">Project Explorer</span></h1>
            <div className="mt-2">
                <select
                    value={selectedRepo ? `${selectedRepo.owner}/${selectedRepo.repo}` : ''}
                    onChange={e => {
                        const [owner, repo] = e.target.value.split('/');
                        dispatch({ type: 'SET_SELECTED_REPO', payload: { owner, repo } });
                    }}
                    className="w-full p-2 bg-surface border border-border rounded-md text-sm"
                >
                    <option value="" disabled>{isLoading === 'repos' ? 'Loading...' : 'Select a
                    repository'}</option>
                    {repos.map(r => <option key={r.id} value={r.full_name}>{r.full_name}</option>)}
                </select>
            </div>
            {error && <p className="text-red-500 text-xs mt-2">{error}</p>}
        </header>
        <div className="flex-grow flex min-h-0">
            <aside className="w-1/3 bg-background border-r border-border p-4 overflow-y-
            auto">
                {isLoading === 'tree' && <div className="flex justify-center"><LoadingSpinner
                /></div>}
                {projectFiles && <FileTree node={projectFiles} onFileSelect={handleFileSelect}
                activePath={activeFile?.path ?? null} />}
            </aside>
            <main className="flex-1 bg-surface flex flex-col">
                 <div className="flex justify-between items-center p-2 border-b border-border bg-
                 gray-50 dark:bg-slate-800">
                    <span className="text-sm font-semibold">{activeFile?.name || 'No file
                    selected'}</span>
```

```
                        <button onClick={handleCommit} disabled={!hasChanges || isLoading === 'commit'}
                        className="btn-primary px-4 py-1 text-sm flex items-center justify-center
                        min-w-[100px]">
                           {isLoading === 'commit' ? <LoadingSpinner/> : 'Commit'}
                        </button>
                     </div>
                     {isLoading === 'file' ? <div className="flex items-center justify-center
                     h-full"><LoadingSpinner /></div> :
                        <textarea
                            value={activeFile?.editedContent ?? 'Select a file to view its content.'}
                            onChange={e => setActiveFile(prev => prev ? { ...prev, editedContent:
                            e.target.value } : null)}
                            disabled={!activeFile}
                            className="w-full h-full p-4 text-sm font-mono bg-transparent resize-none
                            focus:outline-none"
                        />
                     }
                  </main>
               </div>
            </div>
        );
};

// ===== AiCommitGenerator_28.tsx =====

import React, { useState, useCallback, useEffect } from 'react';
import { generateCommitMessageStream } from '../../services/index.ts';
import { downloadFile } from '../../services/fileUtils.ts';
import { GitBranchIcon, ArrowDownTrayIcon } from '../icons.tsx';
import { LoadingSpinner } from '../shared/index.tsx';

const exampleDiff = `diff --git a/src/components/Button.tsx
b/src/components/Button.tsx
index 1b2c3d4..5e6f7g8 100644
--- a/src/components/Button.tsx
+++ b/src/components/Button.tsx
@@ -1,7 +1,7 @@
 import React from 'react';

 interface ButtonProps {
-  text: string;
+  label: string;
   onClick: () => void;
 }
`;

export const AiCommitGenerator: React.FC<{ diff?: string }> = ({ diff:
initialDiff }) => {
    const [diff, setDiff] = useState<string>(initialDiff || exampleDiff);
    const [message, setMessage] = useState<string>('');
    const [isLoading, setIsLoading] = useState<boolean>(false);
    const [error, setError] = useState<string>('');

    const handleGenerate = useCallback(async (diffToAnalyze: string) => {
        if (!diffToAnalyze.trim()) {
            setError('Please paste a diff to generate a message.');
            return;
        }
        setIsLoading(true);
        setError('');
        setMessage('');
        try {
```

```
        const stream = generateCommitMessageStream(diffToAnalyze);
        let fullResponse = '';
        for await (const chunk of stream) {
            fullResponse += chunk;
            setMessage(fullResponse);
        }
    } catch (err) {
        const errorMessage = err instanceof Error ? err.message : 'An unknown error
        occurred.';
        setError(`Failed to generate message: ${errorMessage}`);
    } finally {
        setIsLoading(false);
    }
}, []);

useEffect(() => {
    if (initialDiff) {
        setDiff(initialDiff);
        handleGenerate(initialDiff);
    }
}, [initialDiff, handleGenerate]);

const handleCopy = () => {
    navigator.clipboard.writeText(message);
};

const handleDownload = () => {
    downloadFile(message, 'commit_message.txt', 'text/plain');
};

return (
    <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
        <header className="mb-6">
            <h1 className="text-3xl font-bold flex items-center">
                <GitBranchIcon />
                <span className="ml-3">AI Commit Message Generator</span>
            </h1>
            <p className="text-text-secondary mt-1">Paste your diff and let Gemini craft the
            perfect commit message.</p>
        </header>
        <div className="flex-grow flex flex-col gap-4 min-h-0">
            <div className="flex flex-col flex-1 min-h-0">
                <label htmlFor="diff-input" className="text-sm font-medium text-text-secondary
                mb-2">Git Diff</label>
                <textarea
                    id="diff-input"
                    value={diff}
                    onChange={(e) => setDiff(e.target.value)}
                    placeholder="Paste your git diff here..."
                    className="flex-grow p-4 bg-surface border border-border rounded-md resize-none
                    font-mono text-sm text-text-primary focus:ring-2 focus:ring-primary
                    focus:outline-none"
                />
            </div>
            <div className="flex-shrink-0">
                <button
                    onClick={() => handleGenerate(diff)}
                    disabled={isLoading}
                    className="btn-primary w-full max-w-xs mx-auto flex items-center justify-center
                    px-6 py-3"
                >
                    {isLoading ? <LoadingSpinner /> : 'Generate Commit Message'}
```

```
                    </button>
                </div>
                <div className="flex flex-col flex-1 min-h-0">
                    <div className="flex justify-between items-center mb-2">
                        <label className="text-sm font-medium text-text-secondary">Generated
                        Message</label>
                        {message && !isLoading && (
                            <div className="flex items-center gap-2">
                                <button onClick={handleCopy} className="px-3 py-1 bg-gray-100 text-xs rounded-md
                                hover:bg-gray-200">Copy</button>
                                <button onClick={handleDownload} className="flex items-center gap-1 px-3 py-1
                                bg-gray-100 text-xs rounded-md hover:bg-gray-200">
                                    <ArrowDownTrayIcon className="w-4 h-4" /> Download
                                </button>
                            </div>
                        )}
                    </div>
                    <div className="relative flex-grow p-4 bg-surface border border-border rounded-
                    md overflow-y-auto">
                        {isLoading && (
                             <div className="flex items-center justify-center h-full">
                                <LoadingSpinner />
                             </div>
                        )}
                        {error && <p className="text-red-500">{error}</p>}
                        {message && !isLoading && (
                           <pre className="whitespace-pre-wrap font-sans text-text-primary">{message}</pre>
                        )}
                         {!isLoading && !message && !error && (
                            <div className="text-text-secondary h-full flex items-center justify-center">
                                The commit message will appear here.
                            </div>
                        )}
                    </div>
                </div>
            </div>
        </div>
    );
};

// ===== WorkerThreadDebugger_30.tsx =====

import React, { useState, useCallback, useEffect } from 'react';
import { BugAntIcon, ArrowDownTrayIcon } from '../icons.tsx';
import { analyzeConcurrencyStream } from '../../services/index.ts';
import { LoadingSpinner, MarkdownRenderer } from '../shared/index.tsx';
import { downloadFile } from '../../services/fileUtils.ts';

const exampleCode = `// main.js
const worker = new Worker('worker.js');

// This object is sent back and forth.
// A race condition can occur because both threads
// read the counter, increment it, and send it back.
// The final value depends on which thread's message
// is processed last.
const data = { counter: 0 };

worker.onmessage = function(e) {
  // Main thread reads and updates
  data.counter = e.data.counter;
  console.log('Main received:', data.counter);
```

```
  data.counter++;
  worker.postMessage(data);
};

// Start the process
console.log('Main starting with:', data.counter);
data.counter++;
worker.postMessage(data);


// worker.js
// onmessage = function(e) {
//   // Worker reads and updates
//   let receivedCounter = e.data.counter;
//   console.log('Worker received:', receivedCounter);
//   receivedCounter++;
//   postMessage({ counter: receivedCounter });
// }
`;

export const WorkerThreadDebugger: React.FC<{ codeInput?: string }> = ({
codeInput: initialCode }) => {
    const [codeInput, setCodeInput] = useState(initialCode || exampleCode);
    const [analysis, setAnalysis] = useState('');
    const [isLoading, setIsLoading] = useState(false);
    const [error, setError] = useState('');

    const handleAnalyze = useCallback(async (codeToAnalyze: string) => {
        if (!codeToAnalyze.trim()) {
            setError('Please paste some code to analyze.');
            return;
        }
        setIsLoading(true);
        setError('');
        setAnalysis('');
        try {
            const stream = analyzeConcurrencyStream(codeToAnalyze);
            let fullResponse = '';
            for await (const chunk of stream) {
                fullResponse += chunk;
                setAnalysis(fullResponse);
            }
        } catch (err) {
            const errorMessage = err instanceof Error ? err.message : 'An unknown error
            occurred.';
            setError(`Failed to analyze code: ${errorMessage}`);
        } finally {
            setIsLoading(false);
        }
    }, []);

    useEffect(() => {
        if (initialCode) {
            setCodeInput(initialCode);
            handleAnalyze(initialCode);
        }
    }, [initialCode, handleAnalyze]);

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
            <header className="mb-6">
                <h1 className="text-3xl font-bold flex items-center">
```

```
                    <BugAntIcon />
                    <span className="ml-3">AI Concurrency Analyzer</span>
                </h1>
                <p className="text-text-secondary mt-1">Analyze JavaScript code for potential
                Web Worker concurrency issues.</p>
            </header>
            <div className="flex-grow flex flex-col gap-4 min-h-0">
                <div className="flex flex-col flex-1 min-h-0">
                    <label htmlFor="code-input" className="text-sm font-medium text-text-secondary
                    mb-2">JavaScript Code</label>
                    <textarea
                        id="code-input"
                        value={codeInput}
                        onChange={(e) => setCodeInput(e.target.value)}
                        placeholder="Paste your worker-related JS code here..."
                        className="flex-grow p-4 bg-surface border border-border rounded-md resize-none
                        font-mono text-sm"
                    />
                </div>
                 <div className="flex-shrink-0">
                    <button
                        onClick={() => handleAnalyze(codeInput)}
                        disabled={isLoading}
                        className="btn-primary w-full max-w-xs mx-auto flex items-center justify-center
                        px-6 py-3"
                    >
                        {isLoading ? <LoadingSpinner /> : 'Analyze Code'}
                    </button>
                </div>
                <div className="flex flex-col flex-1 min-h-0">
                    <div className="flex justify-between items-center mb-2">
                        <label className="text-sm font-medium text-text-secondary">AI Analysis</label>
                        {analysis && !isLoading && (
                             <button onClick={() => downloadFile(analysis, 'analysis.md', 'text/markdown')}
                             className="flex items-center gap-1 px-3 py-1 bg-gray-100 text-xs rounded-md
                             hover:bg-gray-200">
                                <ArrowDownTrayIcon className="w-4 h-4"/> Download
                            </button>
                        )}
                    </div>
                    <div className="flex-grow p-4 bg-background border border-border rounded-md
                    overflow-y-auto">
                        {isLoading && <div className="flex items-center justify-center
                        h-full"><LoadingSpinner /></div>}
                        {error && <p className="text-red-500">{error}</p>}
                        {analysis && !isLoading && <MarkdownRenderer content={analysis} />}
                        {!isLoading && !analysis && !error && <div className="text-text-secondary h-full
                        flex items-center justify-center">Analysis will appear here.</div>}
                    </div>
                </div>
            </div>
        </div>
    );
};

// ===== ScreenshotToComponent_30.tsx =====

import React, { useState, useCallback, useRef } from 'react';
import { generateComponentFromImageStream } from '../../services/index.ts';
import { PhotoIcon, ArrowDownTrayIcon } from '../icons.tsx';
import { LoadingSpinner, MarkdownRenderer } from '../shared/index.tsx';
import { fileToBase64, blobToDataURL, downloadFile } from
```

```
'../../services/fileUtils.ts';

export const ScreenshotToComponent: React.FC = () => {
    const [previewImage, setPreviewImage] = useState<string | null>(null);
    const [rawCode, setRawCode] = useState('');
    const [isLoading, setIsLoading] = useState(false);
    const [error, setError] = useState('');
    const fileInputRef = useRef<HTMLInputElement>(null);

    const handleGenerate = async (base64Image: string) => {
        setIsLoading(true);
        setError('');
        setRawCode('');
        try {
            const stream = generateComponentFromImageStream(base64Image);
            let fullResponse = '';
            for await (const chunk of stream) {
                fullResponse += chunk;
                setRawCode(fullResponse.replace(/^```(?:\w+\n)?/, '').replace(/```$/, ''));
            }
        } catch (err) {
            setError(err instanceof Error ? err.message : 'An unknown error occurred.');
        } finally {
            setIsLoading(false);
        }
    };

    const processImageBlob = async (blob: Blob) => {
        try {
            const [dataUrl, base64Image] = await Promise.all([blobToDataURL(blob),
            fileToBase64(blob as File)]);
            setPreviewImage(dataUrl);
            handleGenerate(base64Image);
        } catch (e) {
            setError('Could not process the image.');
        }
    };

    const handlePaste = useCallback(async (event: React.ClipboardEvent) => {
        const items = event.clipboardData.items;
        for (const item of items) {
            if (item.type.indexOf('image') !== -1) {
                const blob = item.getAsFile();
                if (blob) await processImageBlob(blob);
                return;
            }
        }
    }, []);

    const handleFileChange = async (event: React.ChangeEvent<HTMLInputElement>) => {
        const file = event.target.files?.[0];
        if (file) await processImageBlob(file);
    };

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
            <header className="mb-6"><h1 className="text-3xl font-bold flex items-
            center"><PhotoIcon /><span className="ml-3">AI Screenshot-to-
            Component</span></h1><p className="text-text-secondary mt-1">Paste or upload a
            screenshot of a UI element to generate React/Tailwind code.</p></header>
            <div className="flex-grow grid grid-cols-1 lg:grid-cols-2 gap-6 min-h-0">
                <div onPaste={handlePaste} className="flex flex-col items-center justify-center
```

```
                bg-surface p-6 rounded-lg border-2 border-dashed border-border focus:outline-
                none focus:border-primary overflow-y-auto" tabIndex={0}>
                    {previewImage ? (<img src={previewImage} alt="Pasted content" className="max-w-
                    full max-h-full object-contain rounded-md shadow-lg" />) : (<div
                    className="text-center text-text-secondary">
                            <h2 className="text-xl font-bold text-text-primary">Paste an image here</h2>
                            <p className="mb-2">(Cmd/Ctrl + V)</p>
                            <p className="text-sm">or</p>
                            <button onClick={() => fileInputRef.current?.click()} className="mt-2 btn-
                            primary px-4 py-2 text-sm">Upload File</button>
                            <input type="file" ref={fileInputRef} onChange={handleFileChange}
                            accept="image/*" className="hidden"/>
                        </div>)}
                </div>
                <div className="flex flex-col h-full">
                    <div className="flex justify-between items-center mb-2">
                        <label className="text-sm font-medium text-text-secondary">Generated
                        Code</label>
                        {rawCode && !isLoading && (
                            <div className="flex items-center gap-2">
                                <button onClick={() => navigator.clipboard.writeText(rawCode)} className="px-3
                                py-1 bg-gray-100 text-xs rounded-md hover:bg-gray-200">Copy Code</button>
                                <button onClick={() => downloadFile(rawCode, 'Component.tsx',
                                'text/typescript')} className="flex items-center gap-1 px-3 py-1 bg-gray-100
                                text-xs rounded-md hover:bg-gray-200">
                                    <ArrowDownTrayIcon className="w-4 h-4" /> Download
                                </button>
                            </div>
                        )}
                    </div>
                    <div className="flex-grow bg-background border border-border rounded-md
                    overflow-y-auto">
                        {isLoading && (<div className="flex items-center justify-center
                        h-full"><LoadingSpinner /></div>)}
                        {error && <p className="p-4 text-red-500">{error}</p>}
                        {rawCode && !isLoading && <MarkdownRenderer
                        content={`\`\`\`tsx\n${rawCode}\n\`\`\`} />}
                        {!isLoading && !rawCode && !error && (<div className="text-text-secondary h-full
                        flex items-center justify-center">Generated component code will appear
                        here.</div>)}
                    </div>
                </div>
            </div>
        </div>
    );
};

// ===== Connections_28.tsx =====

import React, { useState, useEffect } from 'react';
import { GithubIcon, MapIcon } from '../icons.tsx';
import { useGlobalState } from '../../contexts/GlobalStateContext.tsx';
import { signInWithGoogle } from '../../services/firebaseService.ts';
import { validateToken } from '../../services/authService.ts';
import * as vaultService from '../../services/vaultService.ts';
import { useVaultModal } from '../../contexts/VaultModalContext.tsx';
import { LoadingSpinner } from '../shared/LoadingSpinner.tsx';
import type { GitHubUser } from '../../types.ts';
import { useNotification } from '../../contexts/NotificationContext.tsx';

const GitHubConnection: React.FC = () => {
    const { state, dispatch } = useGlobalState();
```

```
const { user, githubUser } = state;
const { addNotification } = useNotification();
const { requestUnlock } = useVaultModal();
const [tokenInput, setTokenInput] = useState('');
const [isLoading, setIsLoading] = useState(false);
const [error, setError] = useState('');

useEffect(() => {
    const checkStoredToken = async () => {
        if (user && !githubUser && state.vaultState.isUnlocked) {
            setIsLoading(true);
            const token = await vaultService.getDecryptedCredential('github_pat');
            if (token) {
                try {
                    const githubProfile = await validateToken(token);
                    dispatch({ type: 'SET_GITHUB_USER', payload: githubProfile });
                } catch (e) {
                    addNotification('Failed to validate stored GitHub token.', 'error');
                }
            }
            setIsLoading(false);
        }
    };
    checkStoredToken();
}, [user, githubUser, dispatch, addNotification, state.vaultState.isUnlocked]);

const handleConnectGitHub = async () => {
    if (!user || !tokenInput.trim()) {
        setError('Token is required.');
        return;
    }
    if (!state.vaultState.isUnlocked) {
        const unlocked = await requestUnlock();
        if (!unlocked) return;
    }

    setIsLoading(true);
    setError('');
    try {
        const githubProfile = await validateToken(tokenInput);
        await vaultService.saveCredential('github_pat', tokenInput);
        dispatch({ type: 'SET_GITHUB_USER', payload: githubProfile });
        addNotification('GitHub connected successfully!', 'success');
        setTokenInput('');
    } catch (err) {
        setError(err instanceof Error ? `Invalid Token: ${err.message}` : 'Could not
        connect to GitHub.');
    } finally {
        setIsLoading(false);
    }
};

const handleDisconnectGitHub = async () => {
    if (!user) return;
    setIsLoading(true);
    try {
        await vaultService.saveCredential('github_pat', '');
        dispatch({ type: 'SET_GITHUB_USER', payload: null });
        addNotification('GitHub disconnected.', 'info');
    } catch (e) {
        addNotification('Failed to disconnect GitHub.', 'error');
    } finally {
```

```
            setIsLoading(false);
        }
    };

    if (!user) {
        return null;
    }

    return (
        <div className="bg-surface border border-border rounded-lg p-6">
            <div className="flex items-center justify-between">
                <div className="flex items-center gap-4">
                    <div className="w-10 h-10"><GithubIcon /></div>
                    <div>
                        <h3 className="text-lg font-bold text-text-primary">GitHub</h3>
                        {githubUser ? (
                            <p className="text-sm text-green-600">Connected as {githubUser.login}</p>
                        ) : (
                            <p className="text-sm text-text-secondary">Not Connected</p>
                        )}
                    </div>
                </div>
                 {githubUser && (
                    <button onClick={handleDisconnectGitHub} className="px-4 py-2 bg-red-500/10
                    text-red-600 font-semibold rounded-lg hover:bg-red-500/20">
                        Disconnect
                    </button>
                 )}
            </div>

            {!githubUser && (
                <div className="mt-4 pt-4 border-t border-border">
                    <label htmlFor="github-pat" className="block text-sm font-medium text-text-
                    secondary mb-1">Personal Access Token (Classic)</label>
                    <div className="flex gap-2">
                        <input id="github-pat" type="password" value={tokenInput} onChange={(e) =>
                        setTokenInput(e.target.value)} placeholder="ghp_..." className="flex-grow p-2
                        bg-background border border-border rounded-md text-sm" />
                         <button onClick={handleConnectGitHub} disabled={isLoading} className="btn-
                         primary px-6 py-2 flex items-center justify-center min-w-[100px]">
                            {isLoading ? <LoadingSpinner /> : 'Connect'}
                        </button>
                    </div>
                    {error && <p className="text-red-500 text-xs mt-2">{error}</p>}
                </div>
            )}
        </div>
    );
};

const GoogleMapsConnection: React.FC = () => {
    const { state } = useGlobalState();
    const { user } = state;
    const { addNotification } = useNotification();
    const { requestUnlock } = useVaultModal();
    const [apiKey, setApiKey] = useState('');
    const [isSaved, setIsSaved] = useState(false);
    const [isLoading, setIsLoading] = useState(false);

    useEffect(() => {
        const checkKey = async () => {
            if(user && state.vaultState.isUnlocked) {
```

```
                const key = await vaultService.getDecryptedCredential('google_maps_api_key');
                setIsSaved(!!key);
            }
        };
        checkKey();
    }, [user, state.vaultState.isUnlocked]);

    const handleSaveKey = async () => {
        if (!apiKey.trim()) return;
        if (!state.vaultState.isUnlocked) {
            const unlocked = await requestUnlock();
            if (!unlocked) return;
        }
        setIsLoading(true);
        try {
            await vaultService.saveCredential('google_maps_api_key', apiKey);
            addNotification('Google Maps API Key saved!', 'success');
            setIsSaved(true);
            setApiKey('');
        } catch(e) {
            addNotification('Failed to save key.', 'error');
        } finally {
            setIsLoading(false);
        }
    }

    if (!user) return null;

    return (
        <div className="bg-surface border border-border rounded-lg p-6">
            <div className="flex items-center gap-4">
                <div className="w-10 h-10"><MapIcon /></div>
                <div>
                    <h3 className="text-lg font-bold text-text-primary">Google Maps Platform</h3>
                    <p className={`text-sm ${isSaved ? 'text-green-600' : 'text-text-
                    secondary'}`}>{isSaved ? 'API Key is saved in vault' : 'Not Connected'}</p>
                </div>
            </div>
             <div className="mt-4 pt-4 border-t border-border">
                <label htmlFor="maps-key" className="block text-sm font-medium text-text-
                secondary mb-1">API Key</label>
                <div className="flex gap-2">
                    <input id="maps-key" type="password" value={apiKey} onChange={(e) =>
                    setApiKey(e.target.value)} placeholder="AIzaSy..." className="flex-grow p-2 bg-
                    background border border-border rounded-md text-sm" />
                     <button onClick={handleSaveKey} disabled={isLoading} className="btn-primary px-6
                     py-2 flex items-center justify-center min-w-[100px]">
                        {isLoading ? <LoadingSpinner /> : 'Save'}
                    </button>
                </div>
            </div>
        </div>
    )
}

export const Connections: React.FC = () => {
    const { state } = useGlobalState();
    const { user } = state;
    const { addNotification } = useNotification();
    const [isLoading, setIsLoading] = useState(false);

    const handleSignIn = async () => {
```

```
        setIsLoading(true);
        try {
            await signInWithGoogle();
            addNotification('Signed in successfully!', 'success');
        } catch (error) {
            addNotification('Failed to sign in.', 'error');
            console.error(error);
        }
        setIsLoading(false);
    };

    return (
        <div className="h-full p-4 sm:p-6 lg:p-8">
            <div className="max-w-4xl mx-auto">
                <header className="mb-8">
                    <h1 className="text-4xl font-extrabold tracking-tight">Connections</h1>
                    <p className="mt-2 text-lg text-text-secondary">Sign in and connect to external
                    services.</p>
                </header>

                {!user ? (
                    <div className="text-center bg-surface p-8 rounded-lg border border-border">
                        <h2 className="text-xl font-bold">Sign In Required</h2>
                        <p className="text-text-secondary my-4">Please sign in with your Google account
                        to manage connections and save your data.</p>
                        <button onClick={handleSignIn} disabled={isLoading} className="btn-primary px-6
                        py-3 flex items-center justify-center gap-2 mx-auto">
                           {isLoading ? <LoadingSpinner/> : 'Sign in with Google'}
                        </button>
                    </div>
                ) : (
                    <div className="space-y-6">
                        <GitHubConnection />
                        <GoogleMapsConnection />
                    </div>
                )}
            </div>
        </div>
    );
};


// ===== MarkdownSlides_30.tsx =====


import React, { useState, useMemo, useEffect, useRef, useCallback } from
'react';
import { marked } from 'marked';
import { PhotoIcon } from '../icons.tsx';
import { useNotification } from '../../contexts/NotificationContext.tsx';
import { useGlobalState } from '../../contexts/GlobalStateContext.tsx';
import { summarizeForSlides } from '../../services/index.ts';
import { createPresentation, addSlide } from
'../../services/workspaceService.ts';
import { LoadingSpinner } from '../shared/index.tsx';

const exampleMarkdown = `# Slide 1: Welcome

This is a slide deck generated from Markdown.

- Use standard markdown syntax
- Like lists, headers, and **bold** text.
```

```
---

# Slide 2: Features

Navigate using the buttons below.

\`\`\`javascript
console.log("Code blocks work too!");
\`\`\`

---

# Slide 3: The End

Easy to create and present.
`;

export const MarkdownSlides: React.FC = () => {
    const [markdown, setMarkdown] = useState(exampleMarkdown);
    const [currentSlide, setCurrentSlide] = useState(0);
    const [slideHtml, setSlideHtml] = useState<string | TrustedHTML>('');
    const [isExporting, setIsExporting] = useState(false);
    const presentationRef = useRef<HTMLDivElement>(null);
    const { addNotification } = useNotification();
    const { state } = useGlobalState();
    const { user } = state;

    const slides = useMemo(() => markdown.split(/^-{3,}\s*$/m), [markdown]);

    useEffect(() => {
        const parse = async () => {
            const currentSlideContent = slides[currentSlide] || '';
            const html = await marked.parse(currentSlideContent);
            setSlideHtml(html);
        };
        parse();
    }, [slides, currentSlide]);

    const goToNext = useCallback(() => setCurrentSlide(s => Math.min(s + 1,
    slides.length - 1)), [slides.length]);
    const goToPrev = useCallback(() => setCurrentSlide(s => Math.max(s - 1, 0)),
    []);

    const handleFullscreen = () => {
        presentationRef.current?.requestFullscreen();
    };

    const handleExportToSlides = async () => {
        if (!user) {
            addNotification('Please sign in with Google to export to Slides.', 'error');
            return;
        }
        setIsExporting(true);
        try {
            addNotification('Creating presentation...', 'info');
            const presentation = await createPresentation(`Presentation from DevCore: ${new
            Date().toLocaleDateString()}`);
            addNotification(`Presentation created: ${presentation.presentationId}`,
            'success');

            for (let i = 0; i < slides.length; i++) {
                addNotification(`Processing slide ${i + 1}/${slides.length}...`, 'info');
```

```
            const slideContent = slides[i];
            const summary = await summarizeForSlides(slideContent);
            await addSlide(presentation.presentationId, summary);
        }
        addNotification('All slides added successfully!', 'success');
        window.open(presentation.webViewLink, '_blank');

    } catch (e) {
        console.error(e);
        addNotification(e instanceof Error ? e.message : 'Failed to export
        presentation.', 'error');
    } finally {
        setIsExporting(false);
    }
};

useEffect(() => {
    const handleKeyDown = (e: KeyboardEvent) => {
        if (document.fullscreenElement === presentationRef.current) {
            if (e.key === 'ArrowRight' || e.key === ' ') goToNext();
            if (e.key === 'ArrowLeft') goToPrev();
        }
    };
    document.addEventListener('keydown', handleKeyDown);
    return () => document.removeEventListener('keydown', handleKeyDown);
}, [goToNext, goToPrev]);

return (
    <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
        <header className="mb-6">
            <h1 className="text-3xl font-bold flex items-center"><PhotoIcon /><span
            className="ml-3">Markdown to Slides</span></h1>
            <p className="text-text-secondary mt-1">Write markdown, present it as a
            slideshow. Use '---' to separate slides.</p>
        </header>
        <div className="flex-grow grid grid-cols-1 lg:grid-cols-2 gap-6 h-full overflow-
        hidden">
            <div className="flex flex-col h-full">
                 <label htmlFor="md-input" className="text-sm font-medium text-text-secondary
                 mb-2">Markdown Editor</label>
                 <textarea id="md-input" value={markdown} onChange={e =>
                 setMarkdown(e.target.value)} className="flex-grow p-4 bg-surface border border-
                 border rounded-md resize-none font-mono text-sm focus:ring-2 focus:ring-primary
                 focus:outline-none"/>
            </div>
             <div ref={presentationRef} className="flex flex-col h-full bg-surface
             fullscreen:bg-background border border-border rounded-md">
                <div className="flex-shrink-0 flex justify-end items-center p-2 border-b border-
                border gap-2">
                    <button onClick={handleExportToSlides} disabled={isExporting || !user}
                    className="btn-primary text-xs px-3 py-1 flex items-center gap-1 disabled:bg-
                    gray-400">
                       {isExporting ? <LoadingSpinner/> : 'Export to Google Slides'}
                    </button>
                    <button onClick={handleFullscreen} className="px-3 py-1 bg-gray-100 dark:bg-
                    slate-700 rounded-md text-xs hover:bg-gray-200 dark:hover:bg-
                    slate-600">Fullscreen</button>
                </div>
                <div className="relative flex-grow flex flex-col justify-center items-center p-8
                overflow-y-auto">
                    <div className="prose prose-lg max-w-none w-full" dangerouslySetInnerHTML={{
                    __html: slideHtml }} />
```

```
                        <button onClick={goToPrev} disabled={currentSlide === 0} className="absolute
                        left-4 top-1/2 -translate-y-1/2 p-2 bg-gray-200/50 dark:bg-slate-700/50 rounded-
                        full disabled:opacity-30 hover:bg-gray-300/50 dark:hover:bg-
                        slate-600/50">■</button>
                        <button onClick={goToNext} disabled={currentSlide === slides.length - 1}
                        className="absolute right-4 top-1/2 -translate-y-1/2 p-2 bg-gray-200/50 dark:bg-
                        slate-700/50 rounded-full disabled:opacity-30 hover:bg-gray-300/50
                        dark:hover:bg-slate-600/50">■</button>
                        <div className="absolute bottom-4 right-4 text-xs bg-black/50 px-2 py-1 rounded-
                        md text-white">
                           {currentSlide + 1} / {slides.length}
                        </div>
                    </div>
                </div>
            </div>
        </div>
    );
};

// ===== PerformanceProfiler_18.tsx =====

import React, { useState, useCallback } from 'react';
import { analyzePerformanceTrace } from '../../services/index.ts';
import { startTracing, stopTracing, TraceEntry } from
'../../services/profiling/performanceService.ts';
import { parseViteStats, BundleStatsNode } from
'../../services/profiling/bundleAnalyzer.ts';
import { ChartBarIcon, SparklesIcon } from '../icons.tsx';
import { LoadingSpinner, MarkdownRenderer } from '../shared/index.tsx';

const FlameChart: React.FC<{ trace: TraceEntry[] }> = ({ trace }) => {
    if (trace.length === 0) return <p className="text-text-secondary">No trace data
    collected.</p>;
    const maxTime = Math.max(...trace.map(t => t.startTime + t.duration));
    return (
        <div className="space-y-1 font-mono text-xs">
            {trace.filter(t => t.entryType === 'measure').map((entry, i) => (
                <div key={i} className="group relative h-6 bg-primary/20 rounded">
                    <div className="h-full bg-primary" style={{ marginLeft: `${(entry.startTime /
                    maxTime) * 100}%`, width: `${(entry.duration / maxTime) * 100}%` }}></div>
                    <div className="absolute inset-0 px-2 flex items-center text-primary font-
                    bold">{entry.name} ({entry.duration.toFixed(1)}ms)</div>
                </div>
            ))}
        </div>
    );
};

export const PerformanceProfiler: React.FC = () => {
    const [activeTab, setActiveTab] = useState<'runtime' | 'bundle'>('runtime');
    const [isTracing, setIsTracing] = useState(false);
    const [trace, setTrace] = useState<TraceEntry[]>([]);
    const [bundleStats, setBundleStats] = useState<string>('');
    const [bundleTree, setBundleTree] = useState<BundleStatsNode | null>(null);
    const [isLoadingAi, setIsLoadingAi] = useState(false);
    const [aiAnalysis, setAiAnalysis] = useState('');

    const handleTraceToggle = () => {
        if (isTracing) {
            const collectedTrace = stopTracing();
            setTrace(collectedTrace);
            setIsTracing(false);
```

```
    } else {
        setTrace([]);
        startTracing();
        setIsTracing(true);
    }
};

const handleAnalyzeBundle = () => {
    try {
        setBundleTree(parseViteStats(bundleStats));
    } catch (e) {
        alert(e instanceof Error ? e.message : 'Parsing failed.');
    }
};

const handleAiAnalysis = async () => {
    const dataToAnalyze = activeTab === 'runtime' ? trace : bundleTree;
    if (!dataToAnalyze || (Array.isArray(dataToAnalyze) && dataToAnalyze.length ===
    0)) {
        alert('No data to analyze.');
        return;
    }
    setIsLoadingAi(true);
    setAiAnalysis('');
    try {
        const analysis = await analyzePerformanceTrace(dataToAnalyze);
        setAiAnalysis(analysis);
    } catch (e) {
        setAiAnalysis('Error getting analysis from AI.');
    } finally {
        setIsLoadingAi(false);
    }
};

return (
    <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
        <header className="mb-6"><h1 className="text-3xl font-bold flex items-
        center"><ChartBarIcon /><span className="ml-3">AI Performance
        Profiler</span></h1><p className="text-text-secondary mt-1">Analyze runtime
        performance and bundle sizes with AI insights.</p></header>
        <div className="flex border-b border-border mb-4"><button onClick={() =>
        setActiveTab('runtime')} className={`px-4 py-2 text-sm ${activeTab === 'runtime'
        ? 'border-b-2 border-primary' : ''}`}>Runtime Performance</button><button
        onClick={() => setActiveTab('bundle')} className={`px-4 py-2 text-sm ${activeTab
        === 'bundle' ? 'border-b-2 border-primary' : ''}`}>Bundle
        Analysis</button></div>
        <div className="flex-grow grid grid-cols-1 lg:grid-cols-2 gap-6 min-h-0">
            <div className="bg-surface p-4 border border-border rounded-lg flex flex-col">
                {activeTab === 'runtime' ? (
                    <>
                        <button onClick={handleTraceToggle} className="btn-primary mb-4 py-2">{isTracing
                        ? 'Stop Tracing' : 'Start Tracing'}</button>
                        <div className="flex-grow overflow-y-auto"><FlameChart trace={trace} /></div>
                    </>
                ) : (
                     <>
                        <textarea value={bundleStats} onChange={e => setBundleStats(e.target.value)}
                        placeholder="Paste your stats.json content here" className="w-full h-48 p-2 bg-
                        background border rounded font-mono text-xs mb-2"/>
                        <button onClick={handleAnalyzeBundle} className="btn-primary py-2">Analyze
                        Bundle</button>
                        <div className="flex-grow overflow-y-auto mt-2">
```

```
                                <pre className="text-xs">{bundleTree ? JSON.stringify(bundleTree, null, 2) :
                                'Analysis will appear here.'}</pre>
                            </div>
                        </>
                    )}
                </div>
                 <div className="bg-surface p-4 border border-border rounded-lg flex flex-col">
                    <button onClick={handleAiAnalysis} disabled={isLoadingAi} className="btn-primary
                    flex items-center justify-center gap-2 py-2 mb-4"><SparklesIcon />{isLoadingAi ?
                    'Analyzing...' : 'Get AI Optimization Suggestions'}</button>
                    <div className="flex-grow bg-background border border-border rounded p-2
                    overflow-y-auto">
                        {isLoadingAi ? <div className="flex justify-center items-center
                        h-full"><LoadingSpinner/></div> : <MarkdownRenderer content={aiAnalysis} />}
                    </div>
                 </div>
            </div>
        </div>
    );
};

// ===== AccessibilityAuditor_18.tsx =====

import React, { useState, useRef } from 'react';
import { suggestA11yFix } from '../../services/index.ts';
import { runAxeAudit, AxeResult } from
'../../services/auditing/accessibilityService.ts';
import { EyeIcon, SparklesIcon } from '../icons.tsx';
import { LoadingSpinner, MarkdownRenderer } from '../shared/index.tsx';

export const AccessibilityAuditor: React.FC = () => {
    const [url, setUrl] = useState('https://react.dev');
    const [auditUrl, setAuditUrl] = useState('');
    const [results, setResults] = useState<AxeResult | null>(null);
    const [isLoading, setIsLoading] = useState(false);
    const [isLoadingAi, setIsLoadingAi] = useState<string | null>(null);
    const [aiFixes, setAiFixes] = useState<Record<string, string>>({});
    const iframeRef = useRef<HTMLIFrameElement>(null);

    const handleAudit = () => {
        const targetUrl = url.startsWith('http') ? url : `https://${url}`;
        setAuditUrl(targetUrl);
        setIsLoading(true);
        setResults(null);
        setAiFixes({});
    };

    const handleIframeLoad = async () => {
        if (isLoading && iframeRef.current) {
            try {
                const auditResults = await
                runAxeAudit(iframeRef.current.contentWindow!.document);
                setResults(auditResults);
            } catch (error) {
                console.error(error);
                alert('Could not audit this page. This may be due to security restrictions
                (CORS).');
            } finally {
                setIsLoading(false);
            }
        }
    };
```

```
    const handleGetFix = async (issue: any) => {
        const issueId = issue.id;
        setIsLoadingAi(issueId);
        try {
            const fix = await suggestA11yFix(issue);
            setAiFixes(prev => ({...prev, [issueId]: fix}));
        } catch(e) {
            setAiFixes(prev => ({...prev, [issueId]: 'Could not get suggestion.'}));
        } finally {
            setIsLoadingAi(null);
        }
    };

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
            <header className="mb-6"><h1 className="text-3xl font-bold flex items-
            center"><EyeIcon /><span className="ml-3">Automated Accessibility
            Auditor</span></h1><p className="text-text-secondary mt-1">Audit a live URL for
            accessibility issues and get AI-powered fixes.</p></header>
            <div className="flex gap-2 mb-4"><input type="text" value={url} onChange={e =>
            setUrl(e.target.value)} placeholder="https://example.com" className="flex-grow
            p-2 border rounded"/><button onClick={handleAudit} disabled={isLoading}
            className="btn-primary px-6 py-2">{isLoading ? 'Auditing...' :
            'Audit'}</button></div>
            <div className="flex-grow grid grid-cols-1 lg:grid-cols-2 gap-6 min-h-0">
                <div className="bg-background border-2 border-dashed border-border rounded-lg
                overflow-hidden"><iframe ref={iframeRef} src={auditUrl} title="Audit Target"
                className="w-full h-full bg-white" onLoad={handleIframeLoad} sandbox="allow-
                scripts allow-same-origin"/></div>
                <div className="bg-surface p-4 border border-border rounded-lg flex flex-col">
                    <h3 className="text-lg font-bold mb-2">Audit Results</h3>
                    <div className="flex-grow overflow-y-auto pr-2">
                        {isLoading && <div className="flex justify-center items-center
                        h-full"><LoadingSpinner/></div>}
                        {results && (results.violations.length === 0 ? <p>No violations found!</p> :
                            results.violations.map((v, i) => (
                                <div key={v.id + i} className="p-3 mb-2 bg-background border border-border
                                rounded">
                                    <p className="font-bold text-red-600">{v.help}</p>
                                    <p className="text-sm my-1">{v.description}</p>
                                    <button onClick={() => handleGetFix(v)} disabled={!!isLoadingAi}
                                    className="text-xs flex items-center gap-1 text-primary font-
                                    semibold"><SparklesIcon/> {isLoadingAi === v.id ? 'Getting fix...' : 'Ask AI for
                                    a fix'}</button>
                                    {aiFixes[v.id] && <div className="mt-2 text-xs border-t pt-2"><MarkdownRenderer
                                    content={aiFixes[v.id]}/></div>}
                                </div>
                            ))
                        )}
                    </div>
                </div>
            </div>
        </div>
    );
};

// ===== CiCdPipelineGenerator_18.tsx =====

import React, { useState } from 'react';
import { generateCiCdConfig } from '../../services/index.ts';
import { PaperAirplaneIcon, SparklesIcon } from '../icons.tsx';
import { LoadingSpinner, MarkdownRenderer } from '../shared/index.tsx';
```

```
const platforms = ['GitHub Actions', 'GitLab CI', 'CircleCI', 'Jenkins'];
const exampleDescription = "Install Node.js dependencies, run linting and tests,
build the production app, and then deploy to Vercel.";

export const CiCdPipelineGenerator: React.FC = () => {
    const [platform, setPlatform] = useState(platforms[0]);
    const [description, setDescription] = useState(exampleDescription);
    const [config, setConfig] = useState('');
    const [isLoading, setIsLoading] = useState(false);
    const [error, setError] = useState('');

    const handleGenerate = async () => {
        if (!description.trim()) {
            setError('Please provide a description of the pipeline stages.');
            return;
        }
        setIsLoading(true);
        setError('');
        try {
            const result = await generateCiCdConfig(platform, description);
            setConfig(result);
        } catch (err) {
            setError(err instanceof Error ? err.message : 'Failed to generate config.');
        } finally {
            setIsLoading(false);
        }
    };

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
            <header className="mb-6">
                <h1 className="text-3xl font-bold flex items-center"><PaperAirplaneIcon /><span
                className="ml-3">AI CI/CD Pipeline Architect</span></h1>
                <p className="text-text-secondary mt-1">Describe your deployment process and get
                a modern configuration file.</p>
            </header>
            <div className="flex-grow flex flex-col gap-4 min-h-0">
                 <div className="flex flex-col flex-1 min-h-0">
                    <div className="grid grid-cols-1 md:grid-cols-3 gap-4 mb-4">
                        <div><label className="block text-sm">Platform</label><select value={platform}
                        onChange={e => setPlatform(e.target.value)} className="w-full mt-1 p-2 bg-
                        surface border rounded"><option>GitHub Actions</option><option>GitLab
                        CI</option><option>CircleCI</option></select></div>
                        <div className="md:col-span-2"><label className="block text-sm">Describe
                        Stages</label><input type="text" value={description} onChange={e =>
                        setDescription(e.target.value)} className="w-full mt-1 p-2 bg-surface border
                        rounded"/></div>
                    </div>
                     <button onClick={handleGenerate} disabled={isLoading} className="btn-primary
                     w-full max-w-xs mx-auto flex items-center justify-center py-2"><SparklesIcon />
                     {isLoading ? 'Generating...' : 'Generate Configuration'}</button>
                </div>
                 <div className="flex flex-col flex-grow min-h-0">
                    <label className="text-sm font-medium text-text-secondary mb-2">Generated
                    Configuration File</label>
                    <div className="relative flex-grow p-1 bg-background border border-border
                    rounded-md overflow-y-auto">
                        {isLoading && !config && <div className="flex items-center justify-center
                        h-full"><LoadingSpinner /></div>}
                        {error && <p className="p-4 text-red-500">{error}</p>}
                        {config && <MarkdownRenderer content={config} />}
                         {!isLoading && !config && !error && <div className="text-text-secondary h-full
```

```
                        flex items-center justify-center">Generated config will appear here.</div>}
                    </div>
                </div>
            </div>
        </div>
    );
};

// ===== TerraformGenerator_15.tsx =====

import React, { useState, useCallback } from 'react';
import { generateTerraformConfig } from '../../services/index.ts';
import { CpuChipIcon, SparklesIcon } from '../icons.tsx';
import { LoadingSpinner, MarkdownRenderer } from '../shared/index.tsx';

export const TerraformGenerator: React.FC = () => {
    const [description, setDescription] = useState('An S3 bucket for static website
    hosting');
    const [cloud, setCloud] = useState<'aws' | 'gcp'>('aws');
    const [config, setConfig] = useState('');
    const [isLoading, setIsLoading] = useState(false);
    const [error, setError] = useState('');

    const handleGenerate = useCallback(async () => {
        if (!description.trim()) {
            setError('Please provide a description.');
            return;
        }
        setIsLoading(true);
        setError('');
        setConfig('');
        try {
            // Context is stubbed for now but demonstrates future capability
            const context = 'User might have existing VPCs. Check before creating new
            ones.';
            const result = await generateTerraformConfig(cloud, description, context);
            setConfig(result);
        } catch (err) {
            setError(err instanceof Error ? err.message : 'Failed to generate config.');
        } finally {
            setIsLoading(false);
        }
    }, [description, cloud]);

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
            <header className="mb-6">
                <h1 className="text-3xl font-bold flex items-center"><CpuChipIcon /><span
                className="ml-3">AI Terraform Generator</span></h1>
                <p className="text-text-secondary mt-1">Generate infrastructure-as-code from a
                description, with context from your cloud provider.</p>
            </header>
            <div className="flex-grow flex flex-col gap-4 min-h-0">
                 <div className="flex flex-col flex-1 min-h-0">
                     <div className="grid grid-cols-1 md:grid-cols-3 gap-4 mb-4">
                        <div>
                            <label className="block text-sm">Cloud Provider</label>
                            <select value={cloud} onChange={e => setCloud(e.target.value as 'aws' | 'gcp')}
                            className="w-full mt-1 p-2 bg-surface border rounded">
                                <option value="aws">AWS</option>
                                <option value="gcp">GCP</option>
                            </select>
```

```
                        </div>
                        <div className="md:col-span-2">
                            <label className="block text-sm">Describe the infrastructure</label>
                            <input type="text" value={description} onChange={e =>
                            setDescription(e.target.value)} className="w-full mt-1 p-2 bg-surface border
                            rounded"/>
                        </div>
                    </div>
                     <button onClick={handleGenerate} disabled={isLoading} className="btn-primary
                     w-full max-w-xs mx-auto flex items-center justify-center py-2"><SparklesIcon />
                     {isLoading ? 'Generating...' : 'Generate Configuration'}</button>
                </div>
                 <div className="flex flex-col flex-grow min-h-0">
                    <label className="text-sm font-medium text-text-secondary mb-2">Generated
                    Terraform (.tf)</label>
                    <div className="relative flex-grow p-1 bg-background border border-border
                    rounded-md overflow-y-auto">
                        {isLoading && !config && <div className="flex items-center justify-center
                        h-full"><LoadingSpinner /></div>}
                        {error && <p className="p-4 text-red-500">{error}</p>}
                        {config && <MarkdownRenderer content={config} />}
                         {!isLoading && !config && !error && <div className="text-text-secondary h-full
                         flex items-center justify-center">Generated config will appear here.</div>}
                    </div>
                </div>
            </div>
        </div>
    );
};

// ===== AiPersonalityForge_15.tsx =====

import React, { useState, useEffect, useRef } from 'react';
import { SparklesIcon, PlusIcon, TrashIcon, ArrowDownTrayIcon,
ArrowUpOnSquareIcon } from '../icons.tsx';
import { useAiPersonalities } from '../../hooks/useAiPersonalities.ts';
import { formatSystemPromptToString } from '../../utils/promptUtils.ts';
import { streamContent } from '../../services/index.ts';
import { downloadJson } from '../../services/fileUtils.ts';
import type { SystemPrompt } from '../../types.ts';
import { LoadingSpinner, MarkdownRenderer } from '../shared/index.tsx';
import { useNotification } from '../../contexts/NotificationContext.tsx';

const defaultNewPrompt: Omit<SystemPrompt, 'id' | 'name'> = {
    persona: 'You are a helpful assistant.',
    rules: [],
    outputFormat: 'markdown',
    exampleIO: [],
};

export const AiPersonalityForge: React.FC = () => {
    const [personalities, setPersonalities] = useAiPersonalities();
    const [activeId, setActiveId] = useState<string | null>(null);
    const { addNotification } = useNotification();
    const fileInputRef = useRef<HTMLInputElement>(null);

    // Testbed State
    const [testbedInput, setTestbedInput] = useState('');
    const [chatHistory, setChatHistory] = useState<{ role: 'user' | 'model';
    content: string }[]>([]);
    const [isStreaming, setIsStreaming] = useState(false);
    const activePersonality = personalities.find(p => p.id === activeId);
```

```
useEffect(() => {
    if (!activeId && personalities.length > 0) {
        setActiveId(personalities[0].id);
    }
}, [personalities, activeId]);

const handleUpdate = (field: keyof SystemPrompt, value: any) => {
    if (!activePersonality) return;
    const updated = { ...activePersonality, [field]: value };
    setPersonalities(personalities.map(p => (p.id === activeId ? updated : p)));
};

const handleAddNew = () => {
    const newId = Date.now().toString();
    const newPersonality: SystemPrompt = { ...defaultNewPrompt, id: newId, name:
    'Untitled Personality' };
    setPersonalities([...personalities, newPersonality]);
    setActiveId(newId);
};

const handleDelete = (id: string) => {
    if (window.confirm('Are you sure you want to delete this personality?')) {
        setPersonalities(personalities.filter(p => p.id !== id));
        if (activeId === id) {
            setActiveId(personalities.length > 1 ? personalities[0].id : null);
        }
    }
};

const handleTestbedSend = async () => {
    if (!testbedInput.trim() || !activePersonality || isStreaming) return;

    const systemInstruction = formatSystemPromptToString(activePersonality);
    const newHistory = [...chatHistory, { role: 'user' as const, content:
    testbedInput }];
    setChatHistory(newHistory);
    setTestbedInput('');
    setIsStreaming(true);

    try {
        const stream = streamContent(testbedInput, systemInstruction, 0.7);
        let fullResponse = '';
        setChatHistory(prev => [...prev, { role: 'model', content: '' }]);
        for await (const chunk of stream) {
            fullResponse += chunk;
            setChatHistory(prev => {
                const last = prev[prev.length - 1];
                if (last.role === 'model') {
                    return [...prev.slice(0, -1), { role: 'model', content: fullResponse }];
                }
                return prev;
            });
        }
    } catch (e) {
        const errorMsg = e instanceof Error ? e.message : 'An error occurred';
        setChatHistory(prev => [...prev, { role: 'model', content: `**Error:**
        ${errorMsg}` }]);
    } finally {
        setIsStreaming(false);
    }
};
const handleExport = () => {
```

```
    if (!activePersonality) return;
    downloadJson(activePersonality, `${activePersonality.name.replace(/\s+/g,
    '_')}.json`);
    addNotification('Personality exported!', 'success');
};

const handleImport = (e: React.ChangeEvent<HTMLInputElement>) => {
    const file = e.target.files?.[0];
    if (!file) return;
    const reader = new FileReader();
    reader.onload = (event) => {
        try {
            const imported = JSON.parse(event.target?.result as string) as SystemPrompt;
            // Basic validation
            if (imported.id && imported.name && imported.persona) {
                setPersonalities(prev => [...prev.filter(p => p.id !== imported.id), imported]);
                setActiveId(imported.id);
                addNotification('Personality imported!', 'success');
            } else {
                 addNotification('Invalid personality file.', 'error');
            }
        } catch {
             addNotification('Failed to parse JSON file.', 'error');
        }
    };
    reader.readAsText(file);
};

return (
    <div className="h-full flex text-text-primary">
        {/* Sidebar */}
        <aside className="w-64 bg-surface border-r border-border flex flex-col">
            <div className="p-4 border-b border-border">
                <h2 className="text-lg font-bold">Personalities</h2>
            </div>
            <div className="flex-grow overflow-y-auto">
                {personalities.map(p => (
                    <div key={p.id} onClick={() => setActiveId(p.id)} className={`group flex
                    justify-between items-center p-3 text-sm cursor-pointer ${activeId === p.id ?
                    'bg-primary/10 text-primary' : 'hover:bg-gray-100 dark:hover:bg-slate-700'}`}>
                        <span className="truncate">{p.name}</span>
                        <button onClick={(e) => { e.stopPropagation(); handleDelete(p.id)}}
                        className="opacity-0 group-hover:opacity-100 text-text-secondary hover:text-
                        red-500"><TrashIcon /></button>
                    </div>
                ))}
            </div>
            <div className="p-4 border-t border-border space-y-2">
                <button onClick={handleAddNew} className="btn-primary w-full py-2 text-sm flex
                items-center justify-center gap-2"><PlusIcon /> New</button>
                <div className="flex gap-2">
                     <button onClick={() => fileInputRef.current?.click()} className="flex-1 py-2
                     text-sm bg-gray-100 dark:bg-slate-700 rounded-md flex items-center justify-
                     center gap-2"><ArrowUpOnSquareIcon/> Import</button>
                     <button onClick={handleExport} className="flex-1 py-2 text-sm bg-gray-100
                     dark:bg-slate-700 rounded-md flex items-center justify-center
                     gap-2"><ArrowDownTrayIcon/> Export</button>
                     <input type="file" ref={fileInputRef} onChange={handleImport} accept=".json"
                     className="hidden"/>
                </div>
            </div>
        </aside>
```

```
{/* Main Content */}
{activePersonality ? (
     <div className="flex-1 grid grid-cols-2 gap-px bg-border">
        {/* Editor */}
        <div className="bg-background p-4 flex flex-col gap-4 overflow-y-auto">
            <div><label className="font-bold">Name</label><input type="text"
            value={activePersonality.name} onChange={e => handleUpdate('name',
            e.target.value)} className="w-full mt-1 p-2 bg-surface border rounded"/></div>
            <div><label className="font-bold">Persona</label><textarea
            value={activePersonality.persona} onChange={e => handleUpdate('persona',
            e.target.value)} className="w-full mt-1 p-2 bg-surface border rounded
            h-24"/></div>
            <div><label className="font-bold">Rules (one per line)</label><textarea
            value={activePersonality.rules.join('\n')} onChange={e => handleUpdate('rules',
            e.target.value.split('\n'))} className="w-full mt-1 p-2 bg-surface border
            rounded h-32"/></div>
            <div><label className="font-bold">Output Format</label><select
            value={activePersonality.outputFormat} onChange={e =>
            handleUpdate('outputFormat', e.target.value)} className="w-full mt-1 p-2 bg-
            surface border rounded"><option>markdown</option><option>json</option><option>te
            xt</option></select></div>
            <div>
                <h3 className="font-bold mb-2">Examples</h3>
                {activePersonality.exampleIO.map((ex, i) => (
                    <div key={i} className="grid grid-cols-2 gap-2 mb-2 p-2 border rounded bg-
                    surface">
                        <textarea placeholder="User Input" value={ex.input} onChange={e =>
                        handleUpdate('exampleIO', activePersonality.exampleIO.map((item, idx) => idx ===
                        i ? {...item, input: e.target.value} : item))} className="h-20 p-1 bg-background
                        border rounded"/>
                        <textarea placeholder="Model Output" value={ex.output} onChange={e =>
                        handleUpdate('exampleIO', activePersonality.exampleIO.map((item, idx) => idx ===
                        i ? {...item, output: e.target.value} : item))} className="h-20 p-1 bg-
                        background border rounded"/>
                    </div>
                ))}
                <button onClick={() => handleUpdate('exampleIO',
                [...activePersonality.exampleIO, {input: '', output: ''}])} className="text-sm
                text-primary">+ Add Example</button>
            </div>
        </div>
        {/* Testbed */}
        <div className="bg-background p-4 flex flex-col">
            <h2 className="text-lg font-bold mb-2 border-b pb-2">Live Testbed</h2>
            <div className="flex-grow overflow-y-auto space-y-4 pr-2">
               {chatHistory.map((msg, i) => (
                   <div key={i} className={`p-3 rounded-lg ${msg.role === 'user' ? 'bg-primary/10'
                   : 'bg-surface'}`}>
                        <strong className="capitalize">{msg.role}</strong>
                        <MarkdownRenderer content={msg.content} />
                   </div>
               ))}
               {isStreaming && <div className="flex justify-center"><LoadingSpinner/></div>}
            </div>
            <div className="flex gap-2 mt-4">
                <input value={testbedInput} onChange={e => setTestbedInput(e.target.value)}
                onKeyDown={e => e.key === 'Enter' && handleTestbedSend()} className="flex-grow
                p-2 bg-surface border rounded" placeholder="Test your AI..."/>
                <button onClick={handleTestbedSend} disabled={isStreaming} className="btn-
                primary px-4">Send</button>
            </div>
        </div>
```

```
                </div>
            ) : (
                <div className="flex-1 flex items-center justify-center text-text-
                secondary">Select or create a personality to begin.</div>
            )}
        </div>
    );
};

// ===== WeeklyDigestGenerator_9.tsx =====

import React, { useState, useCallback } from 'react';
import { generateWeeklyDigest } from '../../services/index.ts';
import { sendEmail } from '../../services/workspaceService.ts';
import { useGlobalState } from '../../contexts/GlobalStateContext.tsx';
import { useNotification } from '../../contexts/NotificationContext.tsx';
import { MailIcon, SparklesIcon } from '../icons.tsx';
import { LoadingSpinner } from '../shared/index.tsx';

// Dummy data for demonstration purposes
const dummyCommitLogs = `
feat: implement user authentication
fix: resolve issue with button alignment
feat: add dark mode toggle
chore: update dependencies
refactor: simplify data fetching logic
`;
const dummyTelemetry = {
    avgPageLoad: 120,
    errorRate: '0.5%',
    uptime: '99.98%'
};

export const WeeklyDigestGenerator: React.FC = () => {
    const { state } = useGlobalState();
    const { user } = state;
    const { addNotification } = useNotification();
    const [emailHtml, setEmailHtml] = useState('');
    const [isLoading, setIsLoading] = useState(false);
    const [isSending, setIsSending] = useState(false);

    const handleGenerate = useCallback(async () => {
        setIsLoading(true);
        setEmailHtml('');
        try {
            const html = await generateWeeklyDigest(dummyCommitLogs, dummyTelemetry);
            setEmailHtml(html);
            addNotification('Digest content generated!', 'success');
        } catch (e) {
            addNotification(e instanceof Error ? e.message : 'Failed to generate digest',
            'error');
        } finally {
            setIsLoading(false);
        }
    }, []);

    const handleSend = async () => {
        if (!emailHtml || !user?.email) {
            addNotification('Cannot send email. Generate content and ensure you are signed
            in.', 'error');
            return;
        }
```

```
        setIsSending(true);
        try {
            await sendEmail(user.email, 'Your Weekly Project Digest', emailHtml);
            addNotification(`Weekly digest sent to ${user.email}!`, 'success');
        } catch (e) {
             addNotification(e instanceof Error ? e.message : 'Failed to send email',
             'error');
        } finally {
            setIsSending(false);
        }
    };

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
            <header className="mb-6">
                <h1 className="text-3xl font-bold flex items-center"><MailIcon /><span
                className="ml-3">Weekly Digest Generator</span></h1>
                <p className="text-text-secondary mt-1">Generate an AI-powered weekly summary
                and send it via your Gmail.</p>
            </header>

            <div className="flex-grow grid grid-cols-1 lg:grid-cols-2 gap-6 min-h-0">
                <div className="bg-surface p-4 border border-border rounded-lg flex flex-col
                items-center justify-center text-center">
                    <h3 className="text-lg font-bold">Generate & Send</h3>
                    <p className="text-sm text-text-secondary my-4">This tool will use dummy data to
                    generate a project summary and send it to your signed-in email address
                    ({user?.email || 'N/A'}).</p>
                    <div className="flex flex-col gap-4 w-full max-w-xs">
                        <button onClick={handleGenerate} disabled={isLoading || isSending}
                        className="btn-primary flex items-center justify-center gap-2 py-3">
                            {isLoading ? <LoadingSpinner /> : <><SparklesIcon /> Generate Digest</>}
                        </button>
                        <button onClick={handleSend} disabled={!emailHtml || isSending || isLoading ||
                        !user} className="btn-primary flex items-center justify-center gap-2 py-3 bg-
                        emerald-600">
                           {isSending ? <LoadingSpinner /> : 'Send Email via Gmail'}
                        </button>
                    </div>
                </div>

                <div className="bg-surface p-4 border border-border rounded-lg flex flex-col">
                    <h3 className="text-lg font-bold mb-2">Email Preview</h3>
                    <div className="flex-grow bg-white border rounded overflow-hidden">
                        {isLoading && <div className="flex justify-center items-center
                        h-full"><LoadingSpinner /></div>}
                        {emailHtml && <iframe srcDoc={emailHtml} title="Email Preview" className="w-full
                        h-full" />}
                        {!isLoading && !emailHtml && <div className="flex justify-center items-center
                        h-full text-text-secondary">Preview will appear here.</div>}
                    </div>
                </div>
            </div>
        </div>
    );
};

// ===== AiCommandCenter_31.tsx =====

import React, { useState, useCallback } from 'react';
import { Type, FunctionDeclaration } from "@google/genai";
import { logError } from '../../services/telemetryService.ts';
```

```
import { getInferenceFunction, CommandResponse } from
'../../services/aiService.ts';
import { FEATURE_TAXONOMY } from '../../services/taxonomyService.ts';
import { useGlobalState } from '../../contexts/GlobalStateContext.tsx';
import { CommandLineIcon } from '../icons.tsx';
import { LoadingSpinner } from '../shared/index.tsx';
import { ALL_FEATURES } from './index.ts';
import { executeWorkspaceAction, ACTION_REGISTRY } from
'../../services/workspaceConnectorService.ts';

const baseFunctionDeclarations: FunctionDeclaration[] = [
    {
        name: 'navigateTo',
        description: 'Navigates to a specific feature page.',
        parameters: {
            type: Type.OBJECT,
            properties: {
                featureId: {
                    type: Type.STRING,
                    description: 'The ID of the feature to navigate to.',
                    enum: ALL_FEATURES.map(f => f.id)
                },
            },
            required: ['featureId'],
        },
    },
    {
        name: 'runFeatureWithInput',
        description: 'Navigates to a feature and passes initial data to it.',
        parameters: {
            type: Type.OBJECT,
            properties: {
                 featureId: {
                    type: Type.STRING,
                    description: 'The ID of the feature to run.',
                    enum: ALL_FEATURES.map(f => f.id)
                },
                props: {
                    type: Type.OBJECT,
                    description: 'An object containing the initial properties for the feature, based
                    on its required inputs.',
                    properties: {
                        initialCode: { type: Type.STRING },
                        initialPrompt: { type: Type.STRING },
                        beforeCode: { type: Type.STRING },
                        afterCode: { type: Type.STRING },
                        logInput: { type: Type.STRING },
                        diff: { type: Type.STRING },
                        codeInput: { type: Type.STRING },
                        jsonInput: { type: Type.STRING },
                    }
                }
            },
            required: ['featureId', 'props']
        }
    }
];

// Dynamically add the workspace action
const functionDeclarations: FunctionDeclaration[] = [
    ...baseFunctionDeclarations,
    {
```

```
        name: 'runWorkspaceAction',
        description: 'Executes a defined action on a connected workspace service like
        Jira, Slack, or GitHub.',
        parameters: {
            type: Type.OBJECT,
            properties: {
                 actionId: {
                    type: Type.STRING,
                    description: 'The unique identifier for the action to execute.',
                    enum: [ ...ACTION_REGISTRY.keys() ]
                },
                params: {
                    type: Type.OBJECT,
                    description: 'An object containing the parameters for the action, matching its
                    required inputs.'
                }
            },
            required: ['actionId', 'params']
        }
    }
]

const knowledgeBase = FEATURE_TAXONOMY.map(f => `- ${f.name} (${f.id}):
${f.description} Inputs: ${f.inputs}`).join('\n');

const ExamplePromptButton: React.FC<{ text: string, onClick: (text: string) =>
void }> = ({ text, onClick }) => (
    <button
        onClick={() => onClick(text)}
        className="px-3 py-1.5 bg-surface border border-border rounded-full text-xs
        hover:bg-gray-100 transition-colors"
    >
        {text}
    </button>
)

export const AiCommandCenter: React.FC = () => {
    const { dispatch } = useGlobalState();
    const [prompt, setPrompt] = useState('');
    const [isLoading, setIsLoading] = useState(false);
    const [lastResponse, setLastResponse] = useState('');

    const handleCommand = useCallback(async () => {
        if (!prompt.trim()) return;

        setIsLoading(true);
        setLastResponse('');

        try {
            const response: CommandResponse = await getInferenceFunction(prompt,
            functionDeclarations, knowledgeBase);

            if (response.functionCalls && response.functionCalls.length > 0) {
                const call = response.functionCalls[0];
                const { name, args } = call;

                setLastResponse(`Understood! Executing command: ${name}`);

                switch (name) {
                    case 'navigateTo':
                        dispatch({ type: 'SET_VIEW', payload: { view: args.featureId }});
                        break;
```

```
                case 'runFeatureWithInput':
                     dispatch({ type: 'SET_VIEW', payload: { view: args.featureId, props: args.props
                     } });
                    break;
                case 'runWorkspaceAction':
                    try {
                        const result = await executeWorkspaceAction(args.actionId, args.params);
                        setLastResponse(`Action '${args.actionId}' executed successfully.\n\nResult:
                        \`\`\`json\n${JSON.stringify(result, null, 2)}\n\`\`\``);
                    } catch (e) {
                        setLastResponse(`Action failed: ${e instanceof Error ? e.message : 'Unknown
                        error'}`);
                    }
                    break;
                default:
                    setLastResponse(`Unknown command: ${name}`);
            }
             setPrompt('');
        } else {
             setLastResponse(response.text);
        }

    } catch (err) {
        logError(err as Error, { prompt });
        setLastResponse(err instanceof Error ? err.message : 'An unknown error
        occurred.');
    } finally {
        setIsLoading(false);
    }
}, [prompt, dispatch]);

const handleKeyDown = (e: React.KeyboardEvent) => {
    if (e.key === 'Enter' && !e.shiftKey) {
        e.preventDefault();
        handleCommand();
    }
};

const handleExampleClick = (text: string) => {
    setPrompt(text);
}

return (
    <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
        <header className="mb-6 text-center">
            <h1 className="text-4xl font-extrabold tracking-tight flex items-center justify-
            center">
                <CommandLineIcon />
                <span className="ml-3">AI Command Center</span>
            </h1>
            <p className="mt-2 text-lg text-text-secondary">What would you like to do?</p>
        </header>

        <div className="flex-grow flex flex-col justify-end max-w-3xl w-full mx-auto">
            {lastResponse && (
                <div className="mb-4 p-4 bg-surface rounded-lg text-text-primary border border-
                border">
                    <p><strong>AI:</strong> {lastResponse}</p>
                </div>
            )}
             <div className="relative">
                <textarea
```

```
                        value={prompt}
                        onChange={e => setPrompt(e.target.value)}
                        onKeyDown={handleKeyDown}
                        disabled={isLoading}
                        placeholder='Try "explain this code: const a = 1;" or "open the theme designer"'
                        className="w-full p-4 pr-28 rounded-lg bg-surface border border-border
                        focus:ring-2 focus:ring-primary focus:outline-none resize-none shadow-sm"
                        rows={2}
                    />
                    <button
                        onClick={handleCommand}
                        disabled={isLoading}
                        className="btn-primary absolute right-3 top-1/2 -translate-y-1/2 px-4 py-2"
                    >
                       {isLoading ? <LoadingSpinner/> : 'Send'}
                    </button>
                </div>
                 <div className="flex flex-wrap items-center justify-center gap-2 mt-4">
                    <ExamplePromptButton text="Open Theme Designer" onClick={handleExampleClick} />
                    <ExamplePromptButton text="Generate a commit for a bug fix"
                    onClick={handleExampleClick} />
                    <ExamplePromptButton text="Create a regex for email validation"
                    onClick={handleExampleClick} />
                </div>
                 <p className="text-xs text-text-secondary text-center mt-2">Press Enter to send,
                 Shift+Enter for new line.</p>
            </div>
        </div>
    );
};

// ===== WorkspaceConnectorHub.tsx =====

import React, { useState, useEffect, useMemo, useCallback } from 'react';
import { useGlobalState } from '../../contexts/GlobalStateContext.tsx';
import * as vaultService from '../../services/vaultService.ts';
import { useNotification } from '../../contexts/NotificationContext.tsx';
import { validateToken } from '../../services/authService.ts';
import { ACTION_REGISTRY, executeWorkspaceAction } from
'../../services/workspaceConnectorService.ts';
import { RectangleGroupIcon, GithubIcon, SparklesIcon } from '../icons.tsx';
import { LoadingSpinner } from '../shared/index.tsx';
import { signInWithGoogle } from '../../services/firebaseService.ts';
import { useVaultModal } from '../../contexts/VaultModalContext.tsx';

const ServiceConnectionCard: React.FC<{
    serviceName: string;
    icon: React.ReactNode;
    fields: { id: string; label: string; placeholder: string }[];
    onConnect: (credentials: Record<string, string>) => Promise<void>;
    onDisconnect: () => Promise<void>;
    status: string;
    isLoading: boolean;
}> = ({ serviceName, icon, fields, onConnect, onDisconnect, status, isLoading })
=> {
    const [creds, setCreds] = useState<Record<string, string>>({});

    const handleConnect = () => {
        onConnect(creds);
    };

    const isConnected = status.startsWith('Connected');
```

```
    return (
        <div className="bg-surface border border-border rounded-lg p-6">
            <div className="flex items-center justify-between">
                <div className="flex items-center gap-4">
                    <div className="w-10 h-10">{icon}</div>
                    <div>
                        <h3 className="text-lg font-bold text-text-primary">{serviceName}</h3>
                        <p className={`text-sm ${isConnected ? 'text-green-600' : 'text-text-
                        secondary'}`}>{status}</p>
                    </div>
                </div>
                {isConnected && (
                    <button onClick={onDisconnect} className="px-4 py-2 bg-red-500/10 text-red-600
                    font-semibold rounded-lg hover:bg-red-500/20">
                        Disconnect
                    </button>
                )}
            </div>
            {!isConnected && (
                <div className="mt-4 pt-4 border-t border-border space-y-2">
                    {fields.map(field => (
                        <div key={field.id}>
                            <label className="text-xs text-text-secondary">{field.label}</label>
                            <input
                                type={field.id.includes('token') || field.id.includes('pat') ? 'password' :
                                'text'}
                                value={creds[field.id] || ''}
                                onChange={e => setCreds(prev => ({ ...prev, [field.id]: e.target.value }))}
                                placeholder={field.placeholder}
                                className="w-full mt-1 p-2 bg-background border border-border rounded-md text-
                                sm"
                            />
                        </div>
                    ))}
                    <button onClick={handleConnect} disabled={isLoading} className="btn-primary
                    w-full mt-2 py-2 flex items-center justify-center">
                        {isLoading ? <LoadingSpinner /> : 'Connect'}
                    </button>
                </div>
            )}
        </div>
    );
};


export const WorkspaceConnectorHub: React.FC = () => {
    const { state, dispatch } = useGlobalState();
    const { user, githubUser, vaultState } = state;
    const { addNotification } = useNotification();
    const { requestUnlock, requestCreation } = useVaultModal();
    const [loadingStates, setLoadingStates] = useState<Record<string, boolean>>({});
    const [connectionStatuses, setConnectionStatuses] = useState<Record<string,
    string>>({});

    // Manual action state
    const [selectedActionId, setSelectedActionId] =
    useState<string>([...ACTION_REGISTRY.keys()][0]);
    const [actionParams, setActionParams] = useState<Record<string, any>>({});
    const [isExecuting, setIsExecuting] = useState(false);
    const [actionResult, setActionResult] = useState<string>('');

    const services = useMemo(() => {
```

```
    const serviceMap = new Map();
    ACTION_REGISTRY.forEach(action => {
        if (!serviceMap.has(action.service)) {
            serviceMap.set(action.service, {
                name: action.service,
                actions: [],
            });
        }
        serviceMap.get(action.service).actions.push(action);
    });
    return Array.from(serviceMap.values());
}, []);

const checkConnections = useCallback(async () => {
    if (!user || !vaultState.isUnlocked) return;

    const checkCred = async (credId: string, serviceName: string, successMessage:
    string) => {
         const token = await vaultService.getDecryptedCredential(credId);
         setConnectionStatuses(s => ({ ...s, [serviceName]: token ? successMessage : 'Not
         Connected' }));
    };

    await checkCred('github_pat', 'GitHub', githubUser ? `Connected as
    ${githubUser.login}`: 'Connected');
    await checkCred('jira_pat', 'Jira', 'Connected');
    await checkCred('slack_bot_token', 'Slack', 'Connected');

}, [user, vaultState.isUnlocked, githubUser]);

useEffect(() => {
    checkConnections();
}, [checkConnections]);

const withVault = useCallback(async (callback: () => Promise<void>) => {
    if (!vaultState.isInitialized) {
        const created = await requestCreation();
        if (!created) { addNotification('Vault setup is required.', 'error'); return; }
    }
    if (!vaultState.isUnlocked) {
        const unlocked = await requestUnlock();
        if (!unlocked) { addNotification('Vault must be unlocked to manage
        connections.', 'error'); return; }
    }
    await callback();
}, [vaultState, requestCreation, requestUnlock, addNotification]);


const handleConnect = async (serviceName: string, credentials: Record<string,
string>) => {
    await withVault(async () => {
        setLoadingStates(s => ({ ...s, [serviceName]: true }));
        try {
            for (const [key, value] of Object.entries(credentials)) {
                if (value) await vaultService.saveCredential(key, value);
            }
            if (serviceName === 'GitHub' && credentials.github_pat) {
                 const githubProfile = await validateToken(credentials.github_pat);
                 dispatch({ type: 'SET_GITHUB_USER', payload: githubProfile });
                 await vaultService.saveCredential('github_user', JSON.stringify(githubProfile));
            }
            addNotification(`${serviceName} connected successfully!`, 'success');
```

```
            checkConnections();
        } catch (e) {
            addNotification(`Failed to connect ${serviceName}: ${e instanceof Error ?
            e.message : 'Unknown error'}`, 'error');
        } finally {
            setLoadingStates(s => ({ ...s, [serviceName]: false }));
        }
    });
};

const handleDisconnect = async (serviceName: string, credIds: string[]) => {
   await withVault(async () => {
        setLoadingStates(s => ({ ...s, [serviceName]: true }));
        try {
            for (const id of credIds) {
                await vaultService.saveCredential(id, ''); // Overwrite with empty string
            }
             if (serviceName === 'GitHub') {
                dispatch({ type: 'SET_GITHUB_USER', payload: null });
                await vaultService.saveCredential('github_user', '');
            }
            addNotification(`${serviceName} disconnected.`, 'info');
            checkConnections();
        } catch(e) {
            addNotification(`Failed to disconnect ${serviceName}.`, 'error');
        } finally {
             setLoadingStates(s => ({ ...s, [serviceName]: false }));
        }
   });
};

const handleExecuteAction = async () => {
    await withVault(async () => {
        setIsExecuting(true);
        setActionResult('');
        try {
            const result = await executeWorkspaceAction(selectedActionId, actionParams);
            setActionResult(JSON.stringify(result, null, 2));
            addNotification('Action executed successfully!', 'success');
        } catch(e) {
            setActionResult(`Error: ${e instanceof Error ? e.message : 'Unknown Error'}`);
            addNotification('Action failed.', 'error');
        } finally {
            setIsExecuting(false);
        }
    });
};

const handleSignIn = async () => {
    setLoadingStates(s => ({...s, google: true}));
    try {
        await signInWithGoogle();
        addNotification('Signed in successfully!', 'success');
    } catch (error) {
        addNotification('Failed to sign in.', 'error');
    }
    setLoadingStates(s => ({...s, google: false}));
};

const selectedAction = ACTION_REGISTRY.get(selectedActionId);
const actionParameters = selectedAction ? selectedAction.getParameters() : {};
if (!user) {
```

```
    return (
        <div className="h-full flex items-center justify-center">
            <div className="text-center bg-surface p-8 rounded-lg border border-border max-
            w-md">
                <h2 className="text-xl font-bold">Sign In Required</h2>
                <p className="text-text-secondary my-4">Please sign in with your Google account
                to manage workspace connections.</p>
                <button onClick={handleSignIn} disabled={loadingStates.google} className="btn-
                primary px-6 py-3 flex items-center justify-center gap-2 mx-auto">
                    {loadingStates.google ? <LoadingSpinner/> : 'Sign in with Google'}
                </button>
            </div>
        </div>
    );
}

return (
    <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
         <header className="mb-8">
            <h1 className="text-4xl font-extrabold tracking-tight flex items-
            center"><RectangleGroupIcon /><span className="ml-3">Workspace Connector
            Hub</span></h1>
            <p className="mt-2 text-lg text-text-secondary">Connect to your development
            services to unlock cross-platform AI actions.</p>
        </header>
        <div className="flex-grow grid grid-cols-1 lg:grid-cols-2 gap-8 min-h-0">
            <div className="flex flex-col gap-6 overflow-y-auto pr-4">
                <h2 className="text-2xl font-bold">Service Connections</h2>
                <ServiceConnectionCard
                    serviceName="GitHub"
                    icon={<GithubIcon />}
                    fields={[{ id: 'github_pat', label: 'Personal Access Token', placeholder:
                    'ghp_...' }]}
                    onConnect={(creds) => handleConnect('GitHub', creds)}
                    onDisconnect={() => handleDisconnect('GitHub', ['github_pat'])}
                    status={connectionStatuses.GitHub || 'Checking...'}
                    isLoading={loadingStates.GitHub}
                />
                 {/* Placeholder cards for Jira and Slack */}
                <ServiceConnectionCard
                    serviceName="Jira"
                    icon={<div className="w-10 h-10 bg-[#0052CC] rounded flex items-center justify-
                    center text-white font-bold text-xl">J</div>}
                    fields={[
                        { id: 'jira_domain', label: 'Jira Domain', placeholder: 'your-
                        company.atlassian.net' },
                        { id: 'jira_email', label: 'Your Jira Email', placeholder: 'you@example.com' },
                        { id: 'jira_pat', label: 'API Token', placeholder: 'Your API Token' },
                    ]}
                    onConnect={(creds) => handleConnect('Jira', creds)}
                    onDisconnect={() => handleDisconnect('Jira', ['jira_domain', 'jira_email',
                    'jira_pat'])}
                    status={connectionStatuses.Jira || 'Checking...'}
                    isLoading={loadingStates.Jira}
                />
                <ServiceConnectionCard
                    serviceName="Slack"
                    icon={<div className="w-10 h-10 bg-[#4A154B] rounded flex items-center justify-
                    center text-white font-bold text-2xl">#</div>}
                    fields={[{ id: 'slack_bot_token', label: 'Bot User OAuth Token', placeholder:
                    'xoxb-...' }]}
                    onConnect={(creds) => handleConnect('Slack', creds)}
```

```
                        onDisconnect={() => handleDisconnect('Slack', ['slack_bot_token'])}
                        status={connectionStatuses.Slack || 'Checking...'}
                        isLoading={loadingStates.Slack}
                    />
                </div>
                <div className="flex flex-col gap-6 bg-surface p-6 border border-border rounded-
                lg">
                    <h2 className="text-2xl font-bold">Manual Action Runner</h2>
                    <div className="space-y-4">
                         <div>
                            <label className="text-sm font-medium">Action</label>
                            <select value={selectedActionId} onChange={e =>
                            setSelectedActionId(e.target.value)} className="w-full mt-1 p-2 bg-background
                            border rounded">
                                {services.map(service => (
                                    <optgroup label={service.name} key={service.name}>
                                        {service.actions.map((action: any) => (
                                            <option key={action.id} value={action.id}>{action.description}</option>
                                        ))}
                                    </optgroup>
                                ))}
                            </select>
                        </div>
                        {Object.entries(actionParameters).map(([key, param]: [string, any]) => (
                            <div key={key}>
                                <label className="text-sm font-medium">{key} {param.required && '*'}</label>
                                <input
                                    type={param.type}
                                    value={actionParams[key] || ''}
                                    onChange={e => setActionParams(p => ({...p, [key]: e.target.value}))}
                                    placeholder={param.default || ''}
                                    className="w-full mt-1 p-2 bg-background border rounded"
                                />
                            </div>
                        ))}
                        <button onClick={handleExecuteAction} disabled={isExecuting} className="btn-
                        primary w-full py-2 flex items-center justify-center gap-2">
                           {isExecuting ? <LoadingSpinner/> : <><SparklesIcon /> Execute Action</>}
                        </button>
                    </div>
                    <div>
                        <label className="text-sm font-medium">Result</label>
                        <pre className="w-full h-48 mt-1 p-2 bg-background border rounded overflow-auto
                        text-xs">{actionResult || 'Action results will appear here.'}</pre>
                    </div>
                </div>
            </div>
        </div>
    );
};

// ===== ThemeDesigner_36.tsx =====

import React, { useState, useCallback } from 'react';
import { SparklesIcon, ArrowDownTrayIcon, PhotoIcon } from '../icons.tsx';
import { generateSemanticTheme } from '../../services/index.ts';
import { fileToBase64 } from '../../services/fileUtils.ts';
import type { SemanticColorTheme, ColorTheme } from '../../types.ts';
import { LoadingSpinner } from '../shared/index.tsx';
import { useTheme } from '../../hooks/useTheme.ts';

const ColorDisplay: React.FC<{ name: string; color: { name: string; value:
```

```
string; } }> = ({ name, color }) => (
    <div className="flex items-center justify-between p-2 bg-background rounded-md
    border border-border">
        <div className="flex items-center gap-3">
            <div className="w-6 h-6 rounded-full border border-border" style={{
            backgroundColor: color.value }} />
            <div>
                <p className="text-sm font-semibold text-text-primary capitalize">{name}</p>
                <p className="text-xs text-text-secondary">{color.name}</p>
            </div>
        </div>
        <span className="font-mono text-sm text-text-secondary">{color.value}</span>
    </div>
);

const AccessibilityCheck: React.FC<{ name: string, check: { ratio: number;
score: string; } }> = ({ name, check }) => {
    const scoreColor = check.score === 'AAA' ? 'text-green-600' : check.score ===
    'AA' ? 'text-emerald-600' : 'text-red-600';
    return (
        <div className="flex items-center justify-between p-2 bg-background rounded-md
        border border-border text-sm">
            <p className="text-text-secondary">{name}</p>
            <div className="flex items-center gap-2">
                <span className="font-mono">{check.ratio.toFixed(2)}</span>
                <span className={`font-bold px-2 py-0.5 rounded-full text-xs ${scoreColor}
                ${scoreColor.replace('text-', 'bg-')}/10`}>{check.score}</span>
            </div>
        </div>
    );
}

export const ThemeDesigner: React.FC = () => {
    const [theme, setTheme] = useState<SemanticColorTheme | null>(null);
    const [prompt, setPrompt] = useState('');
    const [image, setImage] = useState<{ base64: string, name: string } |
    null>(null);
    const [isLoading, setIsLoading] = useState(false);
    const [error, setError] = useState('');
    const [, , applyCustomTheme] = useTheme();

    const handleGenerate = useCallback(async () => {
        if (!prompt.trim() && !image) {
            setError("Please provide a description or an image.");
            return;
        }
        const textPart = { text: `Generate a theme based on this description:
        "${prompt}"` };
        const imagePart = image ? { inlineData: { mimeType: 'image/png', data:
        image.base64 } } : null;
        const parts = imagePart ? [textPart, imagePart] : [textPart];

        setIsLoading(true); setError('');
        try {
            const newTheme = await generateSemanticTheme({ parts });
            setTheme(newTheme);
        } catch (err) {
            setError(err instanceof Error ? err.message : "An unknown error occurred.");
        } finally {
            setIsLoading(false);
        }
    }, [prompt, image]);
```

```
const handleFileChange = async (e: React.ChangeEvent<HTMLInputElement>) => {
    const file = e.target.files?.[0];
    if (file) {
        const base64 = await fileToBase64(file);
        setImage({ base64, name: file.name });
        setPrompt(`A theme based on the uploaded image: ${file.name}`);
    }
};

const handleApplyTheme = () => {
    if (!theme) return;
    const colorsToApply: ColorTheme = {
        primary: theme.palette.primary.value,
        background: theme.theme.background.value,
        surface: theme.theme.surface.value,
        textPrimary: theme.theme.textPrimary.value,
        textSecondary: theme.theme.textSecondary.value,
        textOnPrimary: theme.theme.textOnPrimary.value,
        border: theme.theme.border.value,
    };
    applyCustomTheme(colorsToApply, theme.mode);
};

return (
    <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
        <header className="mb-6">
            <h1 className="text-3xl font-bold flex items-center"><SparklesIcon /><span
            className="ml-3">AI Theme Designer</span></h1>
            <p className="text-text-secondary mt-1">Generate a full design system from a
            description or image.</p>
        </header>
        <div className="flex-grow grid grid-cols-1 md:grid-cols-2 gap-6 min-h-0">
            <div className="md:col-span-1 flex flex-col gap-4 bg-surface border border-
            border p-6 rounded-lg overflow-y-auto">
                <h3 className="text-xl font-bold">Describe or Upload</h3>
                <textarea value={prompt} onChange={e => setPrompt(e.target.value)}
                className="p-2 bg-background border border-border rounded-md resize-none text-sm
                h-24" placeholder="e.g., A light, airy theme for a blog" />
                 <div className="relative border border-dashed border-border rounded-lg p-4 text-
                 center">
                    <input type="file" onChange={handleFileChange} className="absolute inset-0
                    w-full h-full opacity-0 cursor-pointer" />
                    <PhotoIcon/>
                    <p className="text-sm mt-1">{image ? `Image: ${image.name}` : 'Upload an image
                    (optional)'}</p>
                </div>
                <div className="flex gap-2">
                    <button onClick={handleGenerate} disabled={isLoading} className="btn-primary
                    flex-grow flex items-center justify-center gap-2 px-4 py-2">
                        {isLoading ? <LoadingSpinner /> : 'Generate New Theme'}
                    </button>
                     <button onClick={handleApplyTheme} disabled={isLoading || !theme}
                     className="px-4 py-2 bg-emerald-600 text-white font-bold rounded-md
                     hover:opacity-90 transition-all disabled:opacity-50 shadow-md">
                        Apply to App
                    </button>
                </div>
                {error && <p className="text-red-500 text-xs text-center">{error}</p>}

                {theme && !isLoading && (
                    <div className="mt-4 border-t border-border pt-4 space-y-4">
                        <div><h3 className="text-lg font-bold mb-2">Palette</h3><div
```

```
                        className="space-y-2"><ColorDisplay name="Primary"
                        color={theme.palette.primary}/><ColorDisplay name="Secondary"
                        color={theme.palette.secondary}/><ColorDisplay name="Accent"
                        color={theme.palette.accent}/><ColorDisplay name="Neutral"
                        color={theme.palette.neutral}/></div></div>
                        <div><h3 className="text-lg font-bold mb-2">Theme Roles</h3><div
                        className="space-y-2"><ColorDisplay name="Background"
                        color={theme.theme.background}/><ColorDisplay name="Surface"
                        color={theme.theme.surface}/><ColorDisplay name="Text Primary"
                        color={theme.theme.textPrimary}/><ColorDisplay name="Text Secondary"
                        color={theme.theme.textSecondary}/><ColorDisplay name="Text on Primary"
                        color={theme.theme.textOnPrimary}/><ColorDisplay name="Border"
                        color={theme.theme.border}/></div></div>
                        <div><h3 className="text-lg font-bold mb-2">Accessibility (WCAG 2.1)</h3><div
                        className="space-y-2"><AccessibilityCheck name="Primary on Surface"
                        check={theme.accessibility.primaryOnSurface}/><AccessibilityCheck name="Text on
                        Surface" check={theme.accessibility.textPrimaryOnSurface}/><AccessibilityCheck
                        name="Subtle Text on Surface"
                        check={theme.accessibility.textSecondaryOnSurface}/><AccessibilityCheck
                        name="Text on Primary"
                        check={theme.accessibility.textOnPrimaryOnPrimary}/></div></div>
                    </div>
                )}
            </div>
            <div className="md:col-span-1 rounded-lg p-8 overflow-y-auto border border-
            border" style={{ backgroundColor: theme?.theme.background.value, color:
            theme?.theme.textPrimary.value }}>
                <h3 className="text-2xl font-bold mb-6">Live Preview</h3>
                {theme ? (
                    <div className="p-6 rounded-lg grid grid-cols-1 md:grid-cols-2 gap-6" style={{
                    backgroundColor: theme.theme.surface.value }}>
                        <div className="space-y-4">
                            <h4 className="text-lg font-bold">Sample Card</h4>
                            <p className="text-sm" style={{color: theme.theme.textSecondary.value}}>This is
                            a sample card to demonstrate the theme colors. It contains a primary button and
                            some secondary text.</p>
                            <button className="px-4 py-2 rounded-md font-bold transition-colors" style={{
                            backgroundColor: theme.palette.primary.value, color:
                            theme.theme.textOnPrimary.value }}>Primary Button</button>
                        </div>
                         <div className="space-y-4">
                            <input type="text" placeholder="Text input" className="w-full px-3 py-2 rounded-
                            md border" style={{backgroundColor: theme.theme.background.value, borderColor:
                            theme.theme.border.value, color: theme.theme.textPrimary.value}} />
                            <div className="p-3 border rounded" style={{borderColor:
                            theme.theme.border.value, color: theme.theme.textSecondary.value}}>
                                <p>A bordered container.</p>
                            </div>
                         </div>
                    </div>
                ) : <div className="flex items-center justify-center h-full text-text-
                secondary">Theme preview will appear here.</div>}
            </div>
        </div>
    </div>
  );
};

// ===== AiCodeExplainer_34.tsx =====

import React, { useState, useCallback, useEffect, useMemo, useRef } from
'react';
```

```
import mermaid from 'mermaid';
import { explainCodeStructured, generateMermaidJs } from
'../../services/index.ts';
import type { StructuredExplanation } from '../../types.ts';
import { CpuChipIcon } from '../icons.tsx';
import { MarkdownRenderer, LoadingSpinner } from '../shared/index.tsx';

type ExplanationTab = 'summary' | 'lineByLine' | 'complexity' | 'suggestions' |
'flowchart';

const simpleSyntaxHighlight = (code: string) => {
    const escapedCode = code
        .replace(/&/g, '&amp;')
        .replace(/</g, '&lt;')
        .replace(/>/g, '&gt;');

    return escapedCode
        .replace(/\b(const|let|var|function|return|if|for|=>|import|from|export|default)
        \b/g, '<span class="text-indigo-400 font-semibold">$1</span>')
        .replace(/(\`|'|")(.*?)(\`|'|")/g, '<span class="text-
        emerald-400">$1$2$3</span>')
        .replace(/(\/\/.*)/g, '<span class="text-gray-400 italic">$1</span>')
        .replace(/(\{|\}|\(|\)|\[|\])/g, '<span class="text-gray-400">$1</span>');
};

mermaid.initialize({ startOnLoad: false, theme: 'neutral', securityLevel:
'loose' });

export const AiCodeExplainer: React.FC<{ initialCode?: string }> = ({
initialCode }) => {
    const [code, setCode] = useState<string>(initialCode || '');
    const [explanation, setExplanation] = useState<StructuredExplanation |
    null>(null);
    const [mermaidCode, setMermaidCode] = useState<string>('');
    const [isLoading, setIsLoading] = useState<boolean>(false);
    const [error, setError] = useState<string>('');
    const [activeTab, setActiveTab] = useState<ExplanationTab>('summary');
    const textareaRef = useRef<HTMLTextAreaElement>(null);
    const preRef = useRef<HTMLPreElement>(null);
    const mermaidContainerRef = useRef<HTMLDivElement>(null);

    const handleExplain = useCallback(async (codeToExplain: string) => {
        if (!codeToExplain.trim()) {
            setError('Please enter some code to explain.');
            return;
        }
        setIsLoading(true);
        setError('');
        setExplanation(null);
        setMermaidCode('');
        setActiveTab('summary');
        try {
            const [explanationResult, mermaidResult] = await Promise.all([
                explainCodeStructured(codeToExplain),
                generateMermaidJs(codeToExplain)
            ]);
            setExplanation(explanationResult);
            setMermaidCode(mermaidResult.replace(/```mermaid\n|```/g, ''));

        } catch (err) {
            const errorMessage = err instanceof Error ? err.message : 'An unknown error
            occurred.';
```

```
        setError(`Failed to get explanation: ${errorMessage}`);
    } finally {
        setIsLoading(false);
    }
}, []);

useEffect(() => {
    if (initialCode) {
        setCode(initialCode);
        handleExplain(initialCode);
    }
}, [initialCode, handleExplain]);

useEffect(() => {
    const renderMermaid = async () => {
         if (activeTab === 'flowchart' && mermaidCode && mermaidContainerRef.current) {
            try {
                mermaidContainerRef.current.innerHTML = ''; // Clear previous
                const { svg } = await mermaid.render(`mermaid-graph-${Date.now()}`,
                mermaidCode);
                mermaidContainerRef.current.innerHTML = svg;
            } catch (e) {
                console.error("Mermaid rendering error:", e);
                mermaidContainerRef.current.innerHTML = `<p class="text-red-500">Error rendering
                flowchart.</p>`;
            }
        }
    }
    renderMermaid();
}, [activeTab, mermaidCode]);


const handleScroll = () => {
    if (preRef.current && textareaRef.current) {
        preRef.current.scrollTop = textareaRef.current.scrollTop;
        preRef.current.scrollLeft = textareaRef.current.scrollLeft;
    }
};

const highlightedCode = useMemo(() => simpleSyntaxHighlight(code), [code]);

const renderTabContent = () => {
    if (!explanation) return null;
    switch(activeTab) {
        case 'summary':
            return <MarkdownRenderer content={explanation.summary} />;
        case 'lineByLine':
            return (
                <div className="space-y-3">
                    {explanation.lineByLine.map((item, index) => (
                        <div key={index} className="p-3 bg-background rounded-md border border-border">
                            <p className="font-mono text-xs text-primary mb-1">Lines: {item.lines}</p>
                            <p className="text-sm">{item.explanation}</p>
                        </div>
                    ))}
                </div>
            );
        case 'complexity':
            return (
                <div>
                    <p><strong>Time Complexity:</strong> <span className="font-mono text-
                    amber-600">{explanation.complexity.time}</span></p>
```

```
                    <p><strong>Space Complexity:</strong> <span className="font-mono text-
                    amber-600">{explanation.complexity.space}</span></p>
                </div>
            );
        case 'suggestions':
            return (
                 <ul className="list-disc list-inside space-y-2">
                    {explanation.suggestions.map((item, index) => <li key={index}>{item}</li>)}
                </ul>
            );
        case 'flowchart':
            return (
                <div ref={mermaidContainerRef} className="w-full h-full flex items-center
                justify-center">
                    <LoadingSpinner />
                </div>
            );
    }
}

return (
    <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
        <header className="mb-6 flex-shrink-0">
            <h1 className="text-3xl font-bold flex items-center">
                <CpuChipIcon />
                <span className="ml-3">AI Code Explainer</span>
            </h1>
            <p className="text-text-secondary mt-1">Get a detailed, structured analysis of
            any code snippet.</p>
        </header>
        <div className="flex-grow grid grid-cols-1 md:grid-cols-2 gap-6 min-h-0">

            {/* Left Column: Code Input */}
            <div className="flex flex-col min-h-0 md:col-span-1">
                <label htmlFor="code-input" className="text-sm font-medium text-text-secondary
                mb-2">Your Code</label>
                <div className="relative flex-grow bg-surface border border-border rounded-md
                focus-within:ring-2 focus-within:ring-primary overflow-hidden">
                    <textarea
                        ref={textareaRef}
                        id="code-input"
                        value={code}
                        onChange={(e) => setCode(e.target.value)}
                        onScroll={handleScroll}
                        placeholder="Paste your code here..."
                        spellCheck="false"
                        className="absolute inset-0 w-full h-full p-4 bg-transparent resize-none font-
                        mono text-sm text-transparent caret-primary outline-none z-10"
                    />
                    <pre
                        ref={preRef}
                        aria-hidden="true"
                        className="absolute inset-0 w-full h-full p-4 font-mono text-sm text-text-
                        primary pointer-events-none z-0 whitespace-pre-wrap overflow-auto no-scrollbar"
                        dangerouslySetInnerHTML={{ __html: highlightedCode + '\n' }}
                    />
                </div>
                <div className="mt-4 flex-shrink-0">
                    <button
                        onClick={() => handleExplain(code)}
                        disabled={isLoading}
                        className="btn-primary w-full flex items-center justify-center px-6 py-3"
```

```
                        >
                            {isLoading ? <LoadingSpinner/> : 'Analyze Code'}
                        </button>
                    </div>
                </div>

                {/* Right Column: AI Analysis */}
                <div className="flex flex-col min-h-0 md:col-span-1">
                    <label className="text-sm font-medium text-text-secondary mb-2">AI
                    Analysis</label>
                    <div className="relative flex-grow flex flex-col bg-surface border border-border
                    rounded-md overflow-hidden">
                        <div className="flex-shrink-0 flex border-b border-border">
                           {(['summary', 'lineByLine', 'complexity', 'suggestions', 'flowchart'] as
                           ExplanationTab[]).map(tab => (
                               <button key={tab} onClick={() => setActiveTab(tab)} disabled={!explanation}
                                className={`px-4 py-2 text-sm font-medium capitalize transition-colors
                                ${activeTab === tab ? 'bg-background text-primary font-semibold' : 'text-text-
                                secondary hover:bg-gray-100 dark:hover:bg-slate-700 disabled:text-gray-400
                                dark:disabled:text-slate-500'}`}>
                                   {tab.replace(/([A-Z])/g, ' $1')}
                               </button>
                           ))}
                        </div>
                        <div className="p-4 flex-grow overflow-y-auto">
                            {isLoading && <div className="flex items-center justify-center
                            h-full"><LoadingSpinner /></div>}
                            {error && <p className="text-red-500">{error}</p>}
                            {explanation && !isLoading && renderTabContent()}
                            {!isLoading && !explanation && !error && <div className="text-text-secondary
                            h-full flex items-center justify-center">The analysis will appear here.</div>}
                        </div>
                    </div>
                </div>
            </div>
        </div>
    );
};

// ===== VisualGitTree_36.tsx =====

import React, { useState, useCallback, useEffect, useMemo } from 'react';
import { GitBranchIcon, ArrowDownTrayIcon } from '../icons.tsx';
import { generateChangelogFromLogStream } from '../../services/aiService.ts';
import { LoadingSpinner, MarkdownRenderer } from '../shared/index.tsx';
import { downloadFile } from '../../services/fileUtils.ts';

const CommitGraph = ({ logInput }: { logInput: string }) => {
    const commits = useMemo(() => {
        const lines = logInput.split('\n');
        const parsedCommits: any[] = [];
        let currentCommit: any = null;

        lines.forEach(line => {
            const commitMatch = line.match(/^.?[\\|/ ]*\* commit (\w+)(.*)/);
            if (commitMatch) {
                if (currentCommit) parsedCommits.push(currentCommit);
                currentCommit = {
                    hash: commitMatch[1],
                    shortHash: commitMatch[1].substring(0, 7),
                    refs: commitMatch[2].trim(),
                    message: '',
```

```
                    author: '',
                };
            } else if (currentCommit) {
                 if (line.includes('Author:')) currentCommit.author =
                 line.split('Author:')[1].trim();
                 else if (line.trim().length > 0 && !line.match(/^[\\|/ ]*[\\|/ ]/)) {
                     currentCommit.message += line.trim() + ' ';
                 }
            }
        });
        if (currentCommit) parsedCommits.push(currentCommit);

        return parsedCommits.map((c, i) => ({ ...c, x: 50, y: 50 + i * 60 }));
    }, [logInput]);

    return (
         <svg width="100%" height={50 + commits.length * 60} className="min-h-[200px]">
            {commits.map((commit, i) => {
                const parent = commits[i + 1];
                return (
                    <g key={commit.hash}>
                        {parent && <line x1={commit.x} y1={commit.y} x2={parent.x} y2={parent.y}
                        stroke="var(--color-border)" strokeWidth="2" />}
                        <g className="group cursor-pointer">
                            <circle cx={commit.x} cy={commit.y} r="8" fill="var(--color-primary)"
                            stroke="var(--color-surface)" strokeWidth="3" />
                            <foreignObject x={commit.x + 20} y={commit.y - 25} width="350" height="50">
                                <div className="text-sm p-1">
                                    <p className="font-bold truncate text-text-primary">{commit.message}</p>
                                    <p className="text-xs text-text-secondary font-mono">{commit.shortHash} <span
                                    className="text-amber-600">{commit.refs}</span></p>
                                </div>
                            </foreignObject>
                            <title>{`Commit: ${commit.hash}\nAuthor:
                            ${commit.author}\n\n${commit.message}`}</title>
                        </g>
                    </g>
                );
            })}
        </svg>
    );
};

export const VisualGitTree: React.FC<{ logInput?: string }> = ({ logInput:
initialLogInput }) => {
    const [logInput, setLogInput] = useState(initialLogInput || '');
    const [analysis, setAnalysis] = useState('');
    const [isLoading, setIsLoading] = useState(false);
    const [error, setError] = useState('');

    const handleAnalyze = useCallback(async (logToAnalyze: string) => {
        if (!logToAnalyze.trim()) {
            setError('Please paste git log output.');
            return;
        }
        setIsLoading(true);
        setError('');
        setAnalysis('');
        try {
            const stream = generateChangelogFromLogStream(logToAnalyze);
            let fullResponse = '';
            for await (const chunk of stream) {
```

```
            fullResponse += chunk;
            setAnalysis(fullResponse);
        }
    } catch (err) {
        const errorMessage = err instanceof Error ? err.message : 'An unknown error
        occurred.';
        setError(`Failed to analyze log: ${errorMessage}`);
    } finally {
        setIsLoading(false);
    }
}, []);

useEffect(() => {
    if (initialLogInput) {
        setLogInput(initialLogInput);
        handleAnalyze(initialLogInput);
    }
}, [initialLogInput, handleAnalyze]);

return (
    <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
        <header className="mb-6">
            <h1 className="text-3xl font-bold flex items-center">
                <GitBranchIcon />
                <span className="ml-3">Visual Git Tree</span>
            </h1>
            <p className="text-text-secondary mt-1">Paste your `git log --graph` output to
            visualize the history and get an AI summary.</p>
        </header>
        <div className="flex-grow grid grid-cols-1 lg:grid-cols-2 gap-6 h-full overflow-
        hidden">
            <div className="flex flex-col h-full">
                <label htmlFor="log-input" className="text-sm font-medium text-text-secondary
                mb-2">Git Log Output</label>
                <textarea
                    id="log-input"
                    value={logInput}
                    onChange={(e) => setLogInput(e.target.value)}
                    placeholder="Paste your git log output here..."
                    className="flex-grow p-4 bg-surface border border-border rounded-md resize-none
                    font-mono text-sm"
                />
                <button
                    onClick={() => handleAnalyze(logInput)}
                    disabled={isLoading}
                    className="btn-primary mt-4 w-full flex items-center justify-center px-6 py-3"
                >
                    {isLoading ? <LoadingSpinner /> : 'Analyze & Summarize'}
                </button>
            </div>
            <div className="flex flex-col h-full gap-4">
                <div className="flex flex-col h-1/2">
                    <label className="text-sm font-medium text-text-secondary mb-2">Commit
                    Graph</label>
                    <div className="flex-grow p-2 bg-surface border border-border rounded-md
                    overflow-auto">
                        <CommitGraph logInput={logInput} />
                    </div>
                </div>
                 <div className="flex flex-col h-1/2">
                    <div className="flex justify-between items-center mb-2">
                        <label className="text-sm font-medium text-text-secondary">AI Summary</label>
```

```
                            {analysis && !isLoading && (
                                <button onClick={() => downloadFile(analysis, 'summary.md', 'text/markdown')}
                                className="flex items-center gap-1 px-3 py-1 bg-gray-100 text-xs rounded-md
                                hover:bg-gray-200">
                                    <ArrowDownTrayIcon className="w-4 h-4"/> Download Summary
                                </button>
                            )}
                        </div>
                        <div className="flex-grow p-4 bg-background border border-border rounded-md
                        overflow-y-auto">
                            {isLoading && <div className="flex items-center justify-center
                            h-full"><LoadingSpinner /></div>}
                            {error && <p className="text-red-500">{error}</p>}
                            {analysis && !isLoading && <MarkdownRenderer content={analysis} />}
                            {!isLoading && !analysis && !error && <div className="text-text-secondary h-full
                            flex items-center justify-center">AI summary will appear here.</div>}
                        </div>
                    </div>
                </div>
            </div>
        </div>
    );
};

// ===== SnippetVault_36.tsx =====

import React, { useState, useEffect, useMemo } from 'react';
import { LockClosedIcon, SparklesIcon, TrashIcon, ClipboardDocumentIcon,
ArrowDownTrayIcon } from '../icons.tsx';
import { useLocalStorage } from '../../hooks/useLocalStorage.ts';
import { enhanceSnippetStream, generateTagsForCode } from
'../../services/aiService.ts';
import { LoadingSpinner } from '../shared/index.tsx';
import { downloadFile } from '../../services/fileUtils.ts';
import { useNotification } from '../../contexts/NotificationContext.tsx';

interface Snippet {
    id: number; name: string; code: string; language: string; tags: string[];
}

const langToExt: Record<string, string> = {
    javascript: 'js',
    typescript: 'ts',
    python: 'py',
    css: 'css',
    html: 'html',
    json: 'json',
    markdown: 'md',
    plaintext: 'txt',
};

export const SnippetVault: React.FC = () => {
    const [snippets, setSnippets] = useLocalStorage<Snippet[]>('devcore_snippets',
    []);
    const [activeSnippet, setActiveSnippet] = useState<Snippet | null>(null);
    const [isEnhancing, setIsEnhancing] = useState(false);
    const [searchTerm, setSearchTerm] = useState('');
    const [isEditingName, setIsEditingName] = useState(false);
    const { addNotification } = useNotification();

    const filteredSnippets = useMemo(() => {
        if (!searchTerm) return snippets;
```

```
    const lowerSearch = searchTerm.toLowerCase();
    return snippets.filter((s: Snippet) =>
        s.name.toLowerCase().includes(lowerSearch) ||
        s.code.toLowerCase().includes(lowerSearch) ||
        (s.tags && s.tags.some(t => t.toLowerCase().includes(lowerSearch)))
    );
}, [snippets, searchTerm]);

useEffect(() => {
    if (!activeSnippet && filteredSnippets.length > 0)
    setActiveSnippet(filteredSnippets[0]);
    if (activeSnippet) setActiveSnippet(snippets.find((s: Snippet) => s.id ===
    activeSnippet.id) || null);
}, [snippets, activeSnippet, filteredSnippets]);

const updateSnippet = (snippet: Snippet) => {
    setSnippets(snippets.map((s: Snippet) => s.id === snippet.id ? snippet : s));
    setActiveSnippet(snippet);
};

const handleEnhance = async () => {
    if (!activeSnippet) return;
    setIsEnhancing(true);
    try {
        const stream = enhanceSnippetStream(activeSnippet.code);
        let fullResponse = '';
        for await (const chunk of stream) {
            fullResponse += chunk;
            updateSnippet({ ...activeSnippet, code: fullResponse.replace(/^```(?:\w+\n)?/,
            '').replace(/```$/, '') });
        }
    } finally { setIsEnhancing(false); }
};

const handleAiTagging = async (snippet: Snippet) => {
    if (!snippet.code.trim()) return;
    try {
        const suggestedTags = await generateTagsForCode(snippet.code);
        const newTags = [...new Set([...(snippet.tags || []), ...suggestedTags])];
        updateSnippet({...snippet, tags: newTags});
        addNotification('AI tags added!', 'success');
    } catch(e) {
        console.error("AI tagging failed:", e);
        addNotification('AI tagging failed.', 'error');
    }
};

const handleAddNew = () => {
    const newSnippet: Snippet = { id: Date.now(), name: 'New Snippet', language:
    'plaintext', code: '', tags: [] };
    setSnippets([...snippets, newSnippet]);
    setActiveSnippet(newSnippet);
};

const handleDelete = (id: number) => {
    setSnippets(snippets.filter((s: Snippet) => s.id !== id));
    if(activeSnippet?.id === id) setActiveSnippet(filteredSnippets.length > 1 ?
    filteredSnippets[0] : null);
};

const handleDownload = () => {
    if(!activeSnippet) return;
```

```
    const extension = langToExt[activeSnippet.language] || 'txt';
    const filename = `${activeSnippet.name.replace(/\s/g, '_')}.${extension}`;
    downloadFile(activeSnippet.code, filename);
}

const handleNameChange = (e: React.ChangeEvent<HTMLInputElement>) => {
    if (activeSnippet) updateSnippet({...activeSnippet, name: e.target.value});
};

const handleTagsChange = (e: React.KeyboardEvent<HTMLInputElement>) => {
    if (e.key === 'Enter' && activeSnippet) {
        const newTag = e.currentTarget.value.trim();
        if (newTag && !activeSnippet.tags.includes(newTag)) {
            updateSnippet({...activeSnippet, tags: [...(activeSnippet.tags ?? []),
            newTag]});
        }
        e.currentTarget.value = '';
    }
};

return (
    <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
        <header className="mb-6"><h1 className="text-3xl font-bold flex items-
        center"><LockClosedIcon /><span className="ml-3">Snippet Vault</span></h1><p
        className="text-text-secondary mt-1">Store, search, tag, and enhance your
        reusable code snippets with AI.</p></header>
        <div className="flex-grow flex gap-6 min-h-0">
            <aside className="w-1/3 bg-surface border border-border p-4 rounded-lg flex
            flex-col">
                <input type="text" placeholder="Search snippets..." value={searchTerm}
                onChange={e => setSearchTerm(e.target.value)} className="w-full px-3 py-1.5 mb-3
                rounded-md bg-background border border-border text-sm"/>
                <ul className="space-y-2 flex-grow overflow-y-auto
                pr-2">{filteredSnippets.map((s: Snippet) => (<li key={s.id} className="group
                flex items-center justify-between"><button onClick={() => setActiveSnippet(s)}
                className={`w-full text-left px-3 py-2 rounded-md ${activeSnippet?.id === s.id ?
                'bg-primary/10 text-primary' : 'hover:bg-gray-100 dark:hover:bg-
                slate-700'}`}>{s.name}</button><div className="flex opacity-0 group-
                hover:opacity-100 transition-opacity"><button onClick={() =>
                {navigator.clipboard.writeText(s.code); addNotification("Copied snippet!",
                "success")}} className="ml-2 p-1 text-text-secondary hover:text-primary"
                title="Copy"><ClipboardDocumentIcon /></button><button onClick={() =>
                handleDelete(s.id)} className="ml-2 p-1 text-text-secondary hover:text-red-500"
                title="Delete"><TrashIcon/></button></div></li>))}</ul>
                <div className="mt-4 pt-4 border-t border-border"><button onClick={handleAddNew}
                className="btn-primary w-full text-sm py-2">Add New Snippet</button></div>
            </aside>
            <main className="w-2/3 flex flex-col">
                {activeSnippet ? (<>
                    <div className="flex justify-between items-center mb-2">
                        {isEditingName ? <input type="text" value={activeSnippet.name}
                        onChange={handleNameChange} onBlur={() => setIsEditingName(false)} autoFocus
                        className="text-lg font-bold bg-gray-100 dark:bg-slate-700 rounded px-2"/> : <h3
                        onDoubleClick={() => setIsEditingName(true)} className="text-lg font-bold
                        cursor-pointer">{activeSnippet.name}</h3>}
                        <div className="flex gap-2">
                            <button onClick={() => handleAiTagging(activeSnippet)} className="flex items-
                            center gap-2 px-3 py-1 bg-teal-500/80 text-white font-bold text-xs rounded-
                            md"><SparklesIcon /> AI Tag</button>
                            <button onClick={handleEnhance} disabled={isEnhancing} className="flex items-
                            center gap-2 px-3 py-1 bg-purple-500/80 text-white font-bold text-xs rounded-md
                            disabled:bg-gray-400"><SparklesIcon /> AI Enhance</button>
```

```
                            <button onClick={handleDownload} className="flex items-center gap-1 px-3 py-1
                            bg-gray-100 dark:bg-slate-700 text-xs rounded-md"><ArrowDownTrayIcon
                            className="w-4 h-4"/> Download</button>
                        </div>
                    </div>
                    <textarea value={activeSnippet.code} onChange={e =>
                    updateSnippet({...activeSnippet, code: e.target.value})} className="flex-grow
                    p-4 bg-surface border border-border rounded-md resize-none font-mono text-sm
                    focus:ring-2 focus:ring-primary focus:outline-none"/>
                    <div className="mt-2 text-xs text-text-secondary">
                       <div className="flex items-center gap-2 flex-wrap">
                         <span className="font-bold">Tags:</span> {(activeSnippet.tags ?? []).map(t =>
                         <span key={t} className="bg-gray-200 dark:bg-slate-700 px-2 py-0.5 rounded-
                         full">{t}</span>)}
                         <input type="text" placeholder="+ Add tag" onKeyDown={handleTagsChange}
                         className="bg-transparent border-b border-border focus:outline-none
                         focus:border-primary w-24 text-xs px-1"/>
                       </div>
                    </div>
                </>) : (<div className="flex-grow flex items-center justify-center bg-background
                border border-border rounded-lg text-text-secondary">Select a snippet or create
                a new one.</div>)}
            </main>
        </div>
    </div>
  );
};

// ===== RegexSandbox_36.tsx =====

import React, { useState, useMemo, useCallback, useEffect } from 'react';
import { generateRegExStream } from '../../services/aiService.ts';
import { BeakerIcon } from '../icons.tsx';
import { LoadingSpinner } from '../shared/index.tsx';

const commonPatterns = [
    { name: 'Email', pattern: '/[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\\.[a-zA-Z]{2,}/g'
    },
    { name: 'URL', pattern: '/https?:\\/\\/(www\\.)?[-a-zA-Z0-
    9@:%._\\+~#=]{1,256}\\.[a-zA-Z0-9()]{1,6}\\b([-a-zA-Z0-9()@:%_\\+.~#?&//=]*)/g'
    },
    { name: 'IPv4 Address', pattern:
    '/((25[0-5]|(2[0-4]|1\\d|[1-9]|)\\d)\\.?\\b){4}/g' },
    { name: 'Date (YYYY-MM-DD)', pattern: '/\\d{4}-\\d{2}-\\d{2}/g' },
];

const CheatSheet = () => (
    <div className="bg-surface border border-border p-4 rounded-lg">
        <h3 className="text-lg font-bold mb-2">Regex Cheat Sheet</h3>
        <div className="grid grid-cols-2 gap-x-4 gap-y-1 text-xs font-mono">
            <p><span className="text-primary">.</span> - Any character</p>
            <p><span className="text-primary">\d</span> - Any digit</p>
            <p><span className="text-primary">\w</span> - Word character</p>
            <p><span className="text-primary">\s</span> - Whitespace</p>
            <p><span className="text-primary">[abc]</span> - a, b, or c</p>
            <p><span className="text-primary">[^abc]</span> - Not a, b, or c</p>
            <p><span className="text-primary">*</span> - 0 or more</p>
            <p><span className="text-primary">+</span> - 1 or more</p>
            <p><span className="text-primary">?</span> - 0 or one</p>
            <p><span className="text-primary">^</span> - Start of string</p>
            <p><span className="text-primary">$</span> - End of string</p>
            <p><span className="text-primary">\b</span> - Word boundary</p>
```

```
        </div>
    </div>
);

export const RegexSandbox: React.FC<{ initialPrompt?: string }> = ({
initialPrompt }) => {
    const [pattern, setPattern] = useState<string>('');
    const [testString, setTestString] = useState<string>('');
    const [aiPrompt, setAiPrompt] = useState<string>(initialPrompt || '');
    const [isAiLoading, setIsAiLoading] = useState<boolean>(false);

    const { matches, error } = useMemo(() => {
        if (!pattern) return { matches: null, error: null };
        try {
            const patternParts = pattern.match(/^\/(.*)\/([gimyus]*)$/);
            if (!patternParts) return { matches: null, error: 'Invalid regex literal. Use
            /pattern/flags.' };
            const [, regexBody, regexFlags] = patternParts;
            const regex = new RegExp(regexBody, regexFlags);
            return { matches: [...testString.matchAll(regex)], error: null };
        } catch (e) { return { matches: null, error: e instanceof Error ? e.message :
        'Unknown error.' }; }
    }, [pattern, testString]);

    const handleGenerateRegex = useCallback(async (p: string) => {
        if (!p) return;
        setIsAiLoading(true);
        try {
            const stream = generateRegExStream(p);
            let fullResponse = '';
            for await (const chunk of stream) { fullResponse += chunk; }
            setPattern(fullResponse.trim().replace(/^`+|`+$/g, ''));
        } finally { setIsAiLoading(false); }
    }, []);

    useEffect(() => { if (initialPrompt) handleGenerateRegex(initialPrompt); },
    [initialPrompt, handleGenerateRegex]);

    const highlightedString = useMemo(() => {
        if (!matches || matches.length === 0 || error) return testString;
        let lastIndex = 0;
        const parts: (string | JSX.Element)[] = [];
        matches.forEach((match, i) => {
            if (match.index === undefined) return;
            parts.push(testString.substring(lastIndex, match.index));
            parts.push(<mark key={i} className="bg-primary/20 text-primary rounded
            px-1">{match[0]}</mark>);
            lastIndex = match.index + match[0].length;
        });
        parts.push(testString.substring(lastIndex));
        return parts;
    }, [matches, testString, error]);

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
            <header className="mb-6"><h1 className="text-3xl font-bold flex items-
            center"><BeakerIcon /><span className="ml-3">RegEx Sandbox</span></h1><p
            className="text-text-secondary mt-1">Test your regular expressions and generate
            them with AI.</p></header>
            <div className="flex-grow grid grid-cols-1 lg:grid-cols-3 gap-6 min-h-0">
                <div className="lg:col-span-2 flex flex-col gap-4">
                    <div className="flex gap-2"><input type="text" value={aiPrompt} onChange={(e) =>
```

```
                    setAiPrompt(e.target.value)} placeholder="Describe the pattern to find..."
                    className="flex-grow px-3 py-1.5 rounded-md bg-surface border border-border
                    text-sm focus:ring-2 focus:ring-primary" /><button onClick={() =>
                    handleGenerateRegex(aiPrompt)} disabled={isAiLoading} className="btn-primary
                    px-4 py-1.5 flex items-center">{isAiLoading ? <LoadingSpinner/> :
                    'Generate'}</button></div>
                    <div><label htmlFor="regex-pattern" className="text-sm font-medium text-text-
                    secondary">Regular Expression</label><input id="regex-pattern" type="text"
                    value={pattern} onChange={(e) => setPattern(e.target.value)} className={`w-full
                    mt-1 px-3 py-2 rounded-md bg-surface border ${error ? 'border-red-500' :
                    'border-border'} font-mono text-sm focus:ring-2 focus:ring-primary`} />{error &&
                    <p className="text-red-500 text-xs mt-1">{error}</p>}</div>
                    <div className="flex flex-col flex-grow min-h-0"><label htmlFor="test-string"
                    className="text-sm font-medium text-text-secondary">Test String</label><textarea
                    id="test-string" value={testString} onChange={(e) =>
                    setTestString(e.target.value)} className="w-full mt-1 p-3 rounded-md bg-surface
                    border border-border font-mono text-sm resize-y h-32" /><div className="mt-2 p-3
                    bg-background rounded-md border border-border min-h-[50px] whitespace-pre-
                    wrap">{highlightedString}</div></div>
                    <div className="flex-shrink-0"><h3 className="text-lg font-bold">Match Groups
                    ({matches?.length || 0})</h3><div className="mt-2 p-2 bg-surface rounded-md
                    overflow-y-auto max-h-48 font-mono text-xs border border-border">{matches &&
                    matches.length > 0 ? (matches.map((match, i) => (<details key={i} className="p-2
                    border-b border-border"><summary className="cursor-pointer text-green-700">Match
                    {i + 1}: "{match[0]}"</summary><div className="pl-4
                    mt-1">{Array.from(match).map((group, gIndex) => <p key={gIndex} className="text-
                    text-secondary">Group {gIndex}: <span className="text-
                    amber-700">{String(group)}</span></p>)}</div></details>))) : (<p
                    className="text-text-secondary text-sm p-2">No matches found.</p>)}</div></div>
                </div>
                <div className="lg:col-span-1 space-y-4">
                    <CheatSheet />
                    <div className="bg-surface border border-border p-4 rounded-lg">
                        <h3 className="text-lg font-bold mb-2">Common Patterns</h3>
                        <div className="flex flex-col items-start gap-2">
                            {commonPatterns.map(p => (
                                <button key={p.name} onClick={() => setPattern(p.pattern)} className="text-left
                                text-sm text-primary hover:underline">
                                    {p.name}
                                </button>
                            ))}
                        </div>
                    </div>
                </div>
            </div>
        </div>
    );
};

// ===== AiFeatureBuilder_36.tsx =====

import React, { useState, useCallback, useEffect } from 'react';
import type { GeneratedFile, ViewType } from '../../types.ts';
import { generateFeature, generateFullStackFeature, generateUnitTestsStream,
generateCommitMessageStream, generateDockerfile } from
'../../services/aiService.ts';
import { saveFile, getAllFiles, clearAllFiles } from
'../../services/dbService.ts';
import { useGlobalState } from '../../contexts/GlobalStateContext.tsx';
import { CpuChipIcon, DocumentTextIcon, BeakerIcon, GitBranchIcon, CloudIcon,
SparklesIcon } from '../icons.tsx';
import { LoadingSpinner, MarkdownRenderer } from '../shared/index.tsx';
```

```
type SupplementalTab = 'TESTS' | 'COMMIT' | 'DEPLOYMENT' | 'CODE';
type OutputTab = GeneratedFile | SupplementalTab;

export const AiFeatureBuilder: React.FC = () => {
    const { dispatch } = useGlobalState();
    const [prompt, setPrompt] = useState<string>('');
    const [framework] = useState('React');
    const [styling] = useState('Tailwind CSS');
    const [includeBackend, setIncludeBackend] = useState(false);

    const [generatedFiles, setGeneratedFiles] = useState<GeneratedFile[]>([]);
    const [unitTests, setUnitTests] = useState<string>('');
    const [commitMessage, setCommitMessage] = useState<string>('');
    const [dockerfile, setDockerfile] = useState<string>('');

    const [activeTab, setActiveTab] = useState<OutputTab>('CODE');
    const [isLoading, setIsLoading] = useState<boolean>(false);
    const [error, setError] = useState<string>('');

    useEffect(() => {
        const loadFiles = async () => {
            const files = await getAllFiles();
            setGeneratedFiles(files);
            if (files.length > 0) setActiveTab(files[0]);
        };
        loadFiles();
    }, []);

    const handleGenerate = useCallback(async () => {
        if (!prompt.trim()) { setError('Please enter a feature description.'); return; }
        setIsLoading(true);
        setError('');
        await clearAllFiles();
        setGeneratedFiles([]); setUnitTests(''); setCommitMessage('');
        setDockerfile(''); setActiveTab('CODE');

        try {
            const resultFiles = includeBackend
                ? await generateFullStackFeature(prompt, framework, styling)
                : await generateFeature(prompt, framework, styling);

            for (const file of resultFiles) { await saveFile(file); }
            setGeneratedFiles(resultFiles);

            if (resultFiles.length > 0) {
                const componentFile = resultFiles.find(f => f.filePath.endsWith('.tsx') ||
                f.filePath.endsWith('.jsx'));
                setActiveTab(componentFile || resultFiles[0]);

                const testStream = generateUnitTestsStream(componentFile?.content ||
                resultFiles[0].content);
                const diffContext = resultFiles.map(f => `File:
                ${f.filePath}\n\n${f.content}`).join('\n---\n');
                const commitStream = generateCommitMessageStream(diffContext);

                let tests = ''; for await (const chunk of testStream) { tests += chunk;
                setUnitTests(tests); }
                let commit = ''; for await (const chunk of commitStream) { commit += chunk;
                setCommitMessage(commit); }

                if (!includeBackend) {
                    const dockerfileStream = generateDockerfile(framework);
```

```
                let docker = ''; for await (const chunk of dockerfileStream) { docker += chunk;
                setDockerfile(docker); }
            }
        }
    } catch (err) {
        setError(err instanceof Error ? err.message : 'Failed to generate feature.');
    } finally {
        setIsLoading(false);
    }
}, [prompt, framework, styling, includeBackend]);

const handleSendTo = (view: ViewType, props: any) => {
    dispatch({ type: 'SET_VIEW', payload: { view, props } });
};

const renderContent = () => {
    if (typeof activeTab === 'string') {
        switch (activeTab) {
            case 'TESTS': return <MarkdownRenderer content={unitTests} />;
            case 'COMMIT': return <pre className="w-full h-full p-4 whitespace-pre-wrap
            font-sans text-sm">{commitMessage}</pre>;
            case 'DEPLOYMENT': return <MarkdownRenderer content={dockerfile} />;
            default: return <div className="p-4">Select a file</div>;
        }
    }
    return <MarkdownRenderer content={'```tsx\n' + activeTab.content + '\n```'} />;
}

return (
    <div className="h-full flex flex-col text-text-primary bg-surface">
        <header className="p-4 border-b border-border flex-shrink-0">
            <h1 className="text-xl font-bold flex items-center"><CpuChipIcon /><span
            className="ml-3">AI Feature Builder</span></h1>
        </header>
        <div className="flex-grow flex min-h-0">
            <main className="flex-1 flex flex-col min-w-0">
                <div className="flex-grow flex flex-col bg-background">
                     <div className="border-b border-border flex items-center bg-surface overflow-x-
                     auto">
                        {generatedFiles.map(file => (
                            <button key={file.filePath} onClick={() => setActiveTab(file)} className={`flex-
                            shrink-0 flex items-center gap-2 px-4 py-2 text-sm ${activeTab === file ? 'bg-
                            background border-b-2 border-primary text-text-primary' : 'text-text-secondary
                            hover:bg-gray-50'}`}><DocumentTextIcon /> {file.filePath}</button>
                        ))}
                        {unitTests && <button onClick={() => setActiveTab('TESTS')} className={`flex-
                        shrink-0 flex items-center gap-2 px-4 py-2 text-sm ${activeTab === 'TESTS' ?
                        'bg-background border-b-2 border-primary text-text-primary' : 'text-text-
                        secondary hover:bg-gray-50'}`}><BeakerIcon /> Tests</button>}
                        {commitMessage && <button onClick={() => setActiveTab('COMMIT')}
                        className={`flex-shrink-0 flex items-center gap-2 px-4 py-2 text-sm ${activeTab
                        === 'COMMIT' ? 'bg-background border-b-2 border-primary text-text-primary' :
                        'text-text-secondary hover:bg-gray-50'}`}><GitBranchIcon /> Commit</button>}
                        {dockerfile && !includeBackend && <button onClick={() =>
                        setActiveTab('DEPLOYMENT')} className={`flex-shrink-0 flex items-center gap-2
                        px-4 py-2 text-sm ${activeTab === 'DEPLOYMENT' ? 'bg-background border-b-2
                        border-primary text-text-primary' : 'text-text-secondary hover:bg-
                        gray-50'}`}><CloudIcon /> Dockerfile</button>}
                    </div>
                    <div className="flex-grow p-2 overflow-auto">
                        {isLoading && !generatedFiles.length ? <div className="flex justify-center
                        items-center h-full"><LoadingSpinner/></div> : renderContent()}
```

```
                    </div>
                </div>

                {!isLoading && generatedFiles.length > 0 && (
                    <div className="flex-shrink-0 p-3 border-t border-border bg-surface/80">
                        <h3 className="text-sm font-semibold mb-2 text-center">Next Steps</h3>
                        <div className="flex justify-center gap-2">
                            <button onClick={() => handleSendTo('ai-code-explainer', { initialCode: (typeof
                            activeTab !== 'string' ? activeTab.content : generatedFiles[0].content) })}
                            className="btn-primary text-xs px-3 py-1 flex items-center
                            gap-1"><SparklesIcon/> Explain Code</button>
                            <button onClick={() => handleSendTo('ai-unit-test-generator', { codeInput:
                            (typeof activeTab !== 'string' ? activeTab.content : generatedFiles[0].content)
                            })} className="btn-primary text-xs px-3 py-1 flex items-center
                            gap-1"><BeakerIcon/> Generate Tests</button>
                            <button onClick={() => handleSendTo('ai-commit-generator', { diff: `New feature:
                            ${prompt}` })} className="btn-primary text-xs px-3 py-1 flex items-center
                            gap-1"><GitBranchIcon/> Get Commit Message</button>
                        </div>
                    </div>
                )}

                <div className="flex-shrink-0 p-4 border-t border-border bg-surface">
                     <div className="flex items-center gap-2 mb-2">
                        <label className="flex items-center gap-2 text-sm"><input type="checkbox"
                        checked={includeBackend} onChange={e => setIncludeBackend(e.target.checked)} />
                        Include Backend (Cloud Function + Firestore)</label>
                    </div>
                    <textarea value={prompt} onChange={(e) => setPrompt(e.target.value)}
                    placeholder="e.g., A user profile card with an avatar, name, and bio."
                    className="w-full p-2 bg-background border border-border rounded-md resize-none
                    text-sm h-20"/>
                     <div className="flex gap-2 mt-2">
                         <button onClick={handleGenerate} disabled={isLoading} className="btn-primary
                         flex-grow flex items-center justify-center gap-2 px-4 py-2">
                            {isLoading ? <><LoadingSpinner /> Generating...</> : 'Generate Feature'}
                        </button>
                     </div>
                     {error && <p className="text-red-600 text-xs mt-2 text-center">{error}</p>}
                </div>
            </main>
        </div>
    </div>
  );
};

// ===== AiCommandCenter_34.tsx =====

import React, { useState, useCallback } from 'react';
import { Type, FunctionDeclaration } from "@google/genai";
import { logError } from '../../services/telemetryService.ts';
import { getInferenceFunction, CommandResponse } from
'../../services/aiService.ts';
import { FEATURE_TAXONOMY } from '../../services/taxonomyService.ts';
import { useGlobalState } from '../../contexts/GlobalStateContext.tsx';
import { CommandLineIcon } from '../icons.tsx';
import { LoadingSpinner } from '../shared/index.tsx';
import { ALL_FEATURES } from './index.ts';
import { executeWorkspaceAction, ACTION_REGISTRY } from
'../../services/workspaceConnectorService.ts';

const baseFunctionDeclarations: FunctionDeclaration[] = [
```

```
    {
        name: 'navigateTo',
        description: 'Navigates to a specific feature page.',
        parameters: {
            type: Type.OBJECT,
            properties: {
                featureId: {
                    type: Type.STRING,
                    description: 'The ID of the feature to navigate to.',
                    enum: ALL_FEATURES.map(f => f.id)
                },
            },
            required: ['featureId'],
        },
    },
    {
        name: 'runFeatureWithInput',
        description: 'Navigates to a feature and passes initial data to it.',
        parameters: {
            type: Type.OBJECT,
            properties: {
                 featureId: {
                    type: Type.STRING,
                    description: 'The ID of the feature to run.',
                    enum: ALL_FEATURES.map(f => f.id)
                },
                props: {
                    type: Type.OBJECT,
                    description: 'An object containing the initial properties for the feature, based
                    on its required inputs.',
                    properties: {
                        initialCode: { type: Type.STRING },
                        initialPrompt: { type: Type.STRING },
                        beforeCode: { type: Type.STRING },
                        afterCode: { type: Type.STRING },
                        logInput: { type: Type.STRING },
                        diff: { type: Type.STRING },
                        codeInput: { type: Type.STRING },
                        jsonInput: { type: Type.STRING },
                    }
                }
            },
            required: ['featureId', 'props']
        }
    }
];

// Dynamically add the workspace action
const functionDeclarations: FunctionDeclaration[] = [
    ...baseFunctionDeclarations,
    {
        name: 'runWorkspaceAction',
        description: 'Executes a defined action on a connected workspace service like
        Jira, Slack, or GitHub.',
        parameters: {
            type: Type.OBJECT,
            properties: {
                 actionId: {
                    type: Type.STRING,
                    description: 'The unique identifier for the action to execute.',
                    enum: [ ...ACTION_REGISTRY.keys() ]
                },
```

```
                params: {
                    type: Type.OBJECT,
                    description: 'An object containing the parameters for the action, matching its
                    required inputs.'
                }
            },
            required: ['actionId', 'params']
        }
    }
]

const knowledgeBase = FEATURE_TAXONOMY.map(f => `- ${f.name} (${f.id}):
${f.description} Inputs: ${f.inputs}`).join('\n');

const ExamplePromptButton: React.FC<{ text: string, onClick: (text: string) =>
void }> = ({ text, onClick }) => (
    <button
        onClick={() => onClick(text)}
        className="px-3 py-1.5 bg-surface border border-border rounded-full text-xs
        hover:bg-gray-100 dark:hover:bg-slate-700 transition-colors"
    >
        {text}
    </button>
)

export const AiCommandCenter: React.FC = () => {
    const { dispatch } = useGlobalState();
    const [prompt, setPrompt] = useState('');
    const [isLoading, setIsLoading] = useState(false);
    const [lastResponse, setLastResponse] = useState('');

    const handleCommand = useCallback(async () => {
        if (!prompt.trim()) return;

        setIsLoading(true);
        setLastResponse('');

        try {
            const response: CommandResponse = await getInferenceFunction(prompt,
            functionDeclarations, knowledgeBase);

            if (response.functionCalls && response.functionCalls.length > 0) {
                const call = response.functionCalls[0];
                const { name, args } = call;

                setLastResponse(`Understood! Executing command: ${name}`);

                switch (name) {
                    case 'navigateTo':
                        dispatch({ type: 'SET_VIEW', payload: { view: args.featureId }});
                        break;
                    case 'runFeatureWithInput':
                         dispatch({ type: 'SET_VIEW', payload: { view: args.featureId, props: args.props
                         } });
                        break;
                    case 'runWorkspaceAction':
                        try {
                            const result = await executeWorkspaceAction(args.actionId, args.params);
                            setLastResponse(`Action '${args.actionId}' executed successfully.\n\nResult:
                            \`\`\`json\n${JSON.stringify(result, null, 2)}\n\`\`\``);
                        } catch (e) {
                            setLastResponse(`Action failed: ${e instanceof Error ? e.message : 'Unknown
```

```
                        error'}`);
                    }
                    break;
                default:
                    setLastResponse(`Unknown command: ${name}`);
            }
             setPrompt('');
        } else {
             setLastResponse(response.text);
        }

    } catch (err) {
        logError(err as Error, { prompt });
        setLastResponse(err instanceof Error ? err.message : 'An unknown error
        occurred.');
    } finally {
        setIsLoading(false);
    }
}, [prompt, dispatch]);

const handleKeyDown = (e: React.KeyboardEvent) => {
    if (e.key === 'Enter' && !e.shiftKey) {
        e.preventDefault();
        handleCommand();
    }
};

const handleExampleClick = (text: string) => {
    setPrompt(text);
}

return (
    <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
        <header className="mb-6 text-center">
            <h1 className="text-4xl font-extrabold tracking-tight flex items-center justify-
            center">
                <CommandLineIcon />
                <span className="ml-3">AI Command Center</span>
            </h1>
            <p className="mt-2 text-lg text-text-secondary">What would you like to do?</p>
        </header>

        <div className="flex-grow flex flex-col justify-end max-w-3xl w-full mx-auto">
            {lastResponse && (
                <div className="mb-4 p-4 bg-surface rounded-lg text-text-primary border border-
                border">
                    <p><strong>AI:</strong> {lastResponse}</p>
                </div>
            )}
             <div className="relative">
                <textarea
                    value={prompt}
                    onChange={e => setPrompt(e.target.value)}
                    onKeyDown={handleKeyDown}
                    disabled={isLoading}
                    placeholder='Try "explain this code: const a = 1;" or "open the theme designer"'
                    className="w-full p-4 pr-28 rounded-lg bg-surface border border-border
                    focus:ring-2 focus:ring-primary focus:outline-none resize-none shadow-sm"
                    rows={2}
                />
                <button
                    onClick={handleCommand}
```

```
                        disabled={isLoading}
                        className="btn-primary absolute right-3 top-1/2 -translate-y-1/2 px-4 py-2"
                    >
                       {isLoading ? <LoadingSpinner/> : 'Send'}
                    </button>
                </div>
                 <div className="flex flex-wrap items-center justify-center gap-2 mt-4">
                    <ExamplePromptButton text="Open Theme Designer" onClick={handleExampleClick} />
                    <ExamplePromptButton text="Generate a commit for a bug fix"
                    onClick={handleExampleClick} />
                    <ExamplePromptButton text="Create a regex for email validation"
                    onClick={handleExampleClick} />
                </div>
                 <p className="text-xs text-text-secondary text-center mt-2">Press Enter to send,
                 Shift+Enter for new line.</p>
            </div>
        </div>
    );
};

// ===== AiPullRequestAssistant_34.tsx =====


import React, { useState, useMemo, useCallback } from 'react';
import * as Diff from 'diff';
import { generatePrSummaryStructured, generateTechnicalSpecFromDiff,
downloadFile } from '../../services/index.ts';
import { createDocument, insertText } from '../../services/workspaceService.ts';
import type { StructuredPrSummary } from '../../types.ts';
import { AiPullRequestAssistantIcon, DocumentIcon } from '../icons.tsx';
import { LoadingSpinner } from '../shared/index.tsx';
import { useNotification } from '../../contexts/NotificationContext.tsx';
import { useGlobalState } from '../../contexts/GlobalStateContext.tsx';

export const AiPullRequestAssistant: React.FC = () => {
    const [beforeCode, setBeforeCode] = useState<string>('');
    const [afterCode, setAfterCode] = useState<string>('');
    const [isLoading, setIsLoading] = useState<boolean>(false);
    const [isExporting, setIsExporting] = useState<boolean>(false);
    const [error, setError] = useState<string>('');
    const [summary, setSummary] = useState<StructuredPrSummary | null>(null);

    const { addNotification } = useNotification();
    const { state } = useGlobalState();
    const { user } = state;

    const diff = useMemo(() => Diff.createPatch('component.tsx', beforeCode,
    afterCode), [beforeCode, afterCode]);

    const handleGenerateSummary = useCallback(async () => {
        if (!beforeCode.trim() && !afterCode.trim()) {
            setError('Please provide code to generate a summary.');
            return;
        }
        setIsLoading(true);
        setError('');
        setSummary(null);

        try {
            const result: StructuredPrSummary = await generatePrSummaryStructured(diff);
            setSummary(result);
        } catch (err) {
```

```
        const errorMessage = err instanceof Error ? err.message : 'An unknown error
        occurred.';
        setError(`Failed to generate summary: ${errorMessage}`);
    } finally {
        setIsLoading(false);
    }
}, [diff, beforeCode, afterCode]);

const handleExportToDocs = async () => {
    if (!summary || !user) {
        addNotification('Please generate a summary first and ensure you are signed in.',
        'error');
        return;
    }
    setIsExporting(true);
    try {
        const specContent = await generateTechnicalSpecFromDiff(diff, summary);
        const doc = await createDocument(`Tech Spec: ${summary.title}`);
        await insertText(doc.documentId, specContent);
        addNotification('Successfully exported to Google Docs!', 'success');
        window.open(doc.webViewLink, '_blank');
    } catch (err) {
        const errorMessage = err instanceof Error ? err.message : 'An unknown error
        occurred.';
        addNotification(`Failed to export: ${errorMessage}`, 'error');
    } finally {
        setIsExporting(false);
    }
};

return (
    <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
        <header className="mb-6">
            <h1 className="text-3xl font-bold flex items-center">
                <AiPullRequestAssistantIcon />
                <span className="ml-3">AI Pull Request Assistant</span>
            </h1>
            <p className="text-text-secondary mt-1">Generate a PR summary from code changes
            and export a full tech spec.</p>
        </header>
        <div className="flex-grow grid grid-cols-1 lg:grid-cols-2 gap-6 min-h-0">
            {/* Left side: Inputs and Generator */}
            <div className="flex flex-col gap-4 min-h-0">
                <div className="flex flex-col flex-1 min-h-0">
                    <label htmlFor="before-code" className="text-sm font-medium text-text-secondary
                    mb-2">Before</label>
                    <textarea id="before-code" value={beforeCode} onChange={e =>
                    setBeforeCode(e.target.value)} className="flex-grow p-4 bg-surface border
                    border-border rounded-md resize-none font-mono text-sm" />
                </div>
                <div className="flex flex-col flex-1 min-h-0">
                    <label htmlFor="after-code" className="text-sm font-medium text-text-secondary
                    mb-2">After</label>
                    <textarea id="after-code" value={afterCode} onChange={e =>
                    setAfterCode(e.target.value)} className="flex-grow p-4 bg-surface border border-
                    border rounded-md resize-none font-mono text-sm" />
                </div>
                <button onClick={handleGenerateSummary} disabled={isLoading} className="btn-
                primary w-full flex items-center justify-center px-6 py-3">
                    {isLoading ? <LoadingSpinner /> : 'Generate Summary'}
                </button>
                {error && <p className="text-red-500 text-xs text-center">{error}</p>}
```

```
                </div>

                {/* Right side: Summary and Export */}
                <div className="flex flex-col gap-4 min-h-0">
                    <div className="flex flex-col bg-surface border border-border p-4 rounded-lg
                    flex-grow min-h-0">
                        <h3 className="text-lg font-bold mb-2">Generated Summary</h3>
                        <div className="flex-grow overflow-y-auto pr-2 space-y-2">
                            {summary ? (
                                <>
                                    <input type="text" readOnly value={summary.title} className="w-full font-bold
                                    p-2 bg-background rounded"/>
                                    <textarea readOnly value={summary.summary} className="w-full h-24 p-2 bg-
                                    background rounded resize-none"/>
                                    <div>
                                        <h4 className="font-semibold">Changes:</h4>
                                        <ul className="list-disc list-inside text-sm">
                                            {summary.changes.map((c, i) => <li key={i}>{c}</li>)}
                                        </ul>
                                    </div>
                                </>
                            ) : (
                                <div className="text-text-secondary h-full flex items-center justify-center">
                                    {isLoading ? <LoadingSpinner /> : 'PR summary will appear here.'}
                                </div>
                            )}
                        </div>
                         {summary && user && (
                            <div className="mt-4 pt-4 border-t border-border">
                                <button onClick={handleExportToDocs} disabled={isExporting} className="w-full
                                btn-primary bg-blue-600 flex items-center justify-center gap-2 py-2">
                                    {isExporting ? <LoadingSpinner /> : <><DocumentIcon /> Export to Google Docs</>}
                                </button>
                            </div>
                         )}
                    </div>
                </div>
            </div>
        </div>
    );
};

// ===== ProjectExplorer_34.tsx =====

import React, { useState, useEffect, useCallback } from 'react';
import { useGlobalState } from '../../contexts/GlobalStateContext.tsx';
import { useNotification } from '../../contexts/NotificationContext.tsx';
import { initializeOctokit } from '../../services/authService.ts';
import { getUserToken } from '../../services/firebaseService.ts';
import { getRepos, getRepoTree, getFileContent, commitFiles } from
'../../services/githubService.ts';
import { generateCommitMessageStream, reviewCodeStream, generateUnitTestsStream,
explainCodeStream } from '../../services/index.ts';
import type { Repo, FileNode } from '../../types.ts';
import { FolderIcon, DocumentIcon, SparklesIcon } from '../icons.tsx';
import { LoadingSpinner, MarkdownRenderer } from '../shared/index.tsx';
import * as Diff from 'diff';

const FileTree: React.FC<{ node: FileNode, onFileSelect: (path: string, name:
string) => void, activePath: string | null, level?: number }> = ({ node,
onFileSelect, activePath, level = 0 }) => {
    const [isOpen, setIsOpen] = useState(level < 2);
```

```
    if (node.type === 'file') {
        const isActive = activePath === node.path;
        return (
            <div
                className={`flex items-center space-x-2 py-1 cursor-pointer rounded ${isActive ?
                'bg-primary/20 text-primary' : 'hover:bg-navy'}`}
                style={{ paddingLeft: `${level * 16}px`}}
                onClick={() => onFileSelect(node.path, node.name)}
            >
                <DocumentIcon />
                <span className="truncate">{node.name}</span>
            </div>
        );
    }

    return (
        <div>
            <div
                className="flex items-center space-x-2 py-1 cursor-pointer hover:bg-navy
                rounded"
                style={{ paddingLeft: `${level * 16}px`}}
                onClick={() => setIsOpen(!isOpen)}
            >
                <div className={`transform transition-transform text-silver ${isOpen ?
                'rotate-90' : ''}`}>■</div>
                <FolderIcon />
                <span className="font-semibold truncate">{node.name}</span>
            </div>
            {isOpen && node.children && (
                <div>
                    {node.children.map(child => <FileTree key={child.path} node={child}
                    onFileSelect={onFileSelect} activePath={activePath} level={level + 1} />)}
                </div>
            )}
        </div>
    );
};

type CopilotTab = 'Explain' | 'Review' | 'Test' | 'Commit';

const AiCopilot: React.FC<{ file: { content: string, diff?: string } | null }> =
({ file }) => {
    const [activeTab, setActiveTab] = useState<CopilotTab>('Explain');
    const [result, setResult] = useState('');
    const [isLoading, setIsLoading] = useState(false);

    const handleAction = useCallback(async (action: CopilotTab) => {
        if (!file?.content) return;

        setActiveTab(action);
        setIsLoading(true);
        setResult('');

        let stream;
        switch(action) {
            case 'Explain': stream = explainCodeStream(file.content); break;
            case 'Review': stream = reviewCodeStream(file.content); break;
            case 'Test': stream = generateUnitTestsStream(file.content); break;
            case 'Commit':
                if (!file.diff) {
                    setResult('No changes to commit.');
                    setIsLoading(false);
```

```
                    return;
                }
                stream = generateCommitMessageStream(file.diff);
                break;
            default: setIsLoading(false); return;
        }

        try {
            let fullResponse = '';
            for await (const chunk of stream) {
                fullResponse += chunk;
                setResult(fullResponse);
            }
        } catch (e) {
            setResult(`Error: ${e instanceof Error ? e.message : 'Unknown error'}`);
        } finally {
            setIsLoading(false);
        }

    }, [file]);

    useEffect(() => {
        // Trigger default action when file changes
        if (file?.content) {
            handleAction('Explain');
        } else {
            setResult('');
        }
    }, [file, handleAction]);

    return (
        <div className="bg-navy flex flex-col h-full">
            <div className="flex-shrink-0 flex items-center p-2 gap-2 border-b border-
            border">
                <SparklesIcon/>
                <h3 className="font-bold">AI Co-pilot</h3>
            </div>
            <div className="flex-shrink-0 flex border-b border-border">
                {(['Explain', 'Review', 'Test', 'Commit'] as CopilotTab[]).map(tab => (
                    <button key={tab} onClick={() => handleAction(tab)} disabled={!file ||
                    isLoading} className={`px-4 py-2 text-sm font-medium ${activeTab === tab ? 'bg-
                    background text-primary' : 'text-silver hover:bg-surface'}`}>
                        {tab}
                    </button>
                ))}
            </div>
            <div className="flex-grow p-4 overflow-y-auto">
                {isLoading ? <div className="flex justify-center items-center
                h-full"><LoadingSpinner /></div> : <MarkdownRenderer content={result} />}
            </div>
        </div>
    );
}

export const ProjectExplorer: React.FC = () => {
    const { state, dispatch } = useGlobalState();
    const { user, githubUser, selectedRepo, projectFiles } = state;
    const { addNotification } = useNotification();
    const [repos, setRepos] = useState<Repo[]>([]);
    const [isLoading, setIsLoading] = useState<'repos' | 'tree' | 'file' | 'commit'
    | null>(null);
    const [error, setError] = useState('');
```

```
const [activeFile, setActiveFile] = useState<{ path: string; name: string;
originalContent: string; editedContent: string} | null>(null);

const getApiClient = useCallback(async () => {
    if (!user) throw new Error("You must be logged in.");
    const token = await getUserToken(user.uid, 'github_pat');
    if (!token) throw new Error("GitHub token not found. Please connect on the
    Connections page.");
    return initializeOctokit(token);
}, [user]);

useEffect(() => {
    const loadRepos = async () => {
        if (user && githubUser) {
            setIsLoading('repos'); setError('');
            try {
                const octokit = await getApiClient();
                const userRepos = await getRepos(octokit);
                setRepos(userRepos);
            } catch (err) { setError(err instanceof Error ? err.message : 'Failed to load
            repositories'); } finally { setIsLoading(null); }
        } else { setRepos([]); }
    };
    loadRepos();
}, [user, githubUser, getApiClient]);

useEffect(() => {
    const loadTree = async () => {
         if (selectedRepo && user && githubUser) {
            setIsLoading('tree'); setError(''); setActiveFile(null);
            try {
                const octokit = await getApiClient();
                const tree = await getRepoTree(octokit, selectedRepo.owner, selectedRepo.repo);
                dispatch({ type: 'LOAD_PROJECT_FILES', payload: tree });
            } catch (err) { setError(err instanceof Error ? err.message : 'Failed to load
            repository tree'); } finally { setIsLoading(null); }
        }
    };
    loadTree();
}, [selectedRepo, user, githubUser, dispatch, getApiClient]);

const handleFileSelect = async (path: string, name: string) => {
    if (!selectedRepo) return;
    setIsLoading('file');
    try {
        const octokit = await getApiClient();
        const content = await getFileContent(octokit, selectedRepo.owner,
        selectedRepo.repo, path);
        setActiveFile({ path, name, originalContent: content, editedContent: content });
    } catch (err) { setError((err as Error).message); } finally {
    setIsLoading(null); }
};

const handleCommit = async () => {
    if (!activeFile || !selectedRepo || activeFile.originalContent ===
    activeFile.editedContent) return;
    setIsLoading('commit'); setError('');
    try {
        const diff = Diff.createPatch(activeFile.path, activeFile.originalContent,
        activeFile.editedContent);
        const stream = generateCommitMessageStream(diff);
        let commitMessage = ''; for await (const chunk of stream) { commitMessage +=
```

```
        chunk; }

        const finalMessage = window.prompt("Confirm or edit commit message:",
        commitMessage);
        if (!finalMessage) { setIsLoading(null); return; }

        const octokit = await getApiClient();
        await commitFiles(octokit, selectedRepo.owner, selectedRepo.repo, [{ path:
        activeFile.path, content: activeFile.editedContent }], finalMessage);

        addNotification(`Successfully committed to ${selectedRepo.repo}`, 'success');
        setActiveFile(prev => prev ? { ...prev, originalContent: prev.editedContent } :
        null);
    } catch (err) {
        const message = err instanceof Error ? err.message : 'Failed to commit changes';
        setError(message); addNotification(message, 'error');
    } finally { setIsLoading(null); }
};

if (!user || !githubUser) {
    return (
        <div className="h-full flex flex-col items-center justify-center text-center
        text-silver p-4">
            <FolderIcon />
            <h2 className="text-lg font-semibold mt-2">Connect to GitHub</h2>
            <p>Please sign in and connect to GitHub in the "Connections" tab to explore your
            repositories.</p>
        </div>
    );
}

const hasChanges = activeFile ? activeFile.originalContent !==
activeFile.editedContent : false;

return (
    <div className="h-full flex flex-col text-ink ide-grid">
        {/* File Explorer Pane */}
        <aside className="bg-surface border-r border-border p-2 ide-pane">
            <select value={selectedRepo ? `${selectedRepo.owner}/${selectedRepo.repo}` : ''}
            onChange={e => { const [owner, repo] = e.target.value.split('/'); dispatch({
            type: 'SET_SELECTED_REPO', payload: { owner, repo } }); }} className="w-full p-2
            bg-navy border border-border rounded-md text-sm mb-2">
                <option value="" disabled>{isLoading === 'repos' ? 'Loading...' : 'Select a
                repository'}</option>
                {repos.map(r => <option key={r.id} value={r.full_name}>{r.full_name}</option>)}
            </select>
            {error && <p className="text-red-400 text-xs p-2">{error}</p>}
            {isLoading === 'tree' && <div className="flex justify-center
            pt-4"><LoadingSpinner /></div>}
            {projectFiles && <FileTree node={projectFiles} onFileSelect={handleFileSelect}
            activePath={activeFile?.path ?? null} />}
        </aside>

        {/* Code Editor Pane */}
        <main className="bg-navy flex flex-col">
             <div className="flex justify-between items-center p-2 border-b border-border bg-
             surface">
                <span className="text-sm font-semibold">{activeFile?.name || 'No file
                selected'}</span>
                <button onClick={handleCommit} disabled={!hasChanges || isLoading === 'commit'}
                className="btn-primary px-4 py-1 text-sm flex items-center justify-center
                min-w-[100px]">
```

```
                        {isLoading === 'commit' ? <LoadingSpinner/> : 'Commit'}
                    </button>
                </div>
                {isLoading === 'file' ? <div className="flex items-center justify-center
                h-full"><LoadingSpinner /></div> :
                    <textarea value={activeFile?.editedContent ?? 'Select a file to view its
                    content.'} onChange={e => setActiveFile(prev => prev ? { ...prev, editedContent:
                    e.target.value } : null)} disabled={!activeFile} className="w-full h-full p-4
                    text-sm font-mono bg-transparent resize-none focus:outline-none"/>
                }
            </main>

            {/* AI Co-pilot Pane */}
            <aside className="border-l border-border ide-pane">
                <AiCopilot file={activeFile ? { content: activeFile.editedContent, diff:
                Diff.createPatch(activeFile.path, activeFile.originalContent,
                activeFile.editedContent)} : null} />
            </aside>
        </div>
    );
};

// ===== SvgPathEditor_36.tsx =====

import React, { useState, useRef } from 'react';
import { CodeBracketSquareIcon, ArrowDownTrayIcon } from '../icons.tsx';
import { downloadFile } from '../../services/fileUtils.ts';

const parsePath = (d: string) => {
    const commands = d.match(/[a-df-z][^a-df-z]*/ig) || [];
    return commands.map((cmdStr, i) => {
        const command = cmdStr[0];
        const args = cmdStr.slice(1).trim().split(/[\s,]+/).map(parseFloat).filter(n =>
        !isNaN(n));
        const points = [];
        for (let j = 0; j < args.length; j += 2) {
            points.push({ x: args[j], y: args[j + 1] });
        }
        return { id: i, command, points };
    });
};

const buildPath = (parsed: any[]) => {
    return parsed.map(cmd => `${cmd.command} ${cmd.points.map((p:any) => `${p.x}
    ${p.y}`).join(' ')}`).join(' ');
};

export const SvgPathEditor: React.FC = () => {
    const [pathData, setPathData] = useState('');
    const svgRef = useRef<SVGSVGElement>(null);
    const [draggingPoint, setDraggingPoint] = useState<any>(null);
    const parsedPath = parsePath(pathData);

    const handleMouseDown = (e: React.MouseEvent, cmdIndex: number, pointIndex:
    number) => {
        e.stopPropagation();
        setDraggingPoint({ cmdIndex, pointIndex });
    };

    const handleMouseMove = (e: React.MouseEvent) => {
        if (!draggingPoint || !svgRef.current) return;
        const pt = new DOMPoint(e.clientX, e.clientY);
```

```
        const svgPoint = pt.matrixTransform(svgRef.current.getScreenCTM()?.inverse());

        const newParsedPath = parsedPath.map((cmd, cIdx) => {
            if (cIdx === draggingPoint.cmdIndex) {
                const newPoints = cmd.points.map((p, pIdx) => {
                    if (pIdx === draggingPoint.pointIndex) {
                        return { x: Math.round(svgPoint.x), y: Math.round(svgPoint.y) };
                    }
                    return p;
                });
                return { ...cmd, points: newPoints };
            }
            return cmd;
        });
        setPathData(buildPath(newParsedPath));
    };

    const handleMouseUp = () => setDraggingPoint(null);

    const handleAddPoint = (e: React.MouseEvent) => {
        if (!svgRef.current) return;
        const pt = new DOMPoint(e.clientX, e.clientY);
        const svgPoint = pt.matrixTransform(svgRef.current.getScreenCTM()?.inverse());
        const newPathData = `${pathData} L ${Math.round(svgPoint.x)}
        ${Math.round(svgPoint.y)}`;
        setPathData(newPathData);
    };

    const handleDownload = () => {
        const svgContent = `<svg viewBox="0 0 400 160"
        xmlns="http://www.w3.org/2000/svg">
  <path d="${pathData}" stroke="black" fill="transparent" stroke-width="2"/>
</svg>`;
        downloadFile(svgContent, 'path.svg', 'image/svg+xml');
    };

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
            <header className="mb-6"><h1 className="text-3xl font-bold flex items-
            center"><CodeBracketSquareIcon /><span className="ml-3">SVG Path
            Editor</span></h1><p className="text-text-secondary mt-1">Visually create and
            manipulate SVG path data by dragging points.</p></header>
            <div className="flex-grow grid grid-cols-1 lg:grid-cols-2 gap-6 h-full overflow-
            hidden">
                <div className="flex flex-col h-full overflow-y-auto">
                    <div className="flex justify-between items-center mb-2">
                        <label htmlFor="path-input" className="text-sm font-medium text-text-
                        secondary">Path Data (d attribute)</label>
                         <button onClick={handleDownload} className="flex items-center gap-1 px-3 py-1
                         bg-gray-100 text-xs rounded-md hover:bg-gray-200">
                            <ArrowDownTrayIcon className="w-4 h-4"/> Download SVG
                        </button>
                    </div>
                    <textarea id="path-input" value={pathData} onChange={(e) =>
                    setPathData(e.target.value)} className="h-24 p-4 bg-surface border border-border
                    rounded-md resize-y font-mono text-sm text-primary" />
                     <div className="flex-grow mt-4 p-4 bg-surface border-2 border-dashed border-
                     border rounded-md overflow-hidden flex items-center justify-center
                     min-h-[200px]">
                        <svg ref={svgRef} viewBox="0 0 400 160" className="w-full h-full cursor-
                        crosshair" onMouseMove={handleMouseMove} onMouseUp={handleMouseUp}
                        onMouseLeave={handleMouseUp} onDoubleClick={handleAddPoint}>
```

```
                        <rect width="400" height="160" fill="var(--color-background)" />
                         <path d={pathData} stroke="var(--color-primary)" fill="transparent"
                         strokeWidth="2" />
                         {parsedPath.flatMap((cmd, cmdIndex) =>
                             cmd.points.map((p, pointIndex) => (
                                 <circle
                                     key={`${cmd.id}-${pointIndex}`}
                                     cx={p.x}
                                     cy={p.y}
                                     r="5"
                                     fill={cmd.command.toLowerCase() === 'c' || cmd.command.toLowerCase() === 'q' ||
                                     cmd.command.toLowerCase() === 's' || cmd.command.toLowerCase() === 't' ?
                                     '#fde047' : '#f87171'}
                                     stroke="var(--color-surface)"
                                     strokeWidth="2"
                                     className="cursor-move hover:stroke-primary"
                                     onMouseDown={(e) => handleMouseDown(e, cmdIndex, pointIndex)}
                                 />
                             ))
                         )}
                     </svg>
                 </div>
                 <p className="text-xs text-center text-text-secondary mt-2">Double-click on the
                 canvas to add a new point.</p>
             </div>
             <div className="flex flex-col h-full">
                 <label className="text-sm font-medium text-text-secondary mb-2">Parsed
                 Commands</label>
                 <div className="flex-grow p-2 bg-background border border-border rounded-md
                 overflow-y-auto font-mono text-xs space-y-2">
                     {parsedPath.map(cmd => (
                         <div key={cmd.id} className="p-2 bg-surface rounded">
                             <span className="font-bold text-amber-600">{cmd.command}</span>
                             <span className="text-text-secondary"> {cmd.points.map(p =>
                             `(${p.x},${p.y})`).join(' ')}</span>
                         </div>
                     ))}
                 </div>
             </div>
         </div>
     </div>
 );
};

// ===== WorkerThreadDebugger_36.tsx =====

import React, { useState, useCallback, useEffect } from 'react';
import { BugAntIcon, ArrowDownTrayIcon } from '../icons.tsx';
import { analyzeConcurrencyStream } from '../../services/index.ts';
import { LoadingSpinner, MarkdownRenderer } from '../shared/index.tsx';
import { downloadFile } from '../../services/fileUtils.ts';

export const WorkerThreadDebugger: React.FC<{ codeInput?: string }> = ({
codeInput: initialCode }) => {
    const [codeInput, setCodeInput] = useState(initialCode || '');
    const [analysis, setAnalysis] = useState('');
    const [isLoading, setIsLoading] = useState(false);
    const [error, setError] = useState('');

    const handleAnalyze = useCallback(async (codeToAnalyze: string) => {
        if (!codeToAnalyze.trim()) {
            setError('Please paste some code to analyze.');
```

```
        return;
    }
    setIsLoading(true);
    setError('');
    setAnalysis('');
    try {
        const stream = analyzeConcurrencyStream(codeToAnalyze);
        let fullResponse = '';
        for await (const chunk of stream) {
            fullResponse += chunk;
            setAnalysis(fullResponse);
        }
    } catch (err) {
        const errorMessage = err instanceof Error ? err.message : 'An unknown error
        occurred.';
        setError(`Failed to analyze code: ${errorMessage}`);
    } finally {
        setIsLoading(false);
    }
}, []);

useEffect(() => {
    if (initialCode) {
        setCodeInput(initialCode);
        handleAnalyze(initialCode);
    }
}, [initialCode, handleAnalyze]);

return (
    <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
        <header className="mb-6">
            <h1 className="text-3xl font-bold flex items-center">
                <BugAntIcon />
                <span className="ml-3">AI Concurrency Analyzer</span>
            </h1>
            <p className="text-text-secondary mt-1">Analyze JavaScript code for potential
            Web Worker concurrency issues.</p>
        </header>
        <div className="flex-grow flex flex-col gap-4 min-h-0">
            <div className="flex flex-col flex-1 min-h-0">
                <label htmlFor="code-input" className="text-sm font-medium text-text-secondary
                mb-2">JavaScript Code</label>
                <textarea
                    id="code-input"
                    value={codeInput}
                    onChange={(e) => setCodeInput(e.target.value)}
                    placeholder="Paste your worker-related JS code here..."
                    className="flex-grow p-4 bg-surface border border-border rounded-md resize-none
                    font-mono text-sm"
                />
            </div>
             <div className="flex-shrink-0">
                <button
                    onClick={() => handleAnalyze(codeInput)}
                    disabled={isLoading}
                    className="btn-primary w-full max-w-xs mx-auto flex items-center justify-center
                    px-6 py-3"
                >
                    {isLoading ? <LoadingSpinner /> : 'Analyze Code'}
                </button>
            </div>
            <div className="flex flex-col flex-1 min-h-0">
```

```
                    <div className="flex justify-between items-center mb-2">
                        <label className="text-sm font-medium text-text-secondary">AI Analysis</label>
                        {analysis && !isLoading && (
                             <button onClick={() => downloadFile(analysis, 'analysis.md', 'text/markdown')}
                             className="flex items-center gap-1 px-3 py-1 bg-gray-100 text-xs rounded-md
                             hover:bg-gray-200">
                                <ArrowDownTrayIcon className="w-4 h-4"/> Download
                            </button>
                        )}
                    </div>
                    <div className="flex-grow p-4 bg-background border border-border rounded-md
                    overflow-y-auto">
                        {isLoading && <div className="flex items-center justify-center
                        h-full"><LoadingSpinner /></div>}
                        {error && <p className="text-red-500">{error}</p>}
                        {analysis && !isLoading && <MarkdownRenderer content={analysis} />}
                        {!isLoading && !analysis && !error && <div className="text-text-secondary h-full
                        flex items-center justify-center">Analysis will appear here.</div>}
                    </div>
                </div>
            </div>
        </div>
    );
};

// ===== PromptCraftPad_36.tsx =====

import React, { useState, useEffect, useMemo } from 'react';
import { SparklesIcon } from '../icons.tsx';
import { useLocalStorage } from '../../hooks/useLocalStorage.ts';

interface Prompt {
    id: number;
    name: string;
    text: string;
}

export const PromptCraftPad: React.FC = () => {
    const [prompts, setPrompts] = useLocalStorage<Prompt[]>('devcore_prompts', []);
    const [activePrompt, setActivePrompt] = useState<Prompt | null>(prompts[0] ||
    null);
    const [editingId, setEditingId] = useState<number | null>(null);
    const [tempName, setTempName] = useState('');
    const [variables, setVariables] = useState<Record<string, string>>({});

    const variableNames = useMemo(() => {
        if (!activePrompt) return [];
        return [...activePrompt.text.matchAll(/\{(\w+)\}/g)].map(match => match[1]);
    }, [activePrompt]);

    const renderedPrompt = useMemo(() => {
        if (!activePrompt) return '';
        return variableNames.reduce((acc, varName) => {
            return acc.replace(new RegExp(`\\{${varName}\\}`, 'g'), variables[varName] ||
            `{${varName}}`);
        }, activePrompt.text);
    }, [activePrompt, variables, variableNames]);

    useEffect(() => {
        if(!activePrompt && prompts.length > 0) setActivePrompt(prompts[0]);
        if (activePrompt) setActivePrompt(prompts.find((p: Prompt) => p.id ===
        activePrompt.id) || null);
```

```
}, [prompts, activePrompt]);

const handleTextChange = (e: React.ChangeEvent<HTMLTextAreaElement>) => {
    if (!activePrompt) return;
    const updatedPrompt = { ...activePrompt, text: e.target.value };
    setPrompts(prompts.map((p: Prompt) => p.id === updatedPrompt.id ? updatedPrompt
    : p));
};

const handleNameUpdate = (id: number, newName: string) => {
    setPrompts(prompts.map((p: Prompt) => p.id === id ? {...p, name: newName} : p));
    setEditingId(null);
};

const handleAddNew = () => {
    const newPrompt = { id: Date.now(), name: 'New Untitled Prompt', text: '' };
    setPrompts([...prompts, newPrompt]);
    setActivePrompt(newPrompt);
};

const handleDelete = (id: number) => {
    setPrompts(prompts.filter((p: Prompt) => p.id !== id));
    if(activePrompt?.id === id) setActivePrompt(prompts.length > 1 ? prompts[0] :
    null);
}

return (
    <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
        <header className="mb-6"><h1 className="text-3xl font-bold flex items-
        center"><SparklesIcon /><span className="ml-3">Prompt Craft Pad</span></h1><p
        className="text-text-secondary mt-1">Create, save, and manage your favorite AI
        prompts.</p></header>
        <div className="flex-grow flex gap-6 min-h-0">
            <aside className="w-1/3 bg-surface border border-border p-4 rounded-lg flex
            flex-col">
                <h3 className="font-bold mb-2">My Prompts</h3>
                <ul className="space-y-2 flex-grow overflow-y-auto">{prompts.map((p: Prompt) =>
                (<li key={p.id} className="group flex items-center justify-between"><div
                className={`w-full text-left rounded-md ${activePrompt?.id === p.id ? 'bg-
                primary/10' : ''}`}><button onClick={() => setActivePrompt(p)} onDoubleClick={()
                => {setEditingId(p.id); setTempName(p.name);}} className={`w-full text-left px-3
                py-2 ${activePrompt?.id === p.id ? 'text-primary' : 'hover:bg-gray-100'}`}>
                {editingId === p.id ? <input autoFocus value={tempName} onChange={e =>
                setTempName(e.target.value)} onBlur={() => handleNameUpdate(p.id, tempName)}
                onKeyDown={e => e.key === 'Enter' && handleNameUpdate(p.id, tempName)}
                className="bg-gray-100 text-text-primary w-full"/> : p.name}
                </button></div><button onClick={() => handleDelete(p.id)} className="ml-2 p-1
                text-text-secondary hover:text-red-500 opacity-0 group-
                hover:opacity-100">&times;</button></li>))}</ul>
                <div className="mt-4 pt-4 border-t border-border"><button onClick={handleAddNew}
                className="btn-primary w-full text-sm py-2">Add New Prompt</button></div>
            </aside>
            <main className="w-2/3 flex flex-col gap-4">
                {activePrompt ? (<>
                    <textarea value={activePrompt.text} onChange={handleTextChange} className="flex-
                    grow p-4 bg-surface border border-border rounded-md resize-none font-mono text-
                    sm focus:ring-2 focus:ring-primary focus:outline-none"/>
                    {variableNames.length > 0 && <div className="flex-shrink-0 bg-surface border
                    border-border p-4 rounded-lg"><h4 className="font-bold mb-2">Test
                    Variables</h4><div className="grid grid-cols-2 gap-2">{variableNames.map(v =>
                    (<div key={v}><label className="text-xs">{v}</label><input type="text"
                    value={variables[v] || ''} onChange={e => setVariables({...variables, [v]:
```

```
                        e.target.value})} className="w-full bg-background border border-border px-2 py-1
                        rounded text-sm"/></div>))}</div><h4 className="font-bold mt-4 mb-2">Live
                        Preview</h4><p className="text-sm p-2 bg-background rounded border border-
                        border">{renderedPrompt}</p></div>}
                    </>) : (<div className="flex-grow flex items-center justify-center bg-background
                    rounded-lg text-text-secondary border border-border">Select a prompt or create a
                    new one.</div>)}
                </main>
            </div>
        </div>
    );
};

// ===== AiCodeMigrator_36.tsx =====



import React, { useState, useCallback, useEffect } from 'react';
import { migrateCodeStream } from '../../services/index.ts';
import { ArrowPathIcon } from '../icons.tsx';
import { LoadingSpinner } from '../shared/index.tsx';
import { MarkdownRenderer } from '../shared/index.tsx';

const languages = ['SASS', 'CSS', 'JavaScript', 'TypeScript', 'Python', 'Go',
'React', 'Vue', 'Angular', 'Tailwind CSS'];

export const AiCodeMigrator: React.FC<{ inputCode?: string, fromLang?: string,
toLang?: string }> = ({ inputCode: initialCode, fromLang: initialFrom, toLang:
initialTo }) => {
    const [inputCode, setInputCode] = useState<string>(initialCode || '');
    const [outputCode, setOutputCode] = useState<string>('');
    const [fromLang, setFromLang] = useState(initialFrom || 'SASS');
    const [toLang, setToLang] = useState(initialTo || 'CSS');
    const [isLoading, setIsLoading] = useState<boolean>(false);
    const [error, setError] = useState<string>('');

    const handleMigrate = useCallback(async (code: string, from: string, to: string)
    => {
        if (!code.trim()) {
            setError('Please enter some code to migrate.');
            return;
        }
        setIsLoading(true);
        setError('');
        setOutputCode('');
        try {
            const stream = migrateCodeStream(code, from, to);
            let fullResponse = '';
            for await (const chunk of stream) {
                fullResponse += chunk;
                setOutputCode(fullResponse);
            }
        } catch (err) {
            const errorMessage = err instanceof Error ? err.message : 'An unknown error
            occurred.';
            setError(`Failed to migrate code: ${errorMessage}`);
        } finally {
            setIsLoading(false);
        }
    }, []);

    useEffect(() => {
```

```
    if (initialCode && initialFrom && initialTo) {
        setInputCode(initialCode);
        setFromLang(initialFrom);
        setToLang(initialTo);
        handleMigrate(initialCode, initialFrom, initialTo);
    }
}, [initialCode, initialFrom, initialTo, handleMigrate]);

const LanguageSelector: React.FC<{ value: string, onChange: (val: string) =>
void }> = ({ value, onChange }) => (
    <select value={value} onChange={e => onChange(e.target.value)} className="w-full
    px-3 py-2 rounded-md bg-surface border border-border">
        {languages.map(lang => <option key={lang} value={lang}>{lang}</option>)}
    </select>
);

return (
    <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
        <header className="mb-6">
            <h1 className="text-3xl font-bold flex items-center"><ArrowPathIcon /><span
            className="ml-3">AI Code Migrator</span></h1>
            <p className="text-text-secondary mt-1">Translate code between languages,
            frameworks, and syntax styles.</p>
        </header>
        <div className="flex-grow flex flex-col min-h-0">
            <div className="grid grid-cols-1 lg:grid-cols-2 gap-6 flex-grow min-h-0">
                <div className="flex flex-col h-full">
                    <div className="mb-2">
                        <label className="text-sm font-medium text-text-secondary">From:</label>
                        <LanguageSelector value={fromLang} onChange={setFromLang} />
                    </div>
                    <textarea
                        value={inputCode}
                        onChange={(e) => setInputCode(e.target.value)}
                        placeholder="Paste your source code here..."
                        className="flex-grow p-4 bg-surface border border-border rounded-md resize-none
                        font-mono text-sm"
                    />
                </div>
                <div className="flex flex-col h-full">
                    <div className="mb-2">
                        <label className="text-sm font-medium text-text-secondary">To:</label>
                        <LanguageSelector value={toLang} onChange={setToLang} />
                    </div>
                    <div className="flex-grow p-1 bg-background border border-border rounded-md
                    overflow-y-auto">
                       {isLoading && <div className="flex items-center justify-center
                       h-full"><LoadingSpinner /></div>}
                        {error && <p className="p-4 text-red-500">{error}</p>}
                        {outputCode && !isLoading && <MarkdownRenderer content={outputCode} />}
                        {!isLoading && !outputCode && !error && <div className="text-text-secondary
                        h-full flex items-center justify-center">Migrated code will appear here.</div>}
                    </div>
                </div>
            </div>
             <button
                onClick={() => handleMigrate(inputCode, fromLang, toLang)}
                disabled={isLoading}
                className="btn-primary mt-4 w-full max-w-sm mx-auto flex items-center justify-
                center px-6 py-3"
            >
                {isLoading ? <LoadingSpinner /> : 'Migrate Code'}
```

```
                </button>
            </div>
        </div>
    );
};

// ===== AiImageGenerator_34.tsx =====

import React, { useState, useCallback, useRef } from 'react';
import { generateImage, generateImageFromImageAndText } from
'../../services/aiService.ts';
import { fileToBase64, blobToDataURL } from '../../services/fileUtils.ts';
import { ImageGeneratorIcon, SparklesIcon, ArrowDownTrayIcon, XMarkIcon } from
'../icons.tsx';
import { LoadingSpinner } from '../shared/index.tsx';

const surprisePrompts = [
    'A majestic lion wearing a crown, painted in the style of Van Gogh.',
    'A futuristic cityscape on another planet with two moons in the sky.',
    'A cozy, magical library inside a giant tree.',
    'A surreal image of a ship sailing on a sea of clouds.',
    'An astronaut riding a space-themed bicycle on the moon.',
];

interface UploadedImage {
    base64: string;
    dataUrl: string;
    mimeType: string;
}

export const AiImageGenerator: React.FC = () => {
    const [prompt, setPrompt] = useState<string>('');
    const [uploadedImage, setUploadedImage] = useState<UploadedImage | null>(null);
    const [generatedImageUrl, setGeneratedImageUrl] = useState<string | null>(null);
    const [isLoading, setIsLoading] = useState<boolean>(false);
    const [error, setError] = useState<string>('');
    const fileInputRef = useRef<HTMLInputElement>(null);

    const handleGenerate = useCallback(async () => {
        if (!prompt.trim()) {
            setError('Please enter a prompt to generate an image.');
            return;
        }
        setIsLoading(true);
        setError('');
        setGeneratedImageUrl(null);
        try {
            let resultUrl: string;
            if (uploadedImage) {
                resultUrl = await generateImageFromImageAndText(prompt, uploadedImage.base64,
                uploadedImage.mimeType);
            } else {
                resultUrl = await generateImage(prompt);
            }
            setGeneratedImageUrl(resultUrl);
        } catch (err) {
            const errorMessage = err instanceof Error ? err.message : 'An unknown error
            occurred.';
            setError(`Failed to generate image: ${errorMessage}`);
        } finally {
            setIsLoading(false);
        }
```

```
}, [prompt, uploadedImage]);

const handleSurpriseMe = () => {
    const randomPrompt = surprisePrompts[Math.floor(Math.random() *
    surprisePrompts.length)];
    setPrompt(randomPrompt);
};

const processImageBlob = async (blob: Blob) => {
    try {
        const [dataUrl, base64] = await Promise.all([
            blobToDataURL(blob),
            fileToBase64(blob as File)
        ]);
        setUploadedImage({ dataUrl, base64, mimeType: blob.type });
    } catch (e) {
        setError('Could not process the image.');
    }
};

const handlePaste = useCallback(async (event: React.ClipboardEvent) => {
    const items = event.clipboardData.items;
    for (const item of items) {
        if (item.type.indexOf('image') !== -1) {
            const blob = item.getAsFile();
            if (blob) {
                await processImageBlob(blob);
                return;
            }
        }
    }
}, []);

const handleFileChange = async (event: React.ChangeEvent<HTMLInputElement>) => {
    const file = event.target.files?.[0];
    if (file) {
        await processImageBlob(file);
    }
};

const handleDownload = () => {
    if (!generatedImageUrl) return;
    const link = document.createElement('a');
    link.href = generatedImageUrl;
    link.download = `${prompt.slice(0, 30).replace(/\s/g, '_')}.png`;
    document.body.appendChild(link);
    link.click();
    document.body.removeChild(link);
}

return (
    <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
        <header className="mb-6">
            <h1 className="text-3xl font-bold flex items-center">
                <ImageGeneratorIcon />
                <span className="ml-3">AI Image Generator</span>
            </h1>
            <p className="text-text-secondary mt-1">Generate images from text, or provide an
            image for inspiration.</p>
        </header>

        <div className="flex-grow grid grid-cols-1 lg:grid-cols-2 gap-6 min-h-0">
```

```
{/* Left Column: Inputs */}
<div className="flex flex-col gap-4">
    <div>
        <label htmlFor="prompt-input" className="text-sm font-medium text-text-
        secondary">Your Prompt</label>
        <textarea
            id="prompt-input"
            value={prompt}
            onChange={(e) => setPrompt(e.target.value)}
            placeholder="e.g., A cute cat wearing a wizard hat"
            className="w-full p-3 mt-1 rounded-md bg-surface border border-border
            focus:ring-2 focus:ring-primary focus:outline-none resize-y"
            rows={3}
        />
    </div>

    <div className="flex flex-col flex-grow min-h-[200px]">
         <label className="text-sm font-medium text-text-secondary mb-1">Inspiration
         Image (Optional)</label>
         <div onPaste={handlePaste} className="relative flex-grow flex flex-col items-
         center justify-center bg-surface p-4 rounded-lg border-2 border-dashed border-
         border focus:outline-none focus:border-primary" tabIndex={0}>
            {uploadedImage ? (
                <>
                    <img src={uploadedImage.dataUrl} alt="Uploaded content" className="max-w-full
                    max-h-full object-contain rounded-md shadow-lg" />
                    <button onClick={() => setUploadedImage(null)} className="absolute top-2 right-2
                    p-1 bg-black/30 text-white rounded-full hover:bg-black/50"><XMarkIcon
                    /></button>
                </>
            ) : (
                <div className="text-center text-text-secondary">
                    <h2 className="text-lg font-bold text-text-primary">Paste an image here</h2>
                    <p className="text-sm">(Cmd/Ctrl + V)</p>
                    <p className="text-xs my-1">or</p>
                    <button onClick={() => fileInputRef.current?.click()} className="text-sm font-
                    semibold text-primary hover:underline">Upload File</button>
                    <input type="file" ref={fileInputRef} onChange={handleFileChange}
                    accept="image/*" className="hidden"/>
                </div>
            )}
         </div>
    </div>

    <div className="flex gap-2">
        <button
            onClick={handleGenerate}
            disabled={isLoading}
            className="btn-primary w-full flex items-center justify-center px-6 py-3"
        >
            {isLoading ? <LoadingSpinner /> : 'Generate Image'}
        </button>
        <button
            onClick={handleSurpriseMe}
            disabled={isLoading}
            className="px-4 py-3 bg-surface border border-border rounded-md hover:bg-
            gray-100 transition-colors"
            title="Surprise Me!"
        >
            <SparklesIcon />
        </button>
    </div>
```

```
                </div>

                {/* Right Column: Output */}
                <div className="flex flex-col h-full">
                    <label className="text-sm font-medium text-text-secondary mb-2">Generated
                    Image</label>
                    <div className="flex-grow flex items-center justify-center bg-background
                    border-2 border-dashed border-border rounded-lg p-4 relative overflow-auto">
                        {isLoading && <LoadingSpinner />}
                        {error && <p className="text-red-500 text-center">{error}</p>}
                        {generatedImageUrl && !isLoading && (
                            <>
                                <img src={generatedImageUrl} alt={prompt || "Generated by AI"} className="max-w-
                                full max-h-full object-contain rounded-md shadow-lg" />
                                <button
                                  onClick={handleDownload}
                                  className="absolute top-4 right-4 p-2 bg-black/30 text-white rounded-full
                                  hover:bg-black/50 backdrop-blur-sm"
                                  title="Download Image"
                                >
                                    <ArrowDownTrayIcon />
                                </button>
                            </>
                        )}
                        {!isLoading && !generatedImageUrl && !error && (
                            <div className="text-center text-text-secondary">
                                <p>Your generated image will appear here.</p>
                            </div>
                        )}
                    </div>
                </div>
            </div>
        </div>
    );
};

// ===== AiUnitTestGenerator_36.tsx =====

import React, { useState, useCallback } from 'react';
import { generateUnitTestsStream, downloadFile } from '../../services/index.ts';
import { BeakerIcon, ArrowDownTrayIcon } from '../icons.tsx';
import { LoadingSpinner } from '../shared/index.tsx';
import { MarkdownRenderer } from '../shared/index.tsx';

export const AiUnitTestGenerator: React.FC = () => {
    const [code, setCode] = useState<string>('');
    const [tests, setTests] = useState<string>('');
    const [isLoading, setIsLoading] = useState<boolean>(false);
    const [error, setError] = useState<string>('');

    const handleGenerate = useCallback(async () => {
        if (!code.trim()) {
            setError('Please enter some code to generate tests for.');
            return;
        }
        setIsLoading(true);
        setError('');
        setTests('');
        try {
            const stream = generateUnitTestsStream(code);
            let fullResponse = '';
            for await (const chunk of stream) {
```

```
            fullResponse += chunk;
            setTests(fullResponse);
        }
    } catch (err) {
        const errorMessage = err instanceof Error ? err.message : 'An unknown error
        occurred.';
        setError(`Failed to generate tests: ${errorMessage}`);
    } finally {
        setIsLoading(false);
    }
}, [code]);

const cleanCodeForDownload = (markdown: string) => {
    return markdown.replace(/^```(?:\w+\n)?/, '').replace(/```$/, '');
}

return (
    <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
        <header className="mb-6">
            <h1 className="text-3xl font-bold flex items-center">
                <BeakerIcon />
                <span className="ml-3">AI Unit Test Generator</span>
            </h1>
            <p className="text-text-secondary mt-1">Provide a function or component and let
            AI write the tests.</p>
        </header>
        <div className="flex-grow flex flex-col gap-4 min-h-0">
            <div className="flex flex-col flex-1 min-h-0">
                <label htmlFor="code-input" className="text-sm font-medium text-text-secondary
                mb-2">Source Code</label>
                <textarea
                    id="code-input"
                    value={code}
                    onChange={(e) => setCode(e.target.value)}
                    placeholder="Paste your source code here..."
                    className="flex-grow p-4 bg-surface border border-border rounded-md resize-none
                    font-mono text-sm focus:ring-2 focus:ring-primary focus:outline-none"
                />
            </div>
            <div className="flex-shrink-0">
                <button
                    onClick={handleGenerate}
                    disabled={isLoading}
                    className="btn-primary w-full max-w-xs mx-auto flex items-center justify-center
                    px-6 py-3"
                >
                    {isLoading ? <LoadingSpinner /> : 'Generate Unit Tests'}
                </button>
            </div>
            <div className="flex flex-col flex-1 min-h-0">
                <div className="flex justify-between items-center mb-2">
                    <label className="text-sm font-medium text-text-secondary">Generated
                    Tests</label>
                    {tests && !isLoading && (
                        <div className="flex items-center gap-2">
                            <button onClick={() =>
                            navigator.clipboard.writeText(cleanCodeForDownload(tests))} className="px-3 py-1
                            bg-gray-100 text-xs rounded-md hover:bg-gray-200">Copy Code</button>
                            <button onClick={() => downloadFile(cleanCodeForDownload(tests), 'tests.tsx',
                            'text/typescript')} className="flex items-center gap-1 px-3 py-1 bg-gray-100
                            text-xs rounded-md hover:bg-gray-200">
                                <ArrowDownTrayIcon className="w-4 h-4" /> Download
```

```
                            </button>
                        </div>
                    )}
                </div>
                <div className="flex-grow p-1 bg-background border border-border rounded-md
                overflow-y-auto">
                    {isLoading && !tests && (
                        <div className="flex items-center justify-center h-full">
                            <LoadingSpinner />
                        </div>
                    )}
                    {error && <p className="p-4 text-red-500">{error}</p>}
                    {tests && <MarkdownRenderer content={tests} />}
                    {!isLoading && !tests && !error && (
                        <div className="text-text-secondary h-full flex items-center justify-center">
                            The generated tests will appear here.
                        </div>
                    )}
                </div>
            </div>
        </div>
    </div>
    );
};

// ===== AsyncCallTreeViewer_36.tsx =====



import React, { useState, useMemo } from 'react';
import { ChartBarIcon } from '../icons.tsx';

interface CallNode {
    name: string;
    duration: number;
    children?: CallNode[];
}

const TreeNode: React.FC<{ node: CallNode, level: number, maxDuration: number }>
= ({ node, level, maxDuration }) => {
    const [isOpen, setIsOpen] = React.useState(true);
    const hasChildren = node.children && node.children.length > 0;

    return (
        <div className="my-1">
            <div
                className="flex items-center p-2 rounded-md hover:bg-gray-100"
                style={{ paddingLeft: `${level * 20 + 8}px` }}
            >
                {hasChildren && (
                    <button onClick={() => setIsOpen(!isOpen)} className={`mr-2 text-text-secondary
                    w-4 h-4 flex-shrink-0 transform transition-transform ${isOpen ? 'rotate-90' :
                    ''}`}>
                        ■
                    </button>
                )}
                 {!hasChildren && <div className="w-6 mr-2 flex-shrink-0" />}
                 <div className="flex-grow flex items-center justify-between gap-4">
                    <span className="truncate">{node.name}</span>
                    <div className="flex items-center gap-2 flex-shrink-0">
                         <div className="w-24 h-4 bg-gray-200 rounded-full overflow-hidden">
                            <div className="h-4 bg-primary" style={{ width: `${(node.duration / maxDuration)
```

```
                            * 100}%` }}/>
                         </div>
                        <span className="text-primary w-16 text-
                        right">{node.duration.toFixed(0)}ms</span>
                    </div>
                </div>
            </div>
            {isOpen && hasChildren && (
                <div>
                    {node.children!.map((child, index) => (
                        <TreeNode key={index} node={child} level={level + 1} maxDuration={maxDuration}
                        />
                    ))}
                </div>
            )}
        </div>
    );
};


export const AsyncCallTreeViewer: React.FC = () => {
    const [jsonInput, setJsonInput] = useState('');
    const [error, setError] = useState('');

    const { treeData, maxDuration } = useMemo(() => {
        if (!jsonInput.trim()) {
            return { treeData: null, maxDuration: 0 };
        }
        try {
            const data: CallNode = JSON.parse(jsonInput);
             let max = 0;
            const findMax = (node: CallNode) => {
                if (node.duration > max) max = node.duration;
                if (node.children) node.children.forEach(findMax);
            };
            findMax(data);
            setError('');
            return { treeData: data, maxDuration: max };
        } catch (e) {
            setError('Invalid JSON format.');
            return { treeData: null, maxDuration: 0 };
        }
    }, [jsonInput]);

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
            <header className="mb-6">
                <h1 className="text-3xl flex items-center">
                    <ChartBarIcon />
                    <span className="ml-3">Async Call Tree Viewer</span>
                </h1>
                <p className="text-text-secondary mt-1">Paste a JSON structure to visualize an
                asynchronous function call tree.</p>
            </header>
            <div className="flex-grow flex flex-col gap-4 min-h-0">
                <div className="flex flex-col h-2/5 min-h-[200px]">
                    <label htmlFor="json-input" className="text-sm font-medium text-text-secondary
                    mb-2">JSON Input</label>
                    <textarea
                        id="json-input"
                        value={jsonInput}
                        onChange={e => setJsonInput(e.target.value)}
```

```
                        className={`flex-grow p-4 bg-surface border ${error ? 'border-red-500' :
                        'border-border'} rounded-md resize-y font-mono text-sm`}
                        spellCheck="false"
                    />
                    {error && <p className="text-red-500 text-xs mt-1">{error}</p>}
                </div>
                <div className="flex flex-col flex-grow min-h-0">
                    <label className="text-sm font-medium text-text-secondary mb-2">Visual
                    Tree</label>
                    <div className="flex-grow bg-surface p-4 rounded-lg text-sm overflow-y-auto
                    border border-border">
                        {treeData ? <TreeNode node={treeData} level={0} maxDuration={maxDuration} /> :
                        <div className="text-text-secondary">{error || 'Enter valid JSON to see the
                        tree.'}</div>}
                    </div>
                </div>
            </div>
        </div>
    );
};

// ===== MetaTagEditor_36.tsx =====

import React, { useState, useMemo } from 'react';
import { CodeBracketSquareIcon } from '../icons.tsx';

interface MetaData {
    title: string;
    description: string;
    image: string;
    url: string;
}

const SocialCardPreview: React.FC<{ meta: MetaData }> = ({ meta }) => (
    <div className="w-full max-w-md mx-auto bg-surface border border-border
    rounded-2xl overflow-hidden shadow-lg">
        <div className="h-52 bg-gray-100 flex items-center justify-center">
            {meta.image ? <img src={meta.image} alt="Preview" className="w-full h-full
            object-cover" onError={(e) => e.currentTarget.style.display='none'}/> : <span
            className="text-text-secondary">Image Preview</span>}
        </div>
        <div className="p-4">
            <p className="text-xs text-text-secondary truncate">{meta.url ? new
            URL(meta.url).hostname : 'example.com'}</p>
            <h3 className="font-bold text-text-primary truncate mt-1">{meta.title || 'Your
            Title Here'}</h3>
            <p className="text-sm text-text-secondary mt-1 line-clamp-2">{meta.description
            || 'A concise description of your content will appear here.'}</p>
        </div>
    </div>
);

export const MetaTagEditor: React.FC = () => {
    const [meta, setMeta] = useState<MetaData>({
        title: '', description: '', image: '', url: ''
    });

    const handleChange = (e: React.ChangeEvent<HTMLInputElement>) => {
        setMeta({ ...meta, [e.target.name]: e.target.value });
    };

    const generatedHtml = useMemo(() => {
```

```
        return `<!-- Primary Meta Tags -->
<title>${meta.title}</title>
<meta name="title" content="${meta.title}" />
<meta name="description" content="${meta.description}" />
<!-- Open Graph / Facebook -->
<meta property="og:type" content="website" />
<meta property="og:url" content="${meta.url}" />
<meta property="og:title" content="${meta.title}" />
<meta property="og:description" content="${meta.description}" />
<meta property="og:image" content="${meta.image}" />
<!-- Twitter -->
<meta property="twitter:card" content="summary_large_image" />
<meta property="twitter:url" content="${meta.url}" />
<meta property="twitter:title" content="${meta.title}" />
<meta property="twitter:description" content="${meta.description}" />
<meta property="twitter:image" content="${meta.image}" />`;
    }, [meta]);

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
            <header className="mb-6"><h1 className="text-3xl font-bold flex items-
            center"><CodeBracketSquareIcon /><span className="ml-3">Meta Tag
            Editor</span></h1><p className="text-text-secondary mt-1">Generate SEO & social
            media meta tags with a live preview.</p></header>
            <div className="flex-grow grid grid-cols-1 lg:grid-cols-2 xl:grid-cols-3 gap-6
            min-h-0">
                <div className="xl:col-span-1 flex flex-col gap-4 bg-surface border border-
                border p-6 rounded-lg overflow-y-auto">
                    <h3 className="text-xl font-bold">Metadata</h3>
                    <div><label className="block text-sm">Title</label><input type="text"
                    name="title" value={meta.title} onChange={handleChange} className="w-full mt-1
                    p-2 rounded bg-background border border-border"/></div>
                    <div><label className="block text-sm">Description</label><input type="text"
                    name="description" value={meta.description} onChange={handleChange}
                    className="w-full mt-1 p-2 rounded bg-background border border-border"/></div>
                    <div><label className="block text-sm">Canonical URL</label><input type="text"
                    name="url" value={meta.url} onChange={handleChange} className="w-full mt-1 p-2
                    rounded bg-background border border-border"/></div>
                    <div><label className="block text-sm">Social Image URL</label><input type="text"
                    name="image" value={meta.image} onChange={handleChange} className="w-full mt-1
                    p-2 rounded bg-background border border-border"/></div>
                </div>
                <div className="xl:col-span-1 flex flex-col">
                     <label className="text-sm font-medium text-text-secondary mb-2">Generated
                     HTML</label>
                     <div className="relative flex-grow"><pre className="w-full h-full bg-background
                     p-4 rounded-md text-primary text-sm overflow-auto">{generatedHtml}</pre><button
                     onClick={() => navigator.clipboard.writeText(generatedHtml)} className="absolute
                     top-2 right-2 px-2 py-1 bg-gray-100 hover:bg-gray-200 rounded-md text-
                     xs">Copy</button></div>
                </div>
                 <div className="hidden xl:flex flex-col items-center justify-center">
                    <label className="text-sm font-medium text-text-secondary mb-2">Live
                    Preview</label>
                    <SocialCardPreview meta={meta} />
                </div>
            </div>
        </div>
    );
};

// ===== SchemaDesigner_36.tsx =====
```

```
import React, { useState, useRef } from 'react';
import { MapIcon, ArrowDownTrayIcon } from '../icons.tsx';
import { downloadFile } from '../../services/fileUtils.ts';

interface Column { id: number; name: string; type: string; }
interface Table { id: number; name: string; columns: Column[]; x: number; y:
number; }

const exportToSQL = (tables: Table[]) => {
    return tables.map(table => {
        const columnsSQL = table.columns.map(col => `  "${col.name}"
        ${col.type.toUpperCase()}`).join(',\n');
        return `CREATE TABLE "${table.name}" (\n${columnsSQL}\n);`;
    }).join('\n\n');
};

export const SchemaDesigner: React.FC = () => {
    const [tables, setTables] = useState<Table[]>([]);
    const [dragging, setDragging] = useState<{ id: number; offsetX: number; offsetY:
    number } | null>(null);
    const canvasRef = useRef<HTMLDivElement>(null);

    const onMouseDown = (e: React.MouseEvent<HTMLDivElement>, id: number) => {
        const tableElement = e.currentTarget;
        const rect = tableElement.getBoundingClientRect();
        setDragging({ id, offsetX: e.clientX - rect.left, offsetY: e.clientY - rect.top
        });
    };

    const onMouseMove = (e: React.MouseEvent<HTMLDivElement>) => {
        if (!dragging || !canvasRef.current) return;
        const canvasRect = canvasRef.current.getBoundingClientRect();
        setTables(tables.map(t => t.id === dragging.id ? { ...t, x: e.clientX -
        dragging.offsetX - canvasRect.left + canvasRef.current.scrollLeft, y: e.clientY
        - dragging.offsetY - canvasRect.top + canvasRef.current.scrollTop } : t));
    };

    const onMouseUp = () => setDragging(null);

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
            <header className="mb-6"><h1 className="text-3xl font-bold flex items-
            center"><MapIcon /><span className="ml-3">Schema Designer</span></h1><p
            className="text-text-secondary mt-1">Visually design your database schema with
            drag-and-drop.</p></header>
            <div className="flex-grow flex gap-6 min-h-0">
                <main ref={canvasRef} className="flex-grow relative bg-background rounded-lg
                border-2 border-dashed border-border overflow-auto" onMouseMove={onMouseMove}
                onMouseUp={onMouseUp} onMouseLeave={onMouseUp}>
                    {tables.map(table => (
                        <div key={table.id} className={`absolute w-64 bg-surface rounded-lg shadow-xl
                        border cursor-grab active:cursor-grabbing ${dragging?.id === table.id ? 'border-
                        primary' : 'border-border'}`} style={{ top: table.y, left: table.x }}
                        onMouseDown={e => onMouseDown(e, table.id)}>
                            <h3 className="font-bold text-primary text-lg p-2 bg-gray-50 rounded-t-lg
                            border-b border-border">{table.name}</h3>
                            <div className="p-2 space-y-1 font-mono text-xs">
                                {table.columns.map(col => (<div key={col.id} className="flex justify-between
                                items-center"><span className="text-text-primary">{col.name}</span><span
                                className="text-text-secondary">{col.type}</span></div>))}
                            </div>
                        </div>
```

```
                    ))}
                </main>
                <aside className="w-80 flex-shrink-0 flex flex-col gap-4">
                    <div className="flex flex-col gap-2">
                         <button onClick={() => downloadFile(JSON.stringify(tables, null, 2),
                         'schema.json', 'application/json')} className="flex-1 text-sm py-2 bg-gray-100
                         border border-border rounded-md flex items-center justify-center gap-2 hover:bg-
                         gray-200">
                            <ArrowDownTrayIcon className="w-4 h-4"/> Download JSON
                        </button>
                         <button onClick={() => downloadFile(exportToSQL(tables), 'schema.sql',
                         'application/sql')} className="btn-primary flex-1 text-sm py-2 flex items-center
                         justify-center gap-2">
                            <ArrowDownTrayIcon className="w-4 h-4"/> Download SQL
                         </button>
                    </div>
                    <div className="flex-grow bg-surface border border-border p-4 rounded-lg
                    overflow-y-auto">
                        <h3 className="font-bold mb-2">Editor</h3>
                        <p className="text-xs text-text-secondary">Schema editing coming soon!</p>
                    </div>
                </aside>
            </div>
        </div>
    );
};

// ===== JsonTreeNavigator_36.tsx =====

import React, { useState } from 'react';
import { FileCodeIcon } from '../icons.tsx';

interface JsonNodeProps {
    data: any;
    nodeKey: string;
    isRoot?: boolean;
}

const JsonNode: React.FC<JsonNodeProps> = ({ data, nodeKey, isRoot = false }) =>
{
    const [isOpen, setIsOpen] = useState(isRoot);
    const isObject = typeof data === 'object' && data !== null;

    const toggleOpen = () => setIsOpen(!isOpen);

    if (!isObject) {
        return (
            <div className="ml-4 pl-4 border-l border-border">
                <span className="text-purple-700">{nodeKey}: </span>
                <span className={typeof data === 'string' ? 'text-green-700' : 'text-
                orange-700'}>
                    {typeof data === 'string' ? `"${data}"` : String(data)}
                </span>
            </div>
        );
    }

    const entries = Object.entries(data);
    const bracket = Array.isArray(data) ? '[]' : '{}';

    return (
        <div className={`ml-4 ${!isRoot ? 'pl-4 border-l border-border' : ''}`}>
```

```
            <button onClick={toggleOpen} className="flex items-center cursor-pointer
            hover:bg-gray-100 rounded px-1">
                <span className={`transform transition-transform ${isOpen ? 'rotate-90' :
                'rotate-0'}`}>■</span>
                <span className="ml-1 text-purple-700">{nodeKey}:</span>
                <span className="ml-2 text-text-secondary">{bracket[0]}</span>
                {!isOpen && <span className="text-text-secondary">...{bracket[1]}</span>}
            </button>
            {isOpen && (
                <div>
                    {entries.map(([key, value]) => (
                        <JsonNode key={key} nodeKey={key} data={value} />
                    ))}
                    <div className="text-text-secondary ml-4">{bracket[1]}</div>
                </div>
            )}
        </div>
    );
};

export const JsonTreeNavigator: React.FC<{ initialData?: object }> = ({
initialData }) => {
    const [jsonInput, setJsonInput] = useState(initialData ?
    JSON.stringify(initialData, null, 2) : '');
    const [parsedData, setParsedData] = useState<any>(() => {
        try {
            return JSON.parse(jsonInput);
        } catch {
            return null;
        }
    });
    const [error, setError] = useState('');

    const parseJson = (input: string) => {
        if (!input.trim()) {
            setParsedData(null);
            setError('');
            return;
        }
        try {
            const parsed = JSON.parse(input);
            setParsedData(parsed);
            setError('');
        } catch (e) {
            if (e instanceof Error) setError(e.message);
            setParsedData(null);
        }
    };

    const handleInputChange = (e: React.ChangeEvent<HTMLTextAreaElement>) => {
        setJsonInput(e.target.value);
        parseJson(e.target.value);
    }

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
            <header className="mb-6">
                <h1 className="text-3xl font-bold flex items-center">
                    <FileCodeIcon />
                    <span className="ml-3">JSON Tree Navigator</span>
                </h1>
                <p className="text-text-secondary mt-1">Paste your JSON data to visualize it as
```

```
                a collapsible tree.</p>
            </header>
            <div className="flex-grow flex flex-col gap-4 min-h-0">
                <div className="flex flex-col h-2/5 min-h-[200px]">
                    <label htmlFor="json-input" className="text-sm font-medium text-text-secondary
                    mb-2">JSON Input</label>
                    <textarea
                        id="json-input"
                        value={jsonInput}
                        onChange={handleInputChange}
                        className={`flex-grow p-4 bg-surface border ${error ? 'border-red-500' :
                        'border-border'} rounded-md resize-y font-mono text-sm focus:ring-2 focus:ring-
                        primary focus:outline-none`}
                    />
                    {error && <p className="text-red-500 text-xs mt-1">{error}</p>}
                </div>
                 <div className="flex flex-col flex-grow min-h-0">
                    <label className="text-sm font-medium text-text-secondary mb-2">Tree
                    View</label>
                    <div className="flex-grow p-4 bg-surface border border-border rounded-md
                    overflow-y-auto font-mono text-sm">
                        {parsedData ? <JsonNode data={parsedData} nodeKey="root" isRoot /> : <div
                        className="text-text-secondary">Enter valid JSON to view</div>}
                    </div>
                </div>
            </div>
        </div>
    );
};

// ===== AiStyleTransfer_36.tsx =====

import React, { useState, useCallback } from 'react';
import { transferCodeStyleStream } from '../../services/index.ts';
import { SparklesIcon } from '../icons.tsx';
import { LoadingSpinner } from '../shared/index.tsx';
import { MarkdownRenderer } from '../shared/index.tsx';

export const AiStyleTransfer: React.FC = () => {
    const [inputCode, setInputCode] = useState<string>('');
    const [styleGuide, setStyleGuide] = useState<string>('');
    const [outputCode, setOutputCode] = useState<string>('');
    const [isLoading, setIsLoading] = useState<boolean>(false);
    const [error, setError] = useState<string>('');

    const handleGenerate = useCallback(async () => {
        if (!inputCode.trim() || !styleGuide.trim()) {
            setError('Please provide both code and a style guide.');
            return;
        }
        setIsLoading(true);
        setError('');
        setOutputCode('');
        try {
            const stream = transferCodeStyleStream({ code: inputCode, styleGuide });
            let fullResponse = '';
            for await (const chunk of stream) {
                fullResponse += chunk;
                setOutputCode(fullResponse);
            }
        } catch (err) {
            const errorMessage = err instanceof Error ? err.message : 'An unknown error
```

```
        occurred.';
        setError(`Failed to transfer style: ${errorMessage}`);
    } finally {
        setIsLoading(false);
    }
}, [inputCode, styleGuide]);

return (
    <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
        <header className="mb-6">
            <h1 className="text-3xl font-bold flex items-center">
                <SparklesIcon />
                <span className="ml-3">AI Code Style Transfer</span>
            </h1>
            <p className="text-text-secondary mt-1">Rewrite code to match a specific style
            guide using AI.</p>
        </header>
        <div className="flex-grow flex flex-col gap-4 min-h-0">
            <div className="flex flex-col flex-1 min-h-0">
                <label htmlFor="input-code" className="text-sm font-medium text-text-secondary
                mb-2">Original Code</label>
                <textarea
                    id="input-code"
                    value={inputCode}
                    onChange={(e) => setInputCode(e.target.value)}
                    className="flex-grow p-4 bg-surface border border-border rounded-md resize-y
                    font-mono text-sm"
                />
            </div>
             <div className="flex flex-col flex-1 min-h-0">
                <label htmlFor="style-guide" className="text-sm font-medium text-text-secondary
                mb-2">Style Guide</label>
                <textarea
                    id="style-guide"
                    value={styleGuide}
                    onChange={(e) => setStyleGuide(e.target.value)}
                    className="flex-grow p-4 bg-surface border border-border rounded-md resize-y
                    font-mono text-sm"
                />
            </div>
             <div className="flex-shrink-0">
                <button
                    onClick={handleGenerate}
                    disabled={isLoading}
                    className="btn-primary w-full max-w-xs mx-auto flex items-center justify-center
                    px-6 py-3"
                >
                    {isLoading ? <LoadingSpinner /> : 'Rewrite Code'}
                </button>
            </div>
            <div className="flex flex-col flex-1 min-h-0">
                <label className="text-sm font-medium text-text-secondary mb-2">Rewritten
                Code</label>
                <div className="flex-grow p-1 bg-background border border-border rounded-md
                overflow-y-auto">
                    {isLoading && !outputCode && <div className="flex items-center justify-center
                    h-full"><LoadingSpinner /></div>}
                    {error && <p className="p-4 text-red-500">{error}</p>}
                    {outputCode && <MarkdownRenderer content={outputCode} />}
                     {!isLoading && !outputCode && !error && <div className="text-text-secondary
                     h-full flex items-center justify-center">Rewritten code will appear here.</div>}
                </div>
```

```
                </div>
            </div>
        </div>
    );
};

// ===== CronJobBuilder_36.tsx =====

import React, { useState, useMemo, useCallback, useEffect } from 'react';
import { CommandLineIcon, SparklesIcon } from '../icons.tsx';
import { generateCronFromDescription, CronParts } from
'../../services/index.ts';
import { LoadingSpinner } from '../shared/index.tsx';

const CronPartSelector: React.FC<{ label: string, value: string, onChange:
(value: string) => void, options: (string|number)[] }> = ({ label, value,
onChange, options }) => {
    return (
        <div>
            <label className="block text-sm font-medium text-text-secondary">{label}</label>
            <select value={value} onChange={e => onChange(e.target.value)} className="w-full
            mt-1 px-3 py-2 rounded-md bg-surface border border-border">
                <option value="*">* (every)</option>
                {options.map(o => <option key={o} value={o}>{o}</option>)}
            </select>
        </div>
    );
};

export const CronJobBuilder: React.FC<{ initialPrompt?: string }> = ({
initialPrompt }) => {
    const [minute, setMinute] = useState('0');
    const [hour, setHour] = useState('17');
    const [dayOfMonth, setDayOfMonth] = useState('*');
    const [month, setMonth] = useState('*');
    const [dayOfWeek, setDayOfWeek] = useState('1-5');
    const [aiPrompt, setAiPrompt] = useState(initialPrompt || '');
    const [isLoading, setIsLoading] = useState(false);

    const cronExpression = useMemo(() => {
        return `${minute} ${hour} ${dayOfMonth} ${month} ${dayOfWeek}`;
    }, [minute, hour, dayOfMonth, month, dayOfWeek]);

    const handleAiGenerate = useCallback(async (p: string) => {
        if (!p) return;
        setIsLoading(true);
        try {
            const result: CronParts = await generateCronFromDescription(p);
            setMinute(result.minute);
            setHour(result.hour);
            setDayOfMonth(result.dayOfMonth);
            setMonth(result.month);
            setDayOfWeek(result.dayOfWeek);
        } catch (e) {
            console.error(e);
        } finally {
            setIsLoading(false);
        }
    }, []);

    useEffect(() => {
        if (initialPrompt) {
```

```
            setAiPrompt(initialPrompt);
            handleAiGenerate(initialPrompt);
        }
    }, [initialPrompt, handleAiGenerate]);

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
            <header className="mb-6">
                <h1 className="text-3xl font-bold flex items-center">
                    <CommandLineIcon />
                    <span className="ml-3">AI Cron Job Builder</span>
                </h1>
                <p className="text-text-secondary mt-1">Visually construct a cron expression or
                describe it in plain English.</p>
            </header>
             <div className="flex gap-2 mb-6">
                <input type="text" value={aiPrompt} onChange={e => setAiPrompt(e.target.value)}
                placeholder="Describe a schedule..." className="flex-grow px-3 py-1.5 rounded-md
                bg-surface border border-border text-sm"/>
                <button onClick={() => handleAiGenerate(aiPrompt)} disabled={isLoading}
                className="btn-primary px-4 py-1.5 flex items-center gap-2">
                    {isLoading ? <LoadingSpinner /> : <SparklesIcon />} AI Generate
                </button>
            </div>
            <div className="grid grid-cols-2 md:grid-cols-5 gap-4 mb-6">
                <CronPartSelector label="Minute" value={minute} onChange={setMinute}
                options={Array.from({length: 60}, (_, i) => i)} />
                <CronPartSelector label="Hour" value={hour} onChange={setHour}
                options={Array.from({length: 24}, (_, i) => i)} />
                <CronPartSelector label="Day (Month)" value={dayOfMonth}
                onChange={setDayOfMonth} options={Array.from({length: 31}, (_, i) => i + 1)} />
                <CronPartSelector label="Month" value={month} onChange={setMonth}
                options={Array.from({length: 12}, (_, i) => i + 1)} />
                <CronPartSelector label="Day (Week)" value={dayOfWeek} onChange={setDayOfWeek}
                options={Array.from({length: 7}, (_, i) => i)} />
            </div>
            <div className="bg-surface p-4 rounded-lg text-center border border-border">
                <p className="text-text-secondary text-sm">Generated Expression</p>
                <p className="font-mono text-primary text-2xl mt-1">{cronExpression}</p>
                 <button onClick={() => navigator.clipboard.writeText(cronExpression)}
                 className="mt-4 px-3 py-1 bg-gray-100 hover:bg-gray-200 rounded-md text-
                 xs">Copy</button>
            </div>
        </div>
    );
};

// ===== CodeFormatter_36.tsx =====

import React, { useState, useCallback } from 'react';
import { formatCodeStream } from '../../services/index.ts';
import { CodeBracketSquareIcon } from '../icons.tsx';
import { LoadingSpinner } from '../shared/index.tsx';
import { MarkdownRenderer } from '../shared/index.tsx';

export const CodeFormatter: React.FC = () => {
    const [inputCode, setInputCode] = useState<string>('');
    const [formattedCode, setFormattedCode] = useState<string>('');
    const [isLoading, setIsLoading] = useState<boolean>(false);
    const [error, setError] = useState<string>('');

    const handleFormat = useCallback(async () => {
```

```
    if (!inputCode.trim()) {
        setError('Please enter some code to format.');
        return;
    }
    setIsLoading(true);
    setError('');
    setFormattedCode('');
    try {
        const stream = formatCodeStream(inputCode);
        let fullResponse = '';
        for await (const chunk of stream) {
            fullResponse += chunk;
            setFormattedCode(fullResponse);
        }
    } catch (err) {
        const errorMessage = err instanceof Error ? err.message : 'An unknown error
        occurred.';
        setError(`Failed to format code: ${errorMessage}`);
    } finally {
        setIsLoading(false);
    }
}, [inputCode]);

return (
    <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
        <header className="mb-6">
            <h1 className="text-3xl font-bold flex items-center">
                <CodeBracketSquareIcon />
                <span className="ml-3">AI Code Formatter</span>
            </h1>
            <p className="text-text-secondary mt-1">Clean up your code with AI-powered
            formatting, like a smart Prettier.</p>
        </header>
        <div className="flex-grow flex flex-col min-h-0">
            <div className="grid grid-cols-1 lg:grid-cols-2 gap-6 flex-grow min-h-0">
                <div className="flex flex-col h-full">
                    <label htmlFor="code-input" className="text-sm font-medium text-text-secondary
                    mb-2">Input</label>
                    <textarea
                        id="code-input"
                        value={inputCode}
                        onChange={(e) => setInputCode(e.target.value)}
                        placeholder="Paste your unformatted code here..."
                        className="flex-grow p-4 bg-surface border border-border rounded-md resize-none
                        font-mono text-sm"
                    />
                </div>
                <div className="flex flex-col h-full">
                    <label className="text-sm font-medium text-text-secondary mb-2">Output</label>
                    <div className="flex-grow p-1 bg-background border border-border rounded-md
                    overflow-y-auto">
                       {isLoading && !formattedCode && (
                            <div className="flex items-center justify-center h-full">
                                <LoadingSpinner />
                            </div>
                        )}
                        {error && <p className="p-4 text-red-500">{error}</p>}
                        {formattedCode && <MarkdownRenderer content={formattedCode} />}
                        {!isLoading && !formattedCode && !error && (
                            <div className="text-text-secondary h-full flex items-center justify-center">
                                Formatted code will appear here.
                            </div>
```

```
                            )}
                        </div>
                    </div>
                </div>
                 <button
                    onClick={handleFormat}
                    disabled={isLoading}
                    className="btn-primary mt-4 w-full max-w-sm mx-auto flex items-center justify-
                    center px-6 py-3"
                >
                    {isLoading ? <LoadingSpinner /> : 'Format Code'}
                </button>
            </div>
        </div>
    );
};

// ===== XbrlConverter_34.tsx =====

import React, { useState, useCallback } from 'react';
import { convertJsonToXbrlStream } from '../../services/aiService.ts';
import { XbrlConverterIcon } from '../icons.tsx';
import { LoadingSpinner, MarkdownRenderer } from '../shared/index.tsx';

export const XbrlConverter: React.FC<{ jsonInput?: string }> = ({ jsonInput:
initialJsonInput }) => {
    const [jsonInput, setJsonInput] = useState<string>(initialJsonInput || '');
    const [xbrlOutput, setXbrlOutput] = useState<string>('');
    const [isLoading, setIsLoading] = useState<boolean>(false);
    const [error, setError] = useState<string>('');

    const handleConvert = useCallback(async (jsonToConvert: string) => {
        if (!jsonToConvert.trim()) {
            setError('Please enter valid JSON to convert.');
            return;
        }
        setIsLoading(true);
        setError('');
        setXbrlOutput('');
        try {
            const stream = convertJsonToXbrlStream(jsonToConvert);
            let fullResponse = '';
            for await (const chunk of stream) {
                fullResponse += chunk;
                setXbrlOutput(fullResponse);
            }
        } catch (err) {
            const errorMessage = err instanceof Error ? err.message : 'An unknown error
            occurred.';
            setError(`Failed to convert: ${errorMessage}`);
        } finally {
            setIsLoading(false);
        }
    }, []);

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
            <header className="mb-6">
                <h1 className="text-3xl font-bold flex items-center">
                    <XbrlConverterIcon />
                    <span className="ml-3">JSON to XBRL Converter</span>
                </h1>
```

```
                <p className="text-text-secondary mt-1">Convert JSON data into a simplified
                XBRL-like XML format using AI.</p>
            </header>
            <div className="flex-grow flex flex-col gap-4 min-h-0">
                <div className="flex flex-col flex-1 min-h-0">
                    <label htmlFor="json-input" className="text-sm font-medium text-text-secondary
                    mb-2">JSON Input</label>
                    <textarea
                        id="json-input"
                        value={jsonInput}
                        onChange={(e) => setJsonInput(e.target.value)}
                        placeholder="Paste your JSON here..."
                        className="flex-grow p-4 bg-surface border border-border rounded-md resize-none
                        font-mono text-sm"
                    />
                </div>
                 <div className="flex-shrink-0">
                    <button
                        onClick={() => handleConvert(jsonInput)}
                        disabled={isLoading}
                        className="btn-primary w-full max-w-xs mx-auto flex items-center justify-center
                        px-6 py-3"
                    >
                        {isLoading ? <LoadingSpinner /> : 'Convert to XBRL'}
                    </button>
                </div>
                <div className="flex flex-col flex-1 min-h-0">
                    <label className="text-sm font-medium text-text-secondary mb-2">XBRL-like XML
                    Output</label>
                    <div className="relative flex-grow p-1 bg-background border border-border
                    rounded-md overflow-y-auto">
                        {isLoading && !xbrlOutput && <div className="flex items-center justify-center
                        h-full"><LoadingSpinner /></div>}
                        {error && <p className="p-4 text-red-500">{error}</p>}
                        {xbrlOutput && <MarkdownRenderer content={'```xml\n' +
                        xbrlOutput.replace(/```xml\n|```/g, '') + '\n```'} />}
                        {!isLoading && xbrlOutput && <button onClick={() =>
                        navigator.clipboard.writeText(xbrlOutput)} className="absolute top-2 right-2
                        px-2 py-1 bg-gray-100 hover:bg-gray-200 rounded-md text-xs">Copy XML</button>}
                        {!isLoading && !xbrlOutput && !error && <div className="text-text-secondary
                        h-full flex items-center justify-center">Output will appear here.</div>}
                    </div>
                </div>
            </div>
        </div>
    );
};

// ===== MarkdownSlides_36.tsx =====


import React, { useState, useMemo, useEffect, useRef, useCallback } from
'react';
import { marked } from 'marked';
import { PhotoIcon } from '../icons.tsx';
import { useNotification } from '../../contexts/NotificationContext.tsx';
import { useGlobalState } from '../../contexts/GlobalStateContext.tsx';
import { summarizeForSlides } from '../../services/index.ts';
import { createPresentation, addSlide } from
'../../services/workspaceService.ts';
import { LoadingSpinner } from '../shared/index.tsx';
export const MarkdownSlides: React.FC = () => {
```

```
const [markdown, setMarkdown] = useState('');
const [currentSlide, setCurrentSlide] = useState(0);
const [slideHtml, setSlideHtml] = useState<string | TrustedHTML>('');
const [isExporting, setIsExporting] = useState(false);
const presentationRef = useRef<HTMLDivElement>(null);
const { addNotification } = useNotification();
const { state } = useGlobalState();
const { user } = state;

const slides = useMemo(() => markdown.split(/^-{3,}\s*$/m), [markdown]);

useEffect(() => {
    const parse = async () => {
        const currentSlideContent = slides[currentSlide] || '';
        const html = await marked.parse(currentSlideContent);
        setSlideHtml(html);
    };
    parse();
}, [slides, currentSlide]);

const goToNext = useCallback(() => setCurrentSlide(s => Math.min(s + 1,
slides.length - 1)), [slides.length]);
const goToPrev = useCallback(() => setCurrentSlide(s => Math.max(s - 1, 0)),
[]);

const handleFullscreen = () => {
    presentationRef.current?.requestFullscreen();
};

const handleExportToSlides = async () => {
    if (!user) {
        addNotification('Please sign in with Google to export to Slides.', 'error');
        return;
    }
    setIsExporting(true);
    try {
        addNotification('Creating presentation...', 'info');
        const presentation = await createPresentation(`Presentation from DevCore: ${new
        Date().toLocaleDateString()}`);
        addNotification(`Presentation created: ${presentation.presentationId}`,
        'success');

        for (let i = 0; i < slides.length; i++) {
            addNotification(`Processing slide ${i + 1}/${slides.length}...`, 'info');
            const slideContent = slides[i];
            const summary = await summarizeForSlides(slideContent);
            await addSlide(presentation.presentationId, summary);
        }
        addNotification('All slides added successfully!', 'success');
        window.open(presentation.webViewLink, '_blank');

    } catch (e) {
        console.error(e);
        addNotification(e instanceof Error ? e.message : 'Failed to export
        presentation.', 'error');
    } finally {
        setIsExporting(false);
    }
};

useEffect(() => {
    const handleKeyDown = (e: KeyboardEvent) => {
```

```
            if (document.fullscreenElement === presentationRef.current) {
                if (e.key === 'ArrowRight' || e.key === ' ') goToNext();
                if (e.key === 'ArrowLeft') goToPrev();
            }
        };
        document.addEventListener('keydown', handleKeyDown);
        return () => document.removeEventListener('keydown', handleKeyDown);
    }, [goToNext, goToPrev]);

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
            <header className="mb-6">
                <h1 className="text-3xl font-bold flex items-center"><PhotoIcon /><span
                className="ml-3">Markdown to Slides</span></h1>
                <p className="text-text-secondary mt-1">Write markdown, present it as a
                slideshow. Use '---' to separate slides.</p>
            </header>
            <div className="flex-grow grid grid-cols-1 lg:grid-cols-2 gap-6 h-full overflow-
            hidden">
                <div className="flex flex-col h-full">
                     <label htmlFor="md-input" className="text-sm font-medium text-text-secondary
                     mb-2">Markdown Editor</label>
                     <textarea id="md-input" value={markdown} onChange={e =>
                     setMarkdown(e.target.value)} className="flex-grow p-4 bg-surface border border-
                     border rounded-md resize-none font-mono text-sm focus:ring-2 focus:ring-primary
                     focus:outline-none"/>
                </div>
                 <div ref={presentationRef} className="flex flex-col h-full bg-surface
                 fullscreen:bg-background border border-border rounded-md">
                    <div className="flex-shrink-0 flex justify-end items-center p-2 border-b border-
                    border gap-2">
                        <button onClick={handleExportToSlides} disabled={isExporting || !user}
                        className="btn-primary text-xs px-3 py-1 flex items-center gap-1 disabled:bg-
                        gray-400">
                           {isExporting ? <LoadingSpinner/> : 'Export to Google Slides'}
                        </button>
                        <button onClick={handleFullscreen} className="px-3 py-1 bg-gray-100 dark:bg-
                        slate-700 rounded-md text-xs hover:bg-gray-200 dark:hover:bg-
                        slate-600">Fullscreen</button>
                    </div>
                    <div className="relative flex-grow flex flex-col justify-center items-center p-8
                    overflow-y-auto">
                        <div className="prose prose-lg max-w-none w-full" dangerouslySetInnerHTML={{
                        __html: slideHtml }} />
                         <button onClick={goToPrev} disabled={currentSlide === 0} className="absolute
                         left-4 top-1/2 -translate-y-1/2 p-2 bg-gray-200/50 dark:bg-slate-700/50 rounded-
                         full disabled:opacity-30 hover:bg-gray-300/50 dark:hover:bg-
                         slate-600/50">■</button>
                         <button onClick={goToNext} disabled={currentSlide === slides.length - 1}
                         className="absolute right-4 top-1/2 -translate-y-1/2 p-2 bg-gray-200/50 dark:bg-
                         slate-700/50 rounded-full disabled:opacity-30 hover:bg-gray-300/50
                         dark:hover:bg-slate-600/50">■</button>
                         <div className="absolute bottom-4 right-4 text-xs bg-black/50 px-2 py-1 rounded-
                         md text-white">
                            {currentSlide + 1} / {slides.length}
                        </div>
                    </div>
                </div>
            </div>
        </div>
    );
};
```

```
// ===== SassScssCompiler_36.tsx =====

import React, { useState, useMemo } from 'react';
import { CodeBracketSquareIcon } from '../icons.tsx';

const escapeRegExp = (string: string): string => {
    // $& means the whole matched string
    return string.replace(/[.*+?^${}()|[\]\\]/g, '\\$&');
};

const compileScss = (scss: string): string => {
    try {
        let css = scss;
        css = css.replace(/\/\/.*$/gm, '');

        const variables: Record<string, string> = {};
        css = css.replace(/\$([\w-]+):\s*(.*?);/g, (_, name, value) => {
            variables[name] = value.trim(); return '';
        });

        for (let i = 0; i < 5; i++) {
            Object.entries(variables).forEach(([name, value]) => {
                css = css.replace(new RegExp(`\\$${escapeRegExp(name)}`, 'g'), value);
            });
        }

        css = css.replace(/([\d.]+)(px|rem|em|%)\s*([*\/])\s*([\d.]+)/g, (_, n1, unit,
        op, n2) => {
            const num1 = parseFloat(n1); const num2 = parseFloat(n2);
            const result = op === '*' ? num1 * num2 : num1 / num2;
            return `${result}${unit}`;
        });

        const processBlock = (block: string, parentSelector: string = ''): string => {
            let currentCss = '';
            let nestedCss = '';
            const properties = [];

            const regex =
            /((?:[\w-:.#&>+~*\s,]+|\([^)]*\))\s*\{[^{}]*\}|((?:[\w-]+\s*:[^;]+;))/g;
            const content = block.substring(block.indexOf('{') + 1, block.lastIndexOf('}'));
            let match;
            while ((match = regex.exec(content)) !== null) {
                if (match[1]) {
                    const nestedSelector = match[1].substring(0, match[1].indexOf('{')).trim();
                    const fullSelector = nestedSelector.includes('&') ? nestedSelector.replace(/&/g,
                    parentSelector) : `${parentSelector} ${nestedSelector}`.trim();
                    nestedCss += processBlock(match[1], fullSelector);
                } else if (match[2]) {
                    properties.push(`  ${match[2].trim()}`);
                }
            }

            if (properties.length > 0) {
                currentCss = `${parentSelector} {\n${properties.join('\n')}\n}\n`;
            }

            return currentCss + nestedCss;
        };

        let result = processBlock(`root{${css}}`, '').trim();
        return result.replace(/root\s*\{\s*\}/, '').trim();
```

```
    } catch(e) {
        console.error("SCSS Compilation Error:", e);
        return "/* Error compiling SCSS. Check console for details. */";
    }
};


export const SassScssCompiler: React.FC = () => {
    const [scss, setScss] = useState('');
    const compiledCss = useMemo(() => compileScss(scss), [scss]);

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
            <header className="mb-6">
                <h1 className="text-3xl flex items-center"><CodeBracketSquareIcon /><span
                className="ml-3">SASS/SCSS Compiler</span></h1>
                <p className="text-text-secondary mt-1">A real-time SASS/SCSS to CSS
                compiler.</p>
            </header>
            <div className="flex-grow flex flex-col gap-4 min-h-0">
                <div className="flex flex-col flex-1 min-h-0">
                    <label htmlFor="scss-input" className="text-sm font-medium text-text-secondary
                    mb-2">SASS/SCSS Input</label>
                    <textarea id="scss-input" value={scss} onChange={(e) => setScss(e.target.value)}
                    className="flex-grow p-4 bg-surface border border-border rounded-md resize-y
                    font-mono text-sm text-pink-600" spellCheck="false" />
                </div>
                <div className="flex flex-col flex-1 min-h-0">
                    <label className="text-sm font-medium text-text-secondary mb-2">Compiled CSS
                    Output</label>
                    <pre className="flex-grow p-4 bg-background border border-border rounded-md
                    overflow-y-auto text-blue-700 font-mono text-sm whitespace-pre-
                    wrap">{compiledCss}</pre>
                </div>
            </div>
        </div>
    );
};

// ===== CodeDiffGhost_36.tsx =====

import React, { useState, useEffect, useRef } from 'react';
import { EyeIcon } from '../icons.tsx';

export const CodeDiffGhost: React.FC = () => {
    const [oldCode, setOldCode] = useState('');
    const [newCode, setNewCode] = useState('');
    const [typedCode, setTypedCode] = useState('');
    const [isRunning, setIsRunning] = useState(false);
    const intervalRef = useRef<number | null>(null);

    const startAnimation = () => {
        if (intervalRef.current) clearInterval(intervalRef.current);
        setIsRunning(true);
        setTypedCode('');

        intervalRef.current = window.setInterval(() => {
            setTypedCode(prev => {
                if (prev.length < newCode.length) {
                    return newCode.substring(0, prev.length + 1);
                }
                if (intervalRef.current) clearInterval(intervalRef.current);
```

```
            setIsRunning(false);
            return newCode;
        });
    }, 15);
};

useEffect(() => {
    return () => {
        if (intervalRef.current) clearInterval(intervalRef.current);
    };
}, []);

return (
    <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
        <header className="mb-6">
            <h1 className="text-3xl flex items-center">
                <EyeIcon />
                <span className="ml-3">Code Diff Ghost</span>
            </h1>
            <p className="text-text-secondary mt-1">Visualize code changes with a "ghost
            typing" effect.</p>
        </header>
        <div className="flex justify-center mb-4">
            <button
                onClick={startAnimation}
                disabled={isRunning}
                className="btn-primary px-6 py-2"
            >
                {isRunning ? 'Visualizing...' : 'Show Changes'}
            </button>
        </div>
        <div className="flex-grow grid grid-cols-1 lg:grid-cols-2 gap-6 h-full overflow-
        hidden font-mono text-sm">
            <div className="flex flex-col h-full">
                <label htmlFor="before-code" className="text-sm font-medium text-text-secondary
                mb-2">Before</label>
                <textarea
                    id="before-code"
                    value={oldCode}
                    onChange={e => setOldCode(e.target.value)}
                    className="flex-grow p-4 bg-surface border border-border rounded-md text-red-600
                    whitespace-pre-wrap resize-none"
                    spellCheck="false"
                />
            </div>
             <div className="flex flex-col h-full">
                <label htmlFor="after-code" className="text-sm font-medium text-text-secondary
                mb-2">After</label>
                 <div className="relative flex-grow">
                    <textarea
                        id="after-code"
                        value={newCode}
                        onChange={e => setNewCode(e.target.value)}
                        className="absolute inset-0 w-full h-full p-4 bg-surface border border-border
                        rounded-md text-emerald-700 whitespace-pre-wrap resize-none z-0"
                        spellCheck="false"
                    />
                    {(isRunning || typedCode) && (
                        <pre className="absolute inset-0 w-full h-full p-4 bg-surface pointer-events-
                        none text-emerald-700 whitespace-pre-wrap z-10">
                            {typedCode}{isRunning && <span className="animate-pulse">|</span>}
                        </pre>
```

```
                    )}
                </div>
            </div>
        </div>
    </div>
    );
};

// ===== ResponsiveTester_36.tsx =====

import React, { useState, useEffect } from 'react';
import { EyeIcon } from '../icons.tsx';

const devices = {
    'iPhone 12': { width: 390, height: 844 },
    'Pixel 5': { width: 393, height: 851 },
    'iPad Air': { width: 820, height: 1180 },
    'Surface Duo': { width: 540, height: 720 },
    'Laptop': { width: 1366, height: 768 },
    'Desktop': { width: 1920, height: 1080 },
    'Auto': { width: '100%', height: '100%' },
};

type DeviceName = keyof typeof devices;

export const ResponsiveTester: React.FC = () => {
    const [url, setUrl] = useState('');
    const [displayUrl, setDisplayUrl] = useState('');
    const [size, setSize] = useState<{width: number | string, height: number |
    string}>(devices['Auto']);

    useEffect(() => {
        const handleResize = () => {
            if (size.width === '100%') {
                setSize({ width: '100%', height: '100%' });
            }
        };
        window.addEventListener('resize', handleResize);
        return () => window.removeEventListener('resize', handleResize);
    }, [size.width]);

    const handleUrlSubmit = (e: React.FormEvent) => {
        e.preventDefault();
        if (!url) return;
        setDisplayUrl(url.startsWith('http') ? url : `https://${url}`);
    };

    const handleRotate = () => {
        if(typeof size.width === 'number' && typeof size.height === 'number') {
            setSize({ width: size.height, height: size.width });
        }
    };

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
            <header className="mb-6"><h1 className="text-3xl font-bold flex items-
            center"><EyeIcon /><span className="ml-3">Responsive Tester</span></h1><p
            className="text-text-secondary mt-1">Preview your web pages at different screen
            sizes.</p></header>
            <form onSubmit={handleUrlSubmit} className="flex items-center gap-2 mb-2">
                <input type="text" value={url} onChange={(e) => setUrl(e.target.value)}
                placeholder="https://example.com" className="flex-grow px-4 py-2 rounded-md bg-
```

```
                surface border border-border focus:ring-2 focus:ring-primary focus:outline-
                none"/>
                <button type="submit" className="btn-primary px-6 py-2">Load</button>
            </form>
            <div className="bg-surface p-2 rounded-lg flex flex-wrap justify-center items-
            center gap-2 mb-4 border border-border">
                {Object.keys(devices).map(name => (
                    <button key={name} onClick={() => setSize(devices[name as DeviceName])}
                    className={`px-3 py-1 rounded-md text-sm ${JSON.stringify(size) ===
                    JSON.stringify(devices[name as DeviceName]) ? 'bg-primary/10 text

// ===== SecurityScanner_24.tsx =====

import React, { useState } from 'react';
import { analyzeCodeForVulnerabilities } from '../../services/aiService.ts';
import type { SecurityVulnerability } from '../../types.ts';
import { ShieldCheckIcon, SparklesIcon } from '../icons.tsx';
import { LoadingSpinner, MarkdownRenderer } from '../shared/index.tsx';

const exampleCode = `function UserProfile({ user }) {
  // TODO: remove this temporary api key
  const API_KEY = "sk-xxxxxxxxxxxxxxxxxxxxxxxxxxxx";
  const userContent = user.bio; // This might contain malicious scripts

  return (
    <div>
      <h2>{user.name}</h2>
      <div dangerouslySetInnerHTML={{ __html: userContent }} />
    </div>
  );
}`;

export const SecurityScanner: React.FC = () => {
    const [code, setCode] = useState(exampleCode);
    const [aiIssues, setAiIssues] = useState<SecurityVulnerability[]>([]);
    const [isLoading, setIsLoading] = useState(false);
    const [error, setError] = useState('');

    const handleScan = async () => {
        if (!code.trim()) {
            setError('Please enter code to scan.');
            return;
        }
        setIsLoading(true);
        setError('');
        setAiIssues([]);
        try {
            const geminiIssues = await analyzeCodeForVulnerabilities(code);
            setAiIssues(geminiIssues);

        } catch (err) {
            setError(err instanceof Error ? err.message : 'An error occurred during
            scanning.');
        } finally {
            setIsLoading(false);
        }
    };

    const SeverityBadge: React.FC<{ severity: string }> = ({ severity }) => {
        const colors: Record<string, string> = {
            'Critical': 'bg-red-500 text-white',
            'High': 'bg-red-400 text-white',
```

```
        'Medium': 'bg-yellow-400 text-yellow-900',
        'Low': 'bg-blue-400 text-white',
        'Informational': 'bg-gray-400 text-gray-900',
    };
    return <span className={`px-2 py-0.5 text-xs font-bold rounded-full
    ${colors[severity] || 'bg-gray-300'}`}>{severity}</span>
}

return (
    <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
        <header className="mb-6">
            <h1 className="text-3xl font-bold flex items-center"><ShieldCheckIcon /><span
            className="ml-3">AI Security Co-Pilot</span></h1>
            <p className="text-text-secondary mt-1">Find vulnerabilities in your code with
            static analysis and AI.</p>
        </header>
        <div className="flex-grow grid grid-cols-1 lg:grid-cols-2 gap-6 min-h-0">
            <div className="flex flex-col">
                <label className="text-sm mb-2">Code to Scan</label>
                <textarea value={code} onChange={e => setCode(e.target.value)} className="w-full
                flex-grow p-2 bg-surface border rounded font-mono text-xs" />
                <button onClick={handleScan} disabled={isLoading} className="btn-primary w-full
                mt-4 py-2 flex justify-center items-center gap-2">{isLoading ? <LoadingSpinner/>
                : 'Scan Code'}</button>
            </div>
            <div className="flex flex-col bg-surface p-4 border rounded-lg">
                <h3 className="text-lg font-bold mb-2">Scan Results</h3>
                {error && <p className="text-red-500">{error}</p>}
                <div className="flex-grow overflow-y-auto pr-2 space-y-4">
                    {isLoading && <div className="flex justify-center items-center
                    h-full"><LoadingSpinner/></div>}
                    {!isLoading && aiIssues.length === 0 && <p className="text-text-secondary text-
                    center mt-8">No issues found. Run a scan to begin.</p>}

                     {aiIssues.length > 0 && <div>
                        <h4 className="font-semibold text-sm mb-1 flex items-center
                        gap-1"><SparklesIcon/> AI-Powered Findings</h4>
                        {aiIssues.map((issue, i) => (
                            <details key={`ai-${i}`} className="p-2 bg-background border rounded mb-2">
                                <summary className="cursor-pointer font-bold flex items-center
                                gap-2">{issue.vulnerability} <SeverityBadge severity={issue.severity}
                                /></summary>
                                <div className="mt-2 pt-2 border-t text-xs space-y-2">
                                    <p><strong>Description:</strong> {issue.description}</p>
                                    <p><strong>Mitigation:</strong> {issue.mitigation}</p>
                                    {issue.exploitSuggestion && (
                                        <div>
                                            <strong>Exploit Simulation:</strong>
                                            <div className="mt-1 p-2 bg-gray-50 rounded">
                                                <MarkdownRenderer content={'```bash\n' + issue.exploitSuggestion + '\n```'}/>
                                            </div>
                                        </div>
                                    )}
                                </div>
                            </details>
                        ))}
                    </div>}
                </div>
            </div>
        </div>
    </div>
);
```

```
};

// ===== AppLauncher.tsx =====


import React, { useState, useEffect, useCallback } from 'react';
import { ArrowUpOnSquareIcon, CpuChipIcon } from '../icons.tsx';
import { useGlobalState } from '../../contexts/GlobalStateContext.tsx';
import { getUserToken } from '../../services/firebaseService.ts';
import { initializeOctokit } from '../../services/authService.ts';
import type { Octokit } from 'octokit';
import { LoadingSpinner } from '../shared/index.tsx';
import { useNotification } from '../../contexts/NotificationContext.tsx';
import type { Repo } from '../../types.ts';

const externalApps = [
    { name: 'interactive_ai_book', path: 'jocall3/interactive_ai_book' },
    { name: 'buggedout', path: 'jocall3/buggedout' },
    { name: 'ai-dev-core', path: 'jocall3/ai-dev-core' },
    { name: 'jamesburvelocallaghaniii', path: 'jocall3/jamesburvelocallaghaniii' },
    { name: 'whoknows', path: 'jocall3/whoknows' },
    { name: 'demo', path: 'jocall3/demo' },
    { name: 'ai-file-manager', path: 'jocall3/ai-file-manager' },
    { name: 'ai-presentation-generator', path: 'jocall3/ai-presentation-generator'
    },
    { name: 'generate-book-', path: 'jocall3/generate-book-' },
    { name: 'automatedpatentsalmost', path: 'jocall3/automatedpatentsalmost' },
    { name: 'Ai-Bible-aka-AIBLE-', path: 'jocall3/Ai-Bible-aka-AIBLE-' },
    { name: 'aigooglemaps', path: 'jocall3/aigooglemaps' },
    { name: 'quantumbankinglanguagegen', path: 'jocall3/quantumbankinglanguagegen'
    },
    { name: 'modern_treasury_embedded_payment_flow', path:
    'jocall3/modern_treasury_embedded_payment_flow' },
    { name: 'Quantum-Code-Architect', path: 'jocall3/Quantum-Code-Architect' },
    { name: 'devcoreai', path: 'jocall3/devcoreai' },
    { name: 'aipitchdeck', path: 'jocall3/aipitchdeck' },
    { name: 'googledriveaifilemaker', path: 'jocall3/googledriveaifilemaker' },
    { name: 'swaggerai', path: 'jocall3/swaggerai' },
    { name: 'aifile', path: 'jocall3/aifile' },
    { name: 'ai_web_designer', path: 'jocall3/ai_web_designer' }
];

const AppCard: React.FC<{ repo: Repo | null; onExplore: (owner: string, repo:
string) => void; originalPath: string; }> = ({ repo, onExplore, originalPath })
=> {
    if (!repo) {
        return (
            <div className="bg-surface border border-border rounded-lg p-4 flex flex-col
            items-start justify-between text-text-secondary">
                 <h3 className="font-bold text-text-primary
                 truncate">{originalPath.split('/')[1]}</h3>
                 <p className="text-xs font-mono mb-2">{originalPath}</p>
                 <p className="text-sm my-4">Failed to load repository data.</p>
            </div>
        );
    }

    const { full_name, name, description, stargazers_count, html_url } = repo;
    const [owner, repoName] = full_name.split('/');

    return (
        <div className="bg-surface border border-border rounded-lg p-4 flex flex-col
```

```
        items-start justify-between transition-all duration-200 hover:shadow-lg
        hover:border-primary/50">
            <div>
                <div className="flex items-center gap-3 mb-2">
                    <div className="text-primary"><CpuChipIcon /></div>
                    <h3 className="font-bold text-text-primary truncate">{name}</h3>
                </div>
                <p className="text-xs text-text-secondary font-mono mb-2">{full_name}</p>
                <p className="text-sm text-text-secondary mb-3 h-16 overflow-hidden text-
                ellipsis">{description || 'No description available.'}</p>
                <div className="text-xs text-text-secondary flex items-center gap-1">
                    <svg xmlns="http://www.w3.org/2000/svg" className="h-4 w-4" viewBox="0 0 20 20"
                    fill="currentColor"><path d="M9.049 2.927c.3-.921 1.603-.921 1.902 0l1.07
                    3.292a1 1 0 00.95.69h3.462c.969 0 1.371 1.24.588 1.81l-2.8 2.034a1 1 0 00-.364
                    1.118l1.07 3.292c.3.921-.755 1.688-1.54 1.118l-2.8-2.034a1 1 0 00-1.175 0l-2.8
                    2.034c-.784.57-1.838-.197-1.539-1.118l1.07-3.292a1 1 0 00-.364-1.118L2.98
                    8.72c-.783-.57-.38-1.81.588-1.81h3.461a1 1 0 00.951-.69l1.07-3.292z" /></svg>
                    {stargazers_count}
                </div>
            </div>
            <div className="w-full mt-4 space-y-2">
                 <button
                    onClick={() => onExplore(owner, repoName)}
                    className="w-full text-center px-4 py-2 text-sm bg-gray-100 dark:bg-slate-700
                    rounded-md hover:bg-gray-200 dark:hover:bg-slate-600"
                >
                    Explore Files
                </button>
                <a
                    href={html_url}
                    target="_blank"
                    rel="noopener noreferrer"
                    className="btn-primary w-full text-center block px-4 py-2 text-sm"
                >
                    View on GitHub
                </a>
            </div>
        </div>
    );
};

const AppCardSkeleton: React.FC = () => (
    <div className="bg-surface border border-border rounded-lg p-4 animate-pulse">
        <div className="flex items-center gap-3 mb-2">
            <div className="w-6 h-6 bg-gray-200 dark:bg-slate-700 rounded-full"></div>
            <div className="h-4 bg-gray-200 dark:bg-slate-700 rounded w-3/4"></div>
        </div>
        <div className="h-3 bg-gray-200 dark:bg-slate-700 rounded w-1/2 mb-2"></div>
        <div className="space-y-1 h-16">
            <div className="h-3 bg-gray-200 dark:bg-slate-700 rounded"></div>
            <div className="h-3 bg-gray-200 dark:bg-slate-700 rounded w-5/6"></div>
        </div>
        <div className="h-3 bg-gray-200 dark:bg-slate-700 rounded w-1/4 mt-3"></div>
        <div className="mt-4 space-y-2">
            <div className="h-9 bg-gray-200 dark:bg-slate-700 rounded w-full"></div>
            <div className="h-9 bg-gray-300 dark:bg-slate-600 rounded w-full"></div>
        </div>
    </div>
);


export const AppLauncher: React.FC = () => {
```

```
const { state, dispatch } = useGlobalState();
const { user, githubUser } = state;
const { addNotification } = useNotification();

const [repos, setRepos] = useState<(Repo | null)[]>([]);
const [isLoading, setIsLoading] = useState(true);
const [error, setError] = useState('');

const fetchRepoData = useCallback(async () => {
    if (!user) {
        setError("Please sign in to fetch app data.");
        setIsLoading(false);
        return;
    }
    if (!githubUser) {
        setError("Please connect to GitHub in the Workspace Hub to fetch app data.");
        setIsLoading(false);
        return;
    }

    let octokit: Octokit;
    try {
        const token = await getUserToken(user.uid, 'github_pat');
        if (!token) {
            throw new Error("GitHub token not found. Please add it on the Connections
            page.");
        }
        octokit = initializeOctokit(token);
    } catch (err) {
        setError(err instanceof Error ? err.message : 'Failed to initialize GitHub
        client.');
        setIsLoading(false);
        return;
    }

    setIsLoading(true);
    setError('');

    try {
        const promises = externalApps.map(app => {
            const [owner, repo] = app.path.split('/');
            return octokit.request('GET /repos/{owner}/{repo}', { owner, repo })
                .then(response => response.data as Repo)
                .catch(err => {
                    console.warn(`Failed to fetch ${app.path}:`, err);
                    return null;
                });
        });

        const results = await Promise.all(promises);
        setRepos(results);
    } catch (err) {
        setError(err instanceof Error ? err.message : 'An unexpected error occurred
        while fetching repositories.');
    } finally {
        setIsLoading(false);
    }
}, [user, githubUser]);

useEffect(() => {
    fetchRepoData();
}, [fetchRepoData]);
```

```
    const handleExplore = (owner: string, repo: string) => {
        dispatch({ type: 'SET_SELECTED_REPO', payload: { owner, repo } });
        dispatch({ type: 'SET_VIEW', payload: { view: 'project-explorer' } });
        addNotification(`Loading repository: ${owner}/${repo}`, 'info');
    };

    return (
        <div className="w-full h-full p-4 sm:p-6 lg:p-8 text-text-primary overflow-y-
        auto">
            <header className="mb-8">
                <h1 className="text-4xl font-extrabold tracking-tight flex items-center">
                    <ArrowUpOnSquareIcon />
                    <span className="ml-3">App Launcher</span>
                </h1>
                <p className="mt-2 text-lg text-text-secondary">
                    Explore and launch external AI-powered applications and toolkits from the
                    community.
                </p>
                <p className="mt-1 text-sm text-text-secondary">
                    Note: These are external applications and cannot be "installed" directly into
                    this toolkit. This launcher pulls their data from GitHub and lets you explore
                    their files.
                </p>
            </header>

            {error && (
                <div className="bg-red-500/10 border border-red-500/20 text-red-700 dark:text-
                red-400 p-4 rounded-lg text-center">
                    <p className="font-bold">Could not load apps</p>
                    <p className="text-sm">{error}</p>
                </div>
            )}

            <div className="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-3 xl:grid-cols-4
            gap-4">
                {isLoading
                    ? Array.from({ length: externalApps.length }).map((_, i) => <AppCardSkeleton
                    key={i} />)
                    : repos.map((repo, index) => (
                        <AppCard
                            key={repo?.id || index}
                            repo={repo}
                            onExplore={handleExplore}
                            originalPath={externalApps[index].path}
                        />
                    ))
                }
            </div>
        </div>
    );
};


// ===== AiArchitect.tsx =====

import React, { useState, useCallback, useEffect, useRef } from 'react';
import mermaid from 'mermaid';
import { generateArchitecture } from '../../services/aiService.ts';
import type { GeneratedFile } from '../../types.ts';
import { BuildingLibraryIcon } from '../icons.tsx';
import { LoadingSpinner, MarkdownRenderer } from '../shared/index.tsx';
import { saveFile } from '../../services/dbService.ts';
```

```
import { useNotification } from '../../contexts/NotificationContext.tsx';
import { useGlobalState } from '../../contexts/GlobalStateContext.tsx';

mermaid.initialize({ startOnLoad: false, theme: 'dark', securityLevel: 'loose'
});

export const AiArchitect: React.FC = () => {
    const [prompt, setPrompt] = useState('A simple blog application with posts and
    comments');
    const [diagram, setDiagram] = useState('');
    const [files, setFiles] = useState<GeneratedFile[]>([]);
    const [isLoading, setIsLoading] = useState(false);
    const [error, setError] = useState('');
    const [activeFile, setActiveFile] = useState<GeneratedFile | null>(null);
    const mermaidContainerRef = useRef<HTMLDivElement>(null);
    const { addNotification } = useNotification();
    const { dispatch } = useGlobalState();

    const handleGenerate = useCallback(async () => {
        if (!prompt.trim()) {
            setError('Please describe the application you want to build.');
            return;
        }
        setIsLoading(true);
        setError('');
        setDiagram('');
        setFiles([]);
        setActiveFile(null);
        try {
            const result = await generateArchitecture(prompt);
            setDiagram(result.diagram.replace(/```mermaid\n|```/g, '').trim());
            setFiles(result.files);
            if (result.files.length > 0) {
                setActiveFile(result.files[0]);
            }
        } catch (err) {
            setError(err instanceof Error ? err.message : 'Failed to generate
            architecture.');
        } finally {
            setIsLoading(false);
        }
    }, [prompt]);

    useEffect(() => {
        const renderMermaid = async () => {
            if (diagram && mermaidContainerRef.current) {
                try {
                    mermaidContainerRef.current.innerHTML = ''; // Clear previous
                    const { svg } = await mermaid.render(`mermaid-graph-${Date.now()}`, diagram);
                    mermaidContainerRef.current.innerHTML = svg;
                } catch (e) {
                    console.error("Mermaid rendering error:", e);
                    mermaidContainerRef.current.innerHTML = `<p class="text-red-400">Error rendering
                    diagram. Check console.</p>`;
                }
            }
        };
        renderMermaid();
    }, [diagram]);

    const handleSaveAndOpen = async () => {
        if (files.length === 0) return;
```

```
    try {
        for (const file of files) {
            await saveFile(file);
        }
        addNotification('Project files saved to local storage!', 'success');
        dispatch({ type: 'SET_VIEW', payload: { view: 'project-explorer' } });
    } catch(e) {
        addNotification('Failed to save files.', 'error');
    }
};

return (
    <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-ink">
        <header className="mb-6">
            <h1 className="text-3xl font-bold flex items-center">
                <BuildingLibraryIcon />
                <span className="ml-3">AI Architect</span>
            </h1>
            <p className="text-silver mt-1">Design and bootstrap entire application
            architectures from a high-level description.</p>
        </header>
        <div className="flex-grow grid grid-cols-1 lg:grid-cols-2 gap-6 min-h-0">
            <div className="flex flex-col gap-4">
                <textarea
                    value={prompt}
                    onChange={e => setPrompt(e.target.value)}
                    placeholder="Describe your application..."
                    className="w-full p-3 h-24 bg-navy border border-border rounded-lg resize-none
                    focus:ring-2 focus:ring-primary"
                />
                <div className="flex gap-2">
                    <button onClick={handleGenerate} disabled={isLoading} className="btn-primary
                    flex-grow py-3 flex items-center justify-center">
                        {isLoading ? <LoadingSpinner /> : 'Generate Architecture'}
                    </button>
                     <button onClick={handleSaveAndOpen} disabled={files.length === 0}
                     className="px-6 py-3 bg-emerald-600 text-white font-bold rounded-md
                     hover:opacity-90 disabled:opacity-50">
                        Save & Open in IDE
                    </button>
                </div>
                {error && <p className="text-red-400 text-center">{error}</p>}
                <div className="flex-grow bg-navy border border-border rounded-lg flex items-
                center justify-center p-4 overflow-auto mermaid-diagram">
                    {isLoading && !diagram ? <LoadingSpinner /> : <div ref={mermaidContainerRef}
                    className="w-full h-full" />}
                </div>
            </div>
            <div className="flex flex-col min-h-0">
                <div className="flex-shrink-0 flex border-b border-border mb-2 overflow-x-auto
                no-scrollbar">
                    {files.map(file => (
                        <button key={file.filePath} onClick={() => setActiveFile(file)} className={`px-4
                        py-2 text-sm whitespace-nowrap ${activeFile?.filePath === file.filePath ? 'bg-
                        navy text-primary font-semibold' : 'text-silver'}`}>
                            {file.filePath}
                        </button>
                    ))}
                </div>
                <div className="flex-grow bg-navy border border-border rounded-lg overflow-
                auto">
                    {activeFile ? (
```

```
                            <MarkdownRenderer content={'```tsx\n' + activeFile.content + '\n```'} />
                        ) : (
                            <div className="flex items-center justify-center h-full text-silver">Generated
                            files will appear here.</div>
                        )}
                    </div>
                </div>
            </div>
        </div>
    );
};


// ===== ResponsiveTester_40.tsx =====

import React, { useState, useEffect } from 'react';
import { EyeIcon } from '../icons.tsx';

const devices = {
    'iPhone 12': { width: 390, height: 844 },
    'Pixel 5': { width: 393, height: 851 },
    'iPad Air': { width: 820, height: 1180 },
    'Surface Duo': { width: 540, height: 720 },
    'Laptop': { width: 1366, height: 768 },
    'Desktop': { width: 1920, height: 1080 },
    'Auto': { width: '100%', height: '100%' },
};

type DeviceName = keyof typeof devices;

export const ResponsiveTester: React.FC = () => {
    const [url, setUrl] = useState('');
    const [displayUrl, setDisplayUrl] = useState('');
    const [size, setSize] = useState<{width: number | string, height: number |
    string}>(devices['Auto']);

    useEffect(() => {
        const handleResize = () => {
            if (size.width === '100%') {
                setSize({ width: '100%', height: '100%' });
            }
        };
        window.addEventListener('resize', handleResize);
        return () => window.removeEventListener('resize', handleResize);
    }, [size.width]);

    const handleUrlSubmit = (e: React.FormEvent) => {
        e.preventDefault();
        if (!url) return;
        setDisplayUrl(url.startsWith('http') ? url : `https://${url}`);
    };

    const handleRotate = () => {
        if(typeof size.width === 'number' && typeof size.height === 'number') {
            setSize({ width: size.height, height: size.width });
        }
    };

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
            <header className="mb-6"><h1 className="text-3xl font-bold flex items-
            center"><EyeIcon /><span className="ml-3">Responsive Tester</span></h1><p
```

```
            className="text-text-secondary mt-1">Preview your web pages at different screen
            sizes.</p></header>
            <form onSubmit={handleUrlSubmit} className="flex items-center gap-2 mb-2">
                <input type="text" value={url} onChange={(e) => setUrl(e.target.value)}
                placeholder="https://example.com" className="flex-grow px-4 py-2 rounded-md bg-
                surface border border-border focus:ring-2 focus:ring-primary focus:outline-
                none"/>
                <button type="submit" className="btn-primary px-6 py-2">Load</button>
            </form>
            <div className="bg-surface p-2 rounded-lg flex flex-wrap justify-center items-
            center gap-2 mb-4 border border-border">
                {Object.keys(devices).map(name => (
                    <button key={name} onClick={() => setSize(devices[name as DeviceName])}
                    className={`px-3 py-1 rounded-md text-sm ${JSON.stringify(size) ===
                    JSON.stringify(devices[name as DeviceName]) ? 'bg-primary text-text-on-primary'
                    : 'bg-surface hover:bg-slate-700'}`}>
                        {name}
                    </button>
                ))}
                <button onClick={handleRotate} className="px-3 py-1 bg-surface hover:bg-
                slate-700 rounded-md text-sm">Rotate</button>
                <div className="text-sm font-mono p-1">
                    {typeof size.width === 'number' ? `${size.width}px` : size.width} x {typeof
                    size.height === 'number' ? `${size.height}px` : size.height}
                </div>
            </div>
            <div className="flex-grow bg-background border-2 border-dashed border-border
            rounded-lg flex items-center justify-center overflow-auto p-4">
                <div className="shadow-2xl" style={{ width: size.width, height: size.height }}>
                    {displayUrl ? <iframe src={displayUrl} title="Responsive Preview"
                    className="w-full h-full border-none bg-white"/> : <div className="w-full h-full
                    flex items-center justify-center text-text-secondary">Enter a URL to
                    begin</div>}
                </div>
            </div>
        </div>
    );
};

// ===== AiCommitGenerator_43.tsx =====

import React, { useState, useCallback, useEffect } from 'react';
import { generateCommitMessageStream } from '../../services/index.ts';
import { downloadFile } from '../../services/fileUtils.ts';
import { GitBranchIcon, ArrowDownTrayIcon } from '../icons.tsx';
import { LoadingSpinner } from '../shared/index.tsx';

export const AiCommitGenerator: React.FC<{ diff?: string }> = ({ diff:
initialDiff }) => {
    const [diff, setDiff] = useState<string>(initialDiff || '');
    const [message, setMessage] = useState<string>('');
    const [isLoading, setIsLoading] = useState<boolean>(false);
    const [error, setError] = useState<string>('');

    const handleGenerate = useCallback(async (diffToAnalyze: string) => {
        if (!diffToAnalyze.trim()) {
            setError('Please paste a diff to generate a message.');
            return;
        }
        setIsLoading(true);
        setError('');
        setMessage('');
```

```
    try {
        const stream = generateCommitMessageStream(diffToAnalyze);
        let fullResponse = '';
        for await (const chunk of stream) {
            fullResponse += chunk;
            setMessage(fullResponse);
        }
    } catch (err) {
        const errorMessage = err instanceof Error ? err.message : 'An unknown error
        occurred.';
        setError(`Failed to generate message: ${errorMessage}`);
    } finally {
        setIsLoading(false);
    }
}, []);

useEffect(() => {
    if (initialDiff) {
        setDiff(initialDiff);
        handleGenerate(initialDiff);
    }
}, [initialDiff, handleGenerate]);

const handleCopy = () => {
    navigator.clipboard.writeText(message);
};

const handleDownload = () => {
    downloadFile(message, 'commit_message.txt', 'text/plain');
};

return (
    <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
        <header className="mb-6">
            <h1 className="text-3xl font-bold flex items-center">
                <GitBranchIcon />
                <span className="ml-3">AI Commit Message Generator</span>
            </h1>
            <p className="text-text-secondary mt-1">Paste your diff and let Gemini craft the
            perfect commit message.</p>
        </header>
        <div className="flex-grow flex flex-col gap-4 min-h-0">
            <div className="flex flex-col flex-1 min-h-0">
                <label htmlFor="diff-input" className="text-sm font-medium text-text-secondary
                mb-2">Git Diff</label>
                <textarea
                    id="diff-input"
                    value={diff}
                    onChange={(e) => setDiff(e.target.value)}
                    placeholder="Paste your git diff here..."
                    className="flex-grow p-4 bg-surface border border-border rounded-md resize-none
                    font-mono text-sm text-text-primary focus:ring-2 focus:ring-primary
                    focus:outline-none"
                />
            </div>
            <div className="flex-shrink-0">
                <button
                    onClick={() => handleGenerate(diff)}
                    disabled={isLoading}
                    className="btn-primary w-full max-w-xs mx-auto flex items-center justify-center
                    px-6 py-3"
                >
```

```
                        {isLoading ? <LoadingSpinner /> : 'Generate Commit Message'}
                    </button>
                </div>
                <div className="flex flex-col flex-1 min-h-0">
                    <div className="flex justify-between items-center mb-2">
                        <label className="text-sm font-medium text-text-secondary">Generated
                        Message</label>
                        {message && !isLoading && (
                            <div className="flex items-center gap-2">
                                <button onClick={handleCopy} className="px-3 py-1 bg-gray-100 text-xs rounded-md
                                hover:bg-gray-200">Copy</button>
                                <button onClick={handleDownload} className="flex items-center gap-1 px-3 py-1
                                bg-gray-100 text-xs rounded-md hover:bg-gray-200">
                                    <ArrowDownTrayIcon className="w-4 h-4" /> Download
                                </button>
                            </div>
                        )}
                    </div>
                    <div className="relative flex-grow p-4 bg-surface border border-border rounded-
                    md overflow-y-auto">
                        {isLoading && (
                             <div className="flex items-center justify-center h-full">
                                <LoadingSpinner />
                             </div>
                        )}
                        {error && <p className="text-red-500">{error}</p>}
                        {message && !isLoading && (
                           <pre className="whitespace-pre-wrap font-sans text-text-primary">{message}</pre>
                        )}
                         {!isLoading && !message && !error && (
                            <div className="text-text-secondary h-full flex items-center justify-center">
                                The commit message will appear here.
                            </div>
                        )}
                    </div>
                </div>
            </div>
        </div>
    );
};

// ===== CodeReviewBot_45.tsx =====

import React, { useState, useCallback } from 'react';
import { reviewCodeStream } from '../../services/index.ts';
import { useAiPersonalities } from '../../hooks/useAiPersonalities.ts';
import { formatSystemPromptToString } from '../../utils/promptUtils.ts';
import { CpuChipIcon } from '../icons.tsx';
import { LoadingSpinner } from '../shared/index.tsx';
import { MarkdownRenderer } from '../shared/index.tsx';

export const CodeReviewBot: React.FC = () => {
    const [code, setCode] = useState<string>('');
    const [review, setReview] = useState<string>('');
    const [isLoading, setIsLoading] = useState<boolean>(false);
    const [error, setError] = useState<string>('');
    const [personalities] = useAiPersonalities();
    const [selectedPersonalityId, setSelectedPersonalityId] =
    useState<string>('default');

    const handleGenerate = useCallback(async () => {
        if (!code.trim()) {
```

```
        setError('Please enter some code to review.');
        return;
    }
    setIsLoading(true);
    setError('');
    setReview('');

    let systemInstruction: string | undefined = undefined;
    if (selectedPersonalityId !== 'default') {
        const personality = personalities.find(p => p.id === selectedPersonalityId);
        if (personality) {
            systemInstruction = formatSystemPromptToString(personality);
        }
    }

    try {
        const stream = reviewCodeStream(code, systemInstruction);
        let fullResponse = '';
        for await (const chunk of stream) {
            fullResponse += chunk;
            setReview(fullResponse);
        }
    } catch (err) {
        const errorMessage = err instanceof Error ? err.message : 'An unknown error
        occurred.';
        setError(`Failed to get review: ${errorMessage}`);
    } finally {
        setIsLoading(false);
    }
}, [code, selectedPersonalityId, personalities]);

return (
    <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
        <header className="mb-6">
            <h1 className="text-3xl font-bold flex items-center">
                <CpuChipIcon />
                <span className="ml-3">AI Code Review Bot</span>
            </h1>
            <p className="text-text-secondary mt-1">Get an automated code review from
            Gemini.</p>
        </header>
        <div className="flex-grow flex flex-col gap-4 min-h-0">
            <div className="flex flex-col flex-1 min-h-0">
                <label htmlFor="code-input" className="text-sm font-medium text-text-secondary
                mb-2">Code to Review</label>
                <textarea
                    id="code-input"
                    value={code}
                    onChange={(e) => setCode(e.target.value)}
                    placeholder="Paste your code here..."
                    className="flex-grow p-4 bg-surface border border-border rounded-md resize-none
                    font-mono text-sm"
                />
            </div>
             <div className="flex-shrink-0 flex items-center justify-center gap-4">
                 <div className="w-full max-w-xs">
                    <label htmlFor="personality-select" className="text-sm font-medium text-text-
                    secondary">Reviewer Personality</label>
                    <select
                        id="personality-select"
                        value={selectedPersonalityId}
                        onChange={e => setSelectedPersonalityId(e.target.value)}
```

```
                            className="w-full mt-1 p-2 bg-surface border border-border rounded-md text-sm"
                        >
                            <option value="default">Default</option>
                            {personalities.map(p => (
                                <option key={p.id} value={p.id}>{p.name}</option>
                            ))}
                        </select>
                     </div>
                    <button
                        onClick={handleGenerate}
                        disabled={isLoading}
                        className="btn-primary self-end h-[42px] w-full max-w-xs flex items-center
                        justify-center px-6 py-3"
                    >
                        {isLoading ? <LoadingSpinner /> : 'Request Review'}
                    </button>
                </div>
                <div className="flex flex-col flex-1 min-h-0">
                    <label className="text-sm font-medium text-text-secondary mb-2">AI
                    Feedback</label>
                    <div className="flex-grow p-4 bg-background border border-border rounded-md
                    overflow-y-auto">
                        {isLoading && !review && <div className="flex items-center justify-center
                        h-full"><LoadingSpinner /></div>}
                        {error && <p className="text-red-500">{error}</p>}
                        {review && <MarkdownRenderer content={review} />}
                         {!isLoading && !review && !error && <div className="text-text-secondary h-full
                         flex items-center justify-center">Review will appear here.</div>}
                    </div>
                </div>
            </div>
        </div>
    );
};

// ===== CiCdPipelineGenerator_33.tsx =====

import React, { useState } from 'react';
import { generateCiCdConfig } from '../../services/index.ts';
import { PaperAirplaneIcon, SparklesIcon } from '../icons.tsx';
import { LoadingSpinner, MarkdownRenderer } from '../shared/index.tsx';

const platforms = ['GitHub Actions', 'GitLab CI', 'CircleCI', 'Jenkins'];

export const CiCdPipelineGenerator: React.FC = () => {
    const [platform, setPlatform] = useState(platforms[0]);
    const [description, setDescription] = useState('');
    const [config, setConfig] = useState('');
    const [isLoading, setIsLoading] = useState(false);
    const [error, setError] = useState('');

    const handleGenerate = async () => {
        if (!description.trim()) {
            setError('Please provide a description of the pipeline stages.');
            return;
        }
        setIsLoading(true);
        setError('');
        try {
            const result = await generateCiCdConfig(platform, description);
            setConfig(result);
        } catch (err) {
```

```
            setError(err instanceof Error ? err.message : 'Failed to generate config.');
        } finally {
            setIsLoading(false);
        }
    };

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
            <header className="mb-6">
                <h1 className="text-3xl font-bold flex items-center"><PaperAirplaneIcon /><span
                className="ml-3">AI CI/CD Pipeline Architect</span></h1>
                <p className="text-text-secondary mt-1">Describe your deployment process and get
                a modern configuration file.</p>
            </header>
            <div className="flex-grow flex flex-col gap-4 min-h-0">
                 <div className="flex flex-col flex-1 min-h-0">
                    <div className="grid grid-cols-1 md:grid-cols-3 gap-4 mb-4">
                        <div><label className="block text-sm">Platform</label><select value={platform}
                        onChange={e => setPlatform(e.target.value)} className="w-full mt-1 p-2 bg-
                        surface border rounded"><option>GitHub Actions</option><option>GitLab
                        CI</option><option>CircleCI</option></select></div>
                        <div className="md:col-span-2"><label className="block text-sm">Describe
                        Stages</label><input type="text" value={description} onChange={e =>
                        setDescription(e.target.value)} className="w-full mt-1 p-2 bg-surface border
                        rounded" placeholder="e.g., Install dependencies, run tests, build, and
                        deploy"/></div>
                    </div>
                     <button onClick={handleGenerate} disabled={isLoading} className="btn-primary
                     w-full max-w-xs mx-auto flex items-center justify-center py-2"><SparklesIcon />
                     {isLoading ? 'Generating...' : 'Generate Configuration'}</button>
                </div>
                 <div className="flex flex-col flex-grow min-h-0">
                    <label className="text-sm font-medium text-text-secondary mb-2">Generated
                    Configuration File</label>
                    <div className="relative flex-grow p-1 bg-background border border-border
                    rounded-md overflow-y-auto">
                        {isLoading && !config && <div className="flex items-center justify-center
                        h-full"><LoadingSpinner /></div>}
                        {error && <p className="p-4 text-red-500">{error}</p>}
                        {config && <MarkdownRenderer content={config} />}
                         {!isLoading && !config && !error && <div className="text-text-secondary h-full
                         flex items-center justify-center">Generated config will appear here.</div>}
                    </div>
                </div>
            </div>
        </div>
    );
};

// ===== SecurityScanner_33.tsx =====

import React, { useState } from 'react';
import { analyzeCodeForVulnerabilities } from '../../services/aiService.ts';
import type { SecurityVulnerability } from '../../types.ts';
import { ShieldCheckIcon, SparklesIcon } from '../icons.tsx';
import { LoadingSpinner, MarkdownRenderer } from '../shared/index.tsx';

export const SecurityScanner: React.FC = () => {
    const [code, setCode] = useState('');
    const [aiIssues, setAiIssues] = useState<SecurityVulnerability[]>([]);
    const [isLoading, setIsLoading] = useState(false);
    const [error, setError] = useState('');
```

```
const handleScan = async () => {
    if (!code.trim()) {
        setError('Please enter code to scan.');
        return;
    }
    setIsLoading(true);
    setError('');
    setAiIssues([]);
    try {
        const geminiIssues = await analyzeCodeForVulnerabilities(code);
        setAiIssues(geminiIssues);

    } catch (err) {
        setError(err instanceof Error ? err.message : 'An error occurred during
        scanning.');
    } finally {
        setIsLoading(false);
    }
};

const SeverityBadge: React.FC<{ severity: string }> = ({ severity }) => {
    const colors: Record<string, string> = {
        'Critical': 'bg-red-500 text-white',
        'High': 'bg-red-400 text-white',
        'Medium': 'bg-yellow-400 text-yellow-900',
        'Low': 'bg-blue-400 text-white',
        'Informational': 'bg-gray-400 text-gray-900',
    };
    return <span className={`px-2 py-0.5 text-xs font-bold rounded-full
    ${colors[severity] || 'bg-gray-300'}`}>{severity}</span>
}

return (
    <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
        <header className="mb-6">
            <h1 className="text-3xl font-bold flex items-center"><ShieldCheckIcon /><span
            className="ml-3">AI Security Co-Pilot</span></h1>
            <p className="text-text-secondary mt-1">Find vulnerabilities in your code with
            static analysis and AI.</p>
        </header>
        <div className="flex-grow grid grid-cols-1 lg:grid-cols-2 gap-6 min-h-0">
            <div className="flex flex-col">
                <label className="text-sm mb-2">Code to Scan</label>
                <textarea value={code} onChange={e => setCode(e.target.value)}
                placeholder="Paste your code snippet here..." className="w-full flex-grow p-2
                bg-surface border rounded font-mono text-xs" />
                <button onClick={handleScan} disabled={isLoading} className="btn-primary w-full
                mt-4 py-2 flex justify-center items-center gap-2">{isLoading ? <LoadingSpinner/>
                : 'Scan Code'}</button>
            </div>
            <div className="flex flex-col bg-surface p-4 border rounded-lg">
                <h3 className="text-lg font-bold mb-2">Scan Results</h3>
                {error && <p className="text-red-500">{error}</p>}
                <div className="flex-grow overflow-y-auto pr-2 space-y-4">
                    {isLoading && <div className="flex justify-center items-center
                    h-full"><LoadingSpinner/></div>}
                    {!isLoading && aiIssues.length === 0 && <p className="text-text-secondary text-
                    center mt-8">No issues found. Run a scan to begin.</p>}

                     {aiIssues.length > 0 && <div>
                        <h4 className="font-semibold text-sm mb-1 flex items-center
                        gap-1"><SparklesIcon/> AI-Powered Findings</h4>
```

```
                            {aiIssues.map((issue, i) => (
                                <details key={`ai-${i}`} className="p-2 bg-background border rounded mb-2">
                                    <summary className="cursor-pointer font-bold flex items-center
                                    gap-2">{issue.vulnerability} <SeverityBadge severity={issue.severity}
                                    /></summary>
                                    <div className="mt-2 pt-2 border-t text-xs space-y-2">
                                        <p><strong>Description:</strong> {issue.description}</p>
                                        <p><strong>Mitigation:</strong> {issue.mitigation}</p>
                                        {issue.exploitSuggestion && (
                                            <div>
                                                <strong>Exploit Simulation:</strong>
                                                <div className="mt-1 p-2 bg-gray-50 rounded">
                                                    <MarkdownRenderer content={'```bash\n' + issue.exploitSuggestion + '\n```'}/>
                                                </div>
                                            </div>
                                        )}
                                    </div>
                                </details>
                            ))}
                        </div>}
                    </div>
                </div>
            </div>
        );
};

// ===== OneClickRefactor_18.tsx =====

import React, { useState, useCallback } from 'react';
import * as Diff from 'diff';
import { applySpecificRefactor, refactorForPerformance, refactorForReadability,
generateJsDoc, convertToFunctionalComponent } from
'../../services/aiService.ts';
import { SparklesIcon } from '../icons.tsx';
import { LoadingSpinner } from '../shared/index.tsx';

type RefactorAction = 'readability' | 'performance' | 'jsdoc' | 'functional' |
'custom';

const DiffViewer: React.FC<{ oldCode: string, newCode: string }> = ({ oldCode,
newCode }) => {
    const diff = Diff.diffLines(oldCode, newCode);

    return (
        <pre className="whitespace-pre-wrap font-mono text-xs">
            {diff.map((part, index) => {
                const color = part.added ? 'bg-green-500/20' : part.removed ? 'bg-red-500/20' :
                'bg-transparent';
                return <div key={index} className={color}>{part.value}</div>;
            })}
        </pre>
    );
};


export const OneClickRefactor: React.FC = () => {
    const [code, setCode] = useState('');
    const [refactoredCode, setRefactoredCode] = useState('');
    const [loadingAction, setLoadingAction] = useState<RefactorAction | null>(null);

    const handleRefactor = useCallback(async (action: RefactorAction) => {
```

```
    if (!code.trim()) return;
    setLoadingAction(action);
    setRefactoredCode('');

    let stream;
    switch(action) {
        case 'readability':
            stream = refactorForReadability(code);
            break;
        case 'performance':
            stream = refactorForPerformance(code);
            break;
        case 'jsdoc':
            stream = generateJsDoc(code);
            break;
        case 'functional':
            stream = convertToFunctionalComponent(code);
            break;
        default:
            setLoadingAction(null);
            return;
    }

    try {
        let fullResponse = '';
        for await (const chunk of stream) {
            fullResponse += chunk;
            setRefactoredCode(fullResponse.replace(/^```(?:\w+\n)?/, '').replace(/```$/,
            ''));
        }
    } catch (e) {
        console.error(e);
        setRefactoredCode(`// Error during refactoring: ${e instanceof Error ? e.message
        : 'Unknown error'}`);
    } finally {
        setLoadingAction(null);
    }
}, [code]);

return (
    <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
        <header className="mb-6">
            <h1 className="text-3xl font-bold flex items-center">
                <SparklesIcon />
                <span className="ml-3">One-Click Refactor</span>
            </h1>
            <p className="text-text-secondary mt-1">Apply common refactoring patterns to
            your code with a single click.</p>
        </header>
        <div className="flex items-center justify-center flex-wrap gap-2 mb-4 p-4 bg-
        surface rounded-lg border border-border">
            <button onClick={() => handleRefactor('readability')} disabled={!!loadingAction}
            className="btn-primary px-3 py-1.5 text-sm">{loadingAction === 'readability' ?
            <LoadingSpinner/> : 'Improve Readability'}</button>
            <button onClick={() => handleRefactor('performance')} disabled={!!loadingAction}
            className="btn-primary px-3 py-1.5 text-sm">{loadingAction === 'performance' ?
            <LoadingSpinner/> : 'Boost Performance'}</button>
            <button onClick={() => handleRefactor('jsdoc')} disabled={!!loadingAction}
            className="btn-primary px-3 py-1.5 text-sm">{loadingAction === 'jsdoc' ?
            <LoadingSpinner/> : 'Add JSDoc'}</button>
            <button onClick={() => handleRefactor('functional')} disabled={!!loadingAction}
            className="btn-primary px-3 py-1.5 text-sm">{loadingAction === 'functional' ?
```

```
                <LoadingSpinner/> : 'To Functional Component'}</button>
            </div>
            <div className="flex-grow grid grid-cols-1 lg:grid-cols-2 gap-6 min-h-0">
                <div className="flex flex-col">
                    <label className="text-sm font-medium mb-2">Original Code</label>
                    <textarea value={code} onChange={e => setCode(e.target.value)} className="flex-
                    grow p-2 bg-surface border rounded font-mono text-xs" placeholder="Paste your
                    code here..."/>
                </div>
                 <div className="flex flex-col">
                    <label className="text-sm font-medium mb-2">Refactored Code</label>
                    <div className="flex-grow p-2 bg-background border rounded overflow-auto">
                        {loadingAction ? <div className="flex justify-center items-center
                        h-full"><LoadingSpinner/></div> : <DiffViewer oldCode={code}
                        newCode={refactoredCode} />}
                    </div>
                </div>
            </div>
        </div>
    );
};

// ===== BugReproducer_18.tsx =====

import React, { useState, useCallback } from 'react';
import { generateBugReproductionTestStream } from '../../services/aiService.ts';
import { BugAntIcon } from '../icons.tsx';
import { LoadingSpinner, MarkdownRenderer } from '../shared/index.tsx';

export const BugReproducer: React.FC = () => {
    const [stackTrace, setStackTrace] = useState('');
    const [context, setContext] = useState('');
    const [generatedTest, setGeneratedTest] = useState('');
    const [isLoading, setIsLoading] = useState(false);
    const [error, setError] = useState('');

    const handleGenerate = useCallback(async () => {
        if (!stackTrace.trim()) {
            setError('Please provide a stack trace.');
            return;
        }
        setIsLoading(true);
        setError('');
        setGeneratedTest('');
        try {
            const stream = generateBugReproductionTestStream(stackTrace, context);
            let fullResponse = '';
            for await (const chunk of stream) {
                fullResponse += chunk;
                setGeneratedTest(fullResponse);
            }
        } catch (err) {
            setError(err instanceof Error ? err.message : 'An unknown error occurred.');
        } finally {
            setIsLoading(false);
        }
    }, [stackTrace, context]);

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
            <header className="mb-6">
                <h1 className="text-3xl font-bold flex items-center">
```

```
                    <BugAntIcon />
                    <span className="ml-3">Automated Bug Reproducer</span>
                </h1>
                <p className="text-text-secondary mt-1">Paste a stack trace to automatically
                generate a failing unit test.</p>
            </header>
            <div className="flex-grow grid grid-cols-1 lg:grid-cols-2 gap-6 min-h-0">
                <div className="flex flex-col gap-4">
                    <div className="flex flex-col flex-1 min-h-0">
                        <label htmlFor="stack-trace" className="text-sm font-medium mb-2">Stack
                        Trace</label>
                        <textarea id="stack-trace" value={stackTrace} onChange={e =>
                        setStackTrace(e.target.value)} className="flex-grow p-2 bg-surface border
                        rounded font-mono text-xs" placeholder="Paste stack trace here..."/>
                    </div>
                     <div className="flex flex-col flex-1 min-h-0">
                        <label htmlFor="context" className="text-sm font-medium mb-2">Relevant Code /
                        Context (Optional)</label>
                        <textarea id="context" value={context} onChange={e =>
                        setContext(e.target.value)} className="flex-grow p-2 bg-surface border rounded
                        font-mono text-xs" placeholder="Paste relevant code snippets..."/>
                    </div>
                    <button onClick={handleGenerate} disabled={isLoading} className="btn-primary
                    w-full py-3">{isLoading ? <LoadingSpinner/> : 'Generate Test'}</button>
                </div>
                <div className="flex flex-col">
                    <label className="text-sm font-medium mb-2">Generated Test File</label>
                    <div className="flex-grow p-1 bg-background border rounded overflow-auto">
                        {isLoading && !generatedTest && <div className="flex justify-center items-center
                        h-full"><LoadingSpinner /></div>}
                        {error && <p className="text-red-500 p-4">{error}</p>}
                        {generatedTest && <MarkdownRenderer content={generatedTest} />}
                    </div>
                </div>
            </div>
        </div>
    );
};

// ===== TechDebtSonar_18.tsx =====

import React, { useState, useCallback } from 'react';
import { detectCodeSmells } from '../../services/aiService.ts';
import type { CodeSmell } from '../../types.ts';
import { MagnifyingGlassIcon } from '../icons.tsx';
import { LoadingSpinner } from '../shared/index.tsx';

export const TechDebtSonar: React.FC = () => {
    const [code, setCode] = useState('');
    const [smells, setSmells] = useState<CodeSmell[]>([]);
    const [isLoading, setIsLoading] = useState(false);
    const [error, setError] = useState('');

    const handleScan = useCallback(async () => {
        if (!code.trim()) {
            setError('Please provide code to scan.');
            return;
        }
        setIsLoading(true);
        setError('');
        setSmells([]);
        try {
```

```
            const result = await detectCodeSmells(code);
            setSmells(result);
        } catch (err) {
            setError(err instanceof Error ? err.message : 'An unknown error occurred.');
        } finally {
            setIsLoading(false);
        }
    }, [code]);

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
            <header className="mb-6">
                <h1 className="text-3xl font-bold flex items-center">
                    <MagnifyingGlassIcon />
                    <span className="ml-3">Tech Debt Sonar</span>
                </h1>
                <p className="text-text-secondary mt-1">Scan code to find code smells and areas
                with high complexity.</p>
            </header>
            <div className="flex-grow grid grid-cols-1 lg:grid-cols-2 gap-6 min-h-0">
                <div className="flex flex-col">
                    <label className="text-sm font-medium mb-2">Code to Analyze</label>
                    <textarea value={code} onChange={e => setCode(e.target.value)} className="flex-
                    grow p-2 bg-surface border rounded font-mono text-xs" placeholder="Paste code to
                    analyze for smells..."/>
                    <button onClick={handleScan} disabled={isLoading} className="btn-primary w-full
                    mt-4 py-3">{isLoading ? <LoadingSpinner/> : 'Scan for Code Smells'}</button>
                </div>
                <div className="flex flex-col">
                    <label className="text-sm font-medium mb-2">Detected Smells</label>
                    <div className="flex-grow p-2 bg-background border rounded overflow-auto">
                        {isLoading && <div className="flex justify-center items-center
                        h-full"><LoadingSpinner /></div>}
                        {error && <p className="text-red-500 p-4">{error}</p>}
                        {!isLoading && smells.length === 0 && <p className="text-text-secondary text-
                        center pt-8">No smells detected, or scan not run.</p>}
                        {smells.length > 0 && (
                            <div className="space-y-3">
                                {smells.map((smell, i) => (
                                    <div key={i} className="p-3 bg-surface border border-border rounded-lg">
                                        <div className="flex justify-between items-center">
                                            <h4 className="font-bold text-primary">{smell.smell}</h4>
                                            <span className="text-xs font-mono bg-gray-100 dark:bg-slate-700 px-2 py-1
                                            rounded">Line: {smell.line}</span>
                                        </div>
                                        <p className="text-sm mt-1">{smell.explanation}</p>
                                    </div>
                                ))}
                            </div>
                        )}
                    </div>
                </div>
            </div>
        </div>
    );
};

// ===== IamPolicyGenerator_18.tsx =====

import React, { useState, useCallback } from 'react';
import { generateIamPolicyStream } from '../../services/aiService.ts';
import { ShieldCheckIcon } from '../icons.tsx';
```

```
import { LoadingSpinner, MarkdownRenderer } from '../shared/index.tsx';

export const IamPolicyGenerator: React.FC = () => {
    const [description, setDescription] = useState('');
    const [platform, setPlatform] = useState<'aws' | 'gcp'>('aws');
    const [policy, setPolicy] = useState('');
    const [isLoading, setIsLoading] = useState(false);
    const [error, setError] = useState('');

    const handleGenerate = useCallback(async () => {
        if (!description.trim()) {
            setError('Please provide a description.');
            return;
        }
        setIsLoading(true);
        setError('');
        setPolicy('');
        try {
            const stream = generateIamPolicyStream(description, platform);
            let fullResponse = '';
            for await (const chunk of stream) {
                fullResponse += chunk;
                setPolicy(fullResponse);
            }
        } catch (err) {
            setError(err instanceof Error ? err.message : 'An unknown error occurred.');
        } finally {
            setIsLoading(false);
        }
    }, [description, platform]);

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
            <header className="mb-6">
                <h1 className="text-3xl font-bold flex items-center">
                    <ShieldCheckIcon />
                    <span className="ml-3">IAM Policy Generator</span>
                </h1>
                <p className="text-text-secondary mt-1">Generate AWS or GCP IAM policies from a
                natural language description.</p>
            </header>
            <div className="flex-grow grid grid-cols-1 lg:grid-cols-2 gap-6 min-h-0">
                <div className="flex flex-col gap-4">
                     <div>
                        <label htmlFor="platform" className="text-sm font-medium mb-2 block">Cloud
                        Platform</label>
                        <div className="flex gap-2 p-1 bg-surface rounded-lg border">
                            <button onClick={() => setPlatform('aws')} className={`flex-1 py-2 rounded-md
                            text-sm ${platform === 'aws' ? 'bg-primary text-text-on-primary' :
                            ''}`}>AWS</button>
                            <button onClick={() => setPlatform('gcp')} className={`flex-1 py-2 rounded-md
                            text-sm ${platform === 'gcp' ? 'bg-primary text-text-on-primary' :
                            ''}`}>GCP</button>
                        </div>
                    </div>
                    <div className="flex flex-col flex-1 min-h-0">
                        <label htmlFor="description" className="text-sm font-medium mb-2">Describe the
                        desired permissions</label>
                        <textarea id="description" value={description} onChange={e =>
                        setDescription(e.target.value)} className="flex-grow p-2 bg-surface border
                        rounded text-sm" placeholder="e.g., A user role that can read from S3 buckets
                        but not write or delete."/>
```

```
                    </div>
                    <button onClick={handleGenerate} disabled={isLoading} className="btn-primary
                    w-full py-3">{isLoading ? <LoadingSpinner/> : 'Generate Policy'}</button>
                </div>
                <div className="flex flex-col">
                    <label className="text-sm font-medium mb-2">Generated Policy (JSON)</label>
                    <div className="flex-grow p-1 bg-background border rounded overflow-auto">
                        {isLoading && !policy && <div className="flex justify-center items-center
                        h-full"><LoadingSpinner /></div>}
                        {error && <p className="text-red-500 p-4">{error}</p>}
                        {policy && <MarkdownRenderer content={policy} />}
                    </div>
                </div>
            </div>
        </div>
    );
};

// ===== ProjectExplorer_46.tsx =====

import React, { useState, useEffect, useCallback } from 'react';
import { useGlobalState } from '../../contexts/GlobalStateContext.tsx';
import { useNotification } from '../../contexts/NotificationContext.tsx';
import { initializeOctokit } from '../../services/authService.ts';
import { getDecryptedCredential } from '../../services/vaultService.ts';
import { getRepos, getRepoTree, getFileContent, commitFiles } from
'../../services/githubService.ts';
import { generateCommitMessageStream } from '../../services/index.ts';
import type { Repo, FileNode } from '../../types.ts';
import { FolderIcon, DocumentIcon } from '../icons.tsx';
import { LoadingSpinner } from '../shared/index.tsx';
import * as Diff from 'diff';

const FileTree: React.FC<{ node: FileNode, onFileSelect: (path: string, name:
string) => void, activePath: string | null }> = ({ node, onFileSelect,
activePath }) => {
    const [isOpen, setIsOpen] = useState(true);

    if (node.type === 'file') {
        const isActive = activePath === node.path;
        return (
            <div
                className={`flex items-center space-x-2 pl-4 py-1 cursor-pointer rounded
                ${isActive ? 'bg-primary/10 text-primary' : 'hover:bg-gray-100 dark:hover:bg-
                slate-700'}`}
                onClick={() => onFileSelect(node.path, node.name)}
            >
                <DocumentIcon />
                <span>{node.name}</span>
            </div>
        );
    }

    return (
        <div>
            <div
                className="flex items-center space-x-2 py-1 cursor-pointer hover:bg-gray-100
                dark:hover:bg-slate-700 rounded"
                onClick={() => setIsOpen(!isOpen)}
            >
                <div className={`transform transition-transform ${isOpen ? 'rotate-90' :
                ''}`}>■</div>
```

```
                <FolderIcon />
                <span className="font-semibold">{node.name}</span>
            </div>
            {isOpen && node.children && (
                <div className="pl-4 border-l border-border ml-3">
                    {node.children.map(child => <FileTree key={child.path} node={child}
                    onFileSelect={onFileSelect} activePath={activePath} />)}
                </div>
            )}
        </div>
    );
};

export const ProjectExplorer: React.FC = () => {
    const { state, dispatch } = useGlobalState();
    const { user, githubUser, selectedRepo, projectFiles } = state;
    const { addNotification } = useNotification();
    const [repos, setRepos] = useState<Repo[]>([]);
    const [isLoading, setIsLoading] = useState<'repos' | 'tree' | 'file' | 'commit'
    | null>(null);
    const [error, setError] = useState('');
    const [activeFile, setActiveFile] = useState<{ path: string; name: string;
    originalContent: string; editedContent: string} | null>(null);

    const getApiClient = useCallback(async () => {
        if (!user) {
            throw new Error("You must be logged in to use the Project Explorer.");
        }
        // NOTE: This assumes the vault is unlocked. A more robust implementation
        // might use the useVaultModal hook to prompt for unlock if needed.
        const token = await getDecryptedCredential('github_pat');
        if (!token) {
            throw new Error("GitHub token not found. Please add it on the Connections
            page.");
        }
        return initializeOctokit(token);
    }, [user]);


    useEffect(() => {
        const loadRepos = async () => {
            if (user && githubUser) {
                setIsLoading('repos');
                setError('');
                try {
                    const octokit = await getApiClient();
                    const userRepos = await getRepos(octokit);
                    setRepos(userRepos);
                } catch (err) {
                    setError(err instanceof Error ? err.message : 'Failed to load repositories');
                } finally {
                    setIsLoading(null);
                }
            } else {
                setRepos([]);
            }
        };
        loadRepos();
    }, [user, githubUser, getApiClient]);

    useEffect(() => {
        const loadTree = async () => {
```

```
         if (selectedRepo && user && githubUser) {
            setIsLoading('tree');
            setError('');
            setActiveFile(null);
            try {
                const octokit = await getApiClient();
                const tree = await getRepoTree(octokit, selectedRepo.owner, selectedRepo.repo);
                dispatch({ type: 'LOAD_PROJECT_FILES', payload: tree });
            } catch (err) {
                 setError(err instanceof Error ? err.message : 'Failed to load repository tree');
            } finally {
                setIsLoading(null);
            }
        }
    };
    loadTree();
}, [selectedRepo, user, githubUser, dispatch, getApiClient]);

const handleFileSelect = async (path: string, name: string) => {
    if (!selectedRepo) return;
    setIsLoading('file');
    try {
        const octokit = await getApiClient();
        const content = await getFileContent(octokit, selectedRepo.owner,
        selectedRepo.repo, path);
        setActiveFile({ path, name, originalContent: content, editedContent: content });
    } catch (err) {
        setError((err as Error).message);
    } finally {
        setIsLoading(null);
    }
};

const handleCommit = async () => {
    if (!activeFile || !selectedRepo || activeFile.originalContent ===
    activeFile.editedContent) return;

    setIsLoading('commit');
    setError('');
    try {
        const diff = Diff.createPatch(activeFile.path, activeFile.originalContent,
        activeFile.editedContent);

        const stream = generateCommitMessageStream(diff);
        let commitMessage = '';
        for await (const chunk of stream) { commitMessage += chunk; }

        const finalMessage = window.prompt("Confirm or edit commit message:",
        commitMessage);
        if (!finalMessage) {
            setIsLoading(null);
            return;
        }

        const octokit = await getApiClient();
        await commitFiles(
            octokit,
            selectedRepo.owner,
            selectedRepo.repo,
            [{ path: activeFile.path, content: activeFile.editedContent }],
            finalMessage
        );
```

```
        addNotification(`Successfully committed to ${selectedRepo.repo}`, 'success');
        setActiveFile(prev => prev ? { ...prev, originalContent: prev.editedContent } :
        null);

    } catch (err) {
        const message = err instanceof Error ? err.message : 'Failed to commit changes';
        setError(message);
        addNotification(message, 'error');
    } finally {
        setIsLoading(null);
    }
};

if (!user) {
    return (
        <div className="h-full flex flex-col items-center justify-center text-center
        text-text-secondary p-4">
            <FolderIcon />
            <h2 className="text-lg font-semibold mt-2">Please Sign In</h2>
            <p>Sign in via the "Connections" tab to explore your repositories.</p>
        </div>
    );
}

if (!githubUser) {
     return (
        <div className="h-full flex flex-col items-center justify-center text-center
        text-text-secondary p-4">
            <FolderIcon />
            <h2 className="text-lg font-semibold mt-2">Connect to GitHub</h2>
            <p>Please go to the "Connections" tab and provide a Personal Access Token to
            explore your repositories.</p>
        </div>
    );
}

const hasChanges = activeFile ? activeFile.originalContent !==
activeFile.editedContent : false;

return (
    <div className="h-full flex flex-col text-text-primary">
        <header className="p-4 border-b border-border flex-shrink-0">
            <h1 className="text-xl font-bold flex items-center"><FolderIcon /><span
            className="ml-3">Project Explorer</span></h1>
            <div className="mt-2">
                <select
                    value={selectedRepo ? `${selectedRepo.owner}/${selectedRepo.repo}` : ''}
                    onChange={e => {
                        const [owner, repo] = e.target.value.split('/');
                        dispatch({ type: 'SET_SELECTED_REPO', payload: { owner, repo } });
                    }}
                    className="w-full p-2 bg-surface border border-border rounded-md text-sm"
                >
                    <option value="" disabled>{isLoading === 'repos' ? 'Loading...' : 'Select a
                    repository'}</option>
                    {repos.map(r => <option key={r.id} value={r.full_name}>{r.full_name}</option>)}
                </select>
            </div>
            {error && <p className="text-red-500 text-xs mt-2">{error}</p>}
        </header>
        <div className="flex-grow flex min-h-0">
            <aside className="w-1/3 bg-background border-r border-border p-4 overflow-y-
```

```
                auto">
                    {isLoading === 'tree' && <div className="flex justify-center"><LoadingSpinner
                    /></div>}
                    {projectFiles && <FileTree node={projectFiles} onFileSelect={handleFileSelect}
                    activePath={activeFile?.path ?? null} />}
                </aside>
                <main className="flex-1 bg-surface flex flex-col">
                     <div className="flex justify-between items-center p-2 border-b border-border bg-
                     gray-50 dark:bg-slate-800">
                        <span className="text-sm font-semibold">{activeFile?.name || 'No file
                        selected'}</span>
                        <button onClick={handleCommit} disabled={!hasChanges || isLoading === 'commit'}
                        className="btn-primary px-4 py-1 text-sm flex items-center justify-center
                        min-w-[100px]">
                           {isLoading === 'commit' ? <LoadingSpinner/> : 'Commit'}
                        </button>
                     </div>
                     {isLoading === 'file' ? <div className="flex items-center justify-center
                     h-full"><LoadingSpinner /></div> :
                        <textarea
                            value={activeFile?.editedContent ?? 'Select a file to view its content.'}
                            onChange={e => setActiveFile(prev => prev ? { ...prev, editedContent:
                            e.target.value } : null)}
                            disabled={!activeFile}
                            className="w-full h-full p-4 text-sm font-mono bg-transparent resize-none
                            focus:outline-none"
                        />
                    }
                </main>
            </div>
        </div>
    );
};

// ===== WorkspaceConnectorHub_15.tsx =====

import React, { useState, useEffect, useMemo, useCallback } from 'react';
import { useGlobalState } from '../../contexts/GlobalStateContext.tsx';
import * as vaultService from '../../services/vaultService.ts';
import { useNotification } from '../../contexts/NotificationContext.tsx';
import { validateToken } from '../../services/authService.ts';
import { ACTION_REGISTRY, executeWorkspaceAction } from
'../../services/workspaceConnectorService.ts';
import { RectangleGroupIcon, GithubIcon, SparklesIcon } from '../icons.tsx';
import { LoadingSpinner } from '../shared/index.tsx';
import { signInWithGoogle } from '../../services/firebaseService.ts';
import { useVaultModal } from '../../contexts/VaultModalContext.tsx';

const ServiceConnectionCard: React.FC<{
    serviceName: string;
    icon: React.ReactNode;
    fields: { id: string; label: string; placeholder: string }[];
    onConnect: (credentials: Record<string, string>) => Promise<void>;
    onDisconnect: () => Promise<void>;
    status: string;
    isLoading: boolean;
}> = ({ serviceName, icon, fields, onConnect, onDisconnect, status, isLoading })
=> {
    const [creds, setCreds] = useState<Record<string, string>>({});

    const handleConnect = () => {
        onConnect(creds);
```

```
    };

    const isConnected = status.startsWith('Connected');

    return (
        <div className="bg-surface border border-border rounded-lg p-6">
            <div className="flex items-center justify-between">
                <div className="flex items-center gap-4">
                    <div className="w-10 h-10">{icon}</div>
                    <div>
                        <h3 className="text-lg font-bold text-text-primary">{serviceName}</h3>
                        <p className={`text-sm ${isConnected ? 'text-green-600' : 'text-text-
                        secondary'}`}>{status}</p>
                    </div>
                </div>
                {isConnected && (
                    <button onClick={onDisconnect} className="px-4 py-2 bg-red-500/10 text-red-600
                    font-semibold rounded-lg hover:bg-red-500/20">
                        Disconnect
                    </button>
                )}
            </div>
            {!isConnected && (
                <div className="mt-4 pt-4 border-t border-border space-y-2">
                    {fields.map(field => (
                        <div key={field.id}>
                            <label className="text-xs text-text-secondary">{field.label}</label>
                            <input
                                type={field.id.includes('token') || field.id.includes('pat') ? 'password' :
                                'text'}
                                value={creds[field.id] || ''}
                                onChange={e => setCreds(prev => ({ ...prev, [field.id]: e.target.value }))}
                                placeholder={field.placeholder}
                                className="w-full mt-1 p-2 bg-background border border-border rounded-md text-
                                sm"
                            />
                        </div>
                    ))}
                    <button onClick={handleConnect} disabled={isLoading} className="btn-primary
                    w-full mt-2 py-2 flex items-center justify-center">
                        {isLoading ? <LoadingSpinner /> : 'Connect'}
                    </button>
                </div>
            )}
        </div>
    );
};


export const WorkspaceConnectorHub: React.FC = () => {
    const { state, dispatch } = useGlobalState();
    const { user, githubUser, vaultState } = state;
    const { addNotification } = useNotification();
    const { requestUnlock, requestCreation } = useVaultModal();
    const [loadingStates, setLoadingStates] = useState<Record<string, boolean>>({});
    const [connectionStatuses, setConnectionStatuses] = useState<Record<string,
    string>>({});

    // Manual action state
    const [selectedActionId, setSelectedActionId] =
    useState<string>([...ACTION_REGISTRY.keys()][0]);
    const [actionParams, setActionParams] = useState<Record<string, any>>({});
```

```
const [isExecuting, setIsExecuting] = useState(false);
const [actionResult, setActionResult] = useState<string>('');

const services = useMemo(() => {
    const serviceMap = new Map();
    ACTION_REGISTRY.forEach(action => {
        if (!serviceMap.has(action.service)) {
            serviceMap.set(action.service, {
                name: action.service,
                actions: [],
            });
        }
        serviceMap.get(action.service).actions.push(action);
    });
    return Array.from(serviceMap.values());
}, []);

const checkConnections = useCallback(async () => {
    if (!user || !vaultState.isUnlocked) return;

    const checkCred = async (credId: string, serviceName: string, successMessage:
    string) => {
         const token = await vaultService.getDecryptedCredential(credId);
         setConnectionStatuses(s => ({ ...s, [serviceName]: token ? successMessage : 'Not
         Connected' }));
    };

    await checkCred('github_pat', 'GitHub', githubUser ? `Connected as
    ${githubUser.login}`: 'Connected');
    await checkCred('jira_pat', 'Jira', 'Connected');
    await checkCred('slack_bot_token', 'Slack', 'Connected');

}, [user, vaultState.isUnlocked, githubUser]);

useEffect(() => {
    checkConnections();
}, [checkConnections]);

const withVault = useCallback(async (callback: () => Promise<void>) => {
    if (!vaultState.isInitialized) {
        const created = await requestCreation();
        if (!created) { addNotification('Vault setup is required.', 'error'); return; }
    }
    if (!vaultState.isUnlocked) {
        const unlocked = await requestUnlock();
        if (!unlocked) { addNotification('Vault must be unlocked to manage
        connections.', 'error'); return; }
    }
    await callback();
}, [vaultState, requestCreation, requestUnlock, addNotification]);


const handleConnect = async (serviceName: string, credentials: Record<string,
string>) => {
    await withVault(async () => {
        setLoadingStates(s => ({ ...s, [serviceName]: true }));
        try {
            for (const [key, value] of Object.entries(credentials)) {
                if (value) await vaultService.saveCredential(key, value);
            }
            if (serviceName === 'GitHub' && credentials.github_pat) {
                 const githubProfile = await validateToken(credentials.github_pat);
```

```
                dispatch({ type: 'SET_GITHUB_USER', payload: githubProfile });
                await vaultService.saveCredential('github_user', JSON.stringify(githubProfile));
            }
            addNotification(`${serviceName} connected successfully!`, 'success');
            checkConnections();
        } catch (e) {
            addNotification(`Failed to connect ${serviceName}: ${e instanceof Error ?
            e.message : 'Unknown error'}`, 'error');
        } finally {
            setLoadingStates(s => ({ ...s, [serviceName]: false }));
        }
    });
};

const handleDisconnect = async (serviceName: string, credIds: string[]) => {
   await withVault(async () => {
        setLoadingStates(s => ({ ...s, [serviceName]: true }));
        try {
            for (const id of credIds) {
                await vaultService.saveCredential(id, ''); // Overwrite with empty string
            }
             if (serviceName === 'GitHub') {
                dispatch({ type: 'SET_GITHUB_USER', payload: null });
                await vaultService.saveCredential('github_user', '');
            }
            addNotification(`${serviceName} disconnected.`, 'info');
            checkConnections();
        } catch(e) {
            addNotification(`Failed to disconnect ${serviceName}.`, 'error');
        } finally {
             setLoadingStates(s => ({ ...s, [serviceName]: false }));
        }
   });
};

const handleExecuteAction = async () => {
    await withVault(async () => {
        setIsExecuting(true);
        setActionResult('');
        try {
            const result = await executeWorkspaceAction(selectedActionId, actionParams);
            setActionResult(JSON.stringify(result, null, 2));
            addNotification('Action executed successfully!', 'success');
        } catch(e) {
            setActionResult(`Error: ${e instanceof Error ? e.message : 'Unknown Error'}`);
            addNotification('Action failed.', 'error');
        } finally {
            setIsExecuting(false);
        }
    });
};

const handleSignIn = async () => {
    setLoadingStates(s => ({...s, google: true}));
    try {
        await signInWithGoogle();
        addNotification('Signed in successfully!', 'success');
    } catch (error) {
        addNotification('Failed to sign in.', 'error');
    }
    setLoadingStates(s => ({...s, google: false}));
};
```

```
const selectedAction = ACTION_REGISTRY.get(selectedActionId);
const actionParameters = selectedAction ? selectedAction.getParameters() : {};

if (!user) {
    return (
        <div className="h-full flex items-center justify-center">
            <div className="text-center bg-surface p-8 rounded-lg border border-border max-
            w-md">
                <h2 className="text-xl font-bold">Sign In Required</h2>
                <p className="text-text-secondary my-4">Please sign in with your Google account
                to manage workspace connections.</p>
                <button onClick={handleSignIn} disabled={loadingStates.google} className="btn-
                primary px-6 py-3 flex items-center justify-center gap-2 mx-auto">
                    {loadingStates.google ? <LoadingSpinner/> : 'Sign in with Google'}
                </button>
            </div>
        </div>
    );
}

return (
    <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
         <header className="mb-8">
            <h1 className="text-4xl font-extrabold tracking-tight flex items-
            center"><RectangleGroupIcon /><span className="ml-3">Workspace Connector
            Hub</span></h1>
            <p className="mt-2 text-lg text-text-secondary">Connect to your development
            services to unlock cross-platform AI actions.</p>
        </header>
        <div className="flex-grow grid grid-cols-1 lg:grid-cols-2 gap-8 min-h-0">
            <div className="flex flex-col gap-6 overflow-y-auto pr-4">
                <h2 className="text-2xl font-bold">Service Connections</h2>
                <ServiceConnectionCard
                    serviceName="GitHub"
                    icon={<GithubIcon />}
                    fields={[{ id: 'github_pat', label: 'Personal Access Token', placeholder:
                    'ghp_...' }]}
                    onConnect={(creds) => handleConnect('GitHub', creds)}
                    onDisconnect={() => handleDisconnect('GitHub', ['github_pat'])}
                    status={connectionStatuses.GitHub || 'Checking...'}
                    isLoading={loadingStates.GitHub}
                />
                 {/* Placeholder cards for Jira and Slack */}
                <ServiceConnectionCard
                    serviceName="Jira"
                    icon={<div className="w-10 h-10 bg-[#0052CC] rounded flex items-center justify-
                    center text-white font-bold text-xl">J</div>}
                    fields={[
                        { id: 'jira_domain', label: 'Jira Domain', placeholder: 'your-
                        company.atlassian.net' },
                        { id: 'jira_email', label: 'Your Jira Email', placeholder: 'you@example.com' },
                        { id: 'jira_pat', label: 'API Token', placeholder: 'Your API Token' },
                    ]}
                    onConnect={(creds) => handleConnect('Jira', creds)}
                    onDisconnect={() => handleDisconnect('Jira', ['jira_domain', 'jira_email',
                    'jira_pat'])}
                    status={connectionStatuses.Jira || 'Checking...'}
                    isLoading={loadingStates.Jira}
                />
                <ServiceConnectionCard
                    serviceName="Slack"
                    icon={<div className="w-10 h-10 bg-[#4A154B] rounded flex items-center justify-
```

```
                        center text-white font-bold text-2xl">#</div>}
                        fields={[{ id: 'slack_bot_token', label: 'Bot User OAuth Token', placeholder:
                        'xoxb-...' }]}
                        onConnect={(creds) => handleConnect('Slack', creds)}
                        onDisconnect={() => handleDisconnect('Slack', ['slack_bot_token'])}
                        status={connectionStatuses.Slack || 'Checking...'}
                        isLoading={loadingStates.Slack}
                    />
                </div>
                <div className="flex flex-col gap-6 bg-surface p-6 border border-border rounded-
                lg">
                    <h2 className="text-2xl font-bold">Manual Action Runner</h2>
                    <div className="space-y-4">
                         <div>
                            <label className="text-sm font-medium">Action</label>
                            <select value={selectedActionId} onChange={e =>
                            setSelectedActionId(e.target.value)} className="w-full mt-1 p-2 bg-background
                            border rounded">
                                {services.map(service => (
                                    <optgroup label={service.name} key={service.name}>
                                        {service.actions.map((action: any) => (
                                            <option key={action.id} value={action.id}>{action.description}</option>
                                        ))}
                                    </optgroup>
                                ))}
                            </select>
                        </div>
                        {Object.entries(actionParameters).map(([key, param]: [string, any]) => (
                            <div key={key}>
                                <label className="text-sm font-medium">{key} {param.required && '*'}</label>
                                <input
                                    type={param.type}
                                    value={actionParams[key] || ''}
                                    onChange={e => setActionParams(p => ({...p, [key]: e.target.value}))}
                                    placeholder={param.default || ''}
                                    className="w-full mt-1 p-2 bg-background border rounded"
                                />
                            </div>
                        ))}
                        <button onClick={handleExecuteAction} disabled={isExecuting} className="btn-
                        primary w-full py-2 flex items-center justify-center gap-2">
                           {isExecuting ? <LoadingSpinner/> : <><SparklesIcon /> Execute Action</>}
                        </button>
                    </div>
                    <div>
                        <label className="text-sm font-medium">Result</label>
                        <pre className="w-full h-48 mt-1 p-2 bg-background border rounded overflow-auto
                        text-xs">{actionResult || 'Action results will appear here.'}</pre>
                    </div>
                </div>
            </div>
        </div>
    );
};


// ===== AiCommandCenter_49.tsx =====

import React, { useState, useCallback } from 'react';
import { Type, FunctionDeclaration } from "@google/genai";
import { logError } from '../../services/telemetryService.ts';
import { getInferenceFunction, CommandResponse } from
```

```
'../../services/aiService.ts';
import { FEATURE_TAXONOMY } from '../../services/taxonomyService.ts';
import { useGlobalState } from '../../contexts/GlobalStateContext.tsx';
import { CommandLineIcon } from '../icons.tsx';
import { LoadingSpinner } from '../shared/index.tsx';
import { ALL_FEATURE_IDS } from '../../constants.tsx';
import { executeWorkspaceAction, ACTION_REGISTRY } from
'../../services/workspaceConnectorService.ts';

const baseFunctionDeclarations: FunctionDeclaration[] = [
    {
        name: 'navigateTo',
        description: 'Navigates to a specific feature page.',
        parameters: {
            type: Type.OBJECT,
            properties: {
                featureId: {
                    type: Type.STRING,
                    description: 'The ID of the feature to navigate to.',
                    enum: ALL_FEATURE_IDS
                },
            },
            required: ['featureId'],
        },
    },
    {
        name: 'runFeatureWithInput',
        description: 'Navigates to a feature and passes initial data to it.',
        parameters: {
            type: Type.OBJECT,
            properties: {
                 featureId: {
                    type: Type.STRING,
                    description: 'The ID of the feature to run.',
                    enum: ALL_FEATURE_IDS
                },
                props: {
                    type: Type.OBJECT,
                    description: 'An object containing the initial properties for the feature, based
                    on its required inputs.',
                    properties: {
                        initialCode: { type: Type.STRING },
                        initialPrompt: { type: Type.STRING },
                        beforeCode: { type: Type.STRING },
                        afterCode: { type: Type.STRING },
                        logInput: { type: Type.STRING },
                        diff: { type: Type.STRING },
                        codeInput: { type: Type.STRING },
                        jsonInput: { type: Type.STRING },
                    }
                }
            },
            required: ['featureId', 'props']
        }
    }
];

// Dynamically add the workspace action
const functionDeclarations: FunctionDeclaration[] = [
    ...baseFunctionDeclarations,
    {
        name: 'runWorkspaceAction',
```

```
        description: 'Executes a defined action on a connected workspace service like
        Jira, Slack, or GitHub.',
        parameters: {
            type: Type.OBJECT,
            properties: {
                 actionId: {
                    type: Type.STRING,
                    description: 'The unique identifier for the action to execute.',
                    enum: [ ...ACTION_REGISTRY.keys() ]
                },
                params: {
                    type: Type.OBJECT,
                    description: 'An object containing the parameters for the action, matching its
                    required inputs.'
                }
            },
            required: ['actionId', 'params']
        }
    }
]

const knowledgeBase = FEATURE_TAXONOMY.map(f => `- ${f.name} (${f.id}):
${f.description} Inputs: ${f.inputs}`).join('\n');

const ExamplePromptButton: React.FC<{ text: string, onClick: (text: string) =>
void }> = ({ text, onClick }) => (
    <button
        onClick={() => onClick(text)}
        className="px-3 py-1.5 bg-surface border border-border rounded-full text-xs
        hover:bg-gray-100 dark:hover:bg-slate-700 transition-colors"
    >
        {text}
    </button>
)

export const AiCommandCenter: React.FC = () => {
    const { dispatch } = useGlobalState();
    const [prompt, setPrompt] = useState('');
    const [isLoading, setIsLoading] = useState(false);
    const [lastResponse, setLastResponse] = useState('');

    const handleCommand = useCallback(async () => {
        if (!prompt.trim()) return;

        setIsLoading(true);
        setLastResponse('');

        try {
            const response: CommandResponse = await getInferenceFunction(prompt,
            functionDeclarations, knowledgeBase);

            if (response.functionCalls && response.functionCalls.length > 0) {
                const call = response.functionCalls[0];
                const { name, args } = call;

                setLastResponse(`Understood! Executing command: ${name}`);

                switch (name) {
                    case 'navigateTo':
                        dispatch({ type: 'SET_VIEW', payload: { view: args.featureId }});
                        break;
                    case 'runFeatureWithInput':
```

```
                     dispatch({ type: 'SET_VIEW', payload: { view: args.featureId, props: args.props
                     } });
                    break;
                case 'runWorkspaceAction':
                    try {
                        const result = await executeWorkspaceAction(args.actionId, args.params);
                        setLastResponse(`Action '${args.actionId}' executed successfully.\n\nResult:
                        \`\`\`json\n${JSON.stringify(result, null, 2)}\n\`\`\``);
                    } catch (e) {
                        setLastResponse(`Action failed: ${e instanceof Error ? e.message : 'Unknown
                        error'}`);
                    }
                    break;
                default:
                    setLastResponse(`Unknown command: ${name}`);
            }
             setPrompt('');
        } else {
             setLastResponse(response.text);
        }

    } catch (err) {
        logError(err as Error, { prompt });
        setLastResponse(err instanceof Error ? err.message : 'An unknown error
        occurred.');
    } finally {
        setIsLoading(false);
    }
}, [prompt, dispatch]);

const handleKeyDown = (e: React.KeyboardEvent) => {
    if (e.key === 'Enter' && !e.shiftKey) {
        e.preventDefault();
        handleCommand();
    }
};

const handleExampleClick = (text: string) => {
    setPrompt(text);
}

return (
    <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
        <header className="mb-6 text-center">
            <h1 className="text-4xl font-extrabold tracking-tight flex items-center justify-
            center">
                <CommandLineIcon />
                <span className="ml-3">AI Command Center</span>
            </h1>
            <p className="mt-2 text-lg text-text-secondary">What would you like to do?</p>
        </header>

        <div className="flex-grow flex flex-col justify-end max-w-3xl w-full mx-auto">
            {lastResponse && (
                <div className="mb-4 p-4 bg-surface rounded-lg text-text-primary border border-
                border">
                    <p><strong>AI:</strong> {lastResponse}</p>
                </div>
            )}
             <div className="relative">
                <textarea
                    value={prompt}
```

```
                        onChange={e => setPrompt(e.target.value)}
                        onKeyDown={handleKeyDown}
                        disabled={isLoading}
                        placeholder='Try "explain this code: const a = 1;" or "open the theme designer"'
                        className="w-full p-4 pr-28 rounded-lg bg-surface border border-border
                        focus:ring-2 focus:ring-primary focus:outline-none resize-none shadow-sm"
                        rows={2}
                    />
                    <button
                        onClick={handleCommand}
                        disabled={isLoading}
                        className="btn-primary absolute right-3 top-1/2 -translate-y-1/2 px-4 py-2"
                    >
                       {isLoading ? <LoadingSpinner/> : 'Send'}
                    </button>
                </div>
                 <div className="flex flex-wrap items-center justify-center gap-2 mt-4">
                    <ExamplePromptButton text="Open Theme Designer" onClick={handleExampleClick} />
                    <ExamplePromptButton text="Generate a commit for a bug fix"
                    onClick={handleExampleClick} />
                    <ExamplePromptButton text="Create a regex for email validation"
                    onClick={handleExampleClick} />
                </div>
                 <p className="text-xs text-text-secondary text-center mt-2">Press Enter to send,
                 Shift+Enter for new line.</p>
            </div>
        </div>
    );
};

// ===== Connections_49.tsx =====

import React, { useState, useEffect, useMemo, useCallback } from 'react';
import { useGlobalState } from '../../contexts/GlobalStateContext.tsx';
import * as vaultService from '../../services/vaultService.ts';
import { useNotification } from '../../contexts/NotificationContext.tsx';
import { validateToken } from '../../services/authService.ts';
import { ACTION_REGISTRY, executeWorkspaceAction } from
'../../services/workspaceConnectorService.ts';
import { RectangleGroupIcon, GithubIcon, SparklesIcon } from '../icons.tsx';
import { LoadingSpinner } from '../shared/index.tsx';
import { signInWithGoogle } from '../../services/googleAuthService.ts';
import { useVaultModal } from '../../contexts/VaultModalContext.tsx';

const ServiceConnectionCard: React.FC<{
    serviceName: string;
    icon: React.ReactNode;
    fields: { id: string; label: string; placeholder: string }[];
    onConnect: (credentials: Record<string, string>) => Promise<void>;
    onDisconnect: () => Promise<void>;
    status: string;
    isLoading: boolean;
}> = ({ serviceName, icon, fields, onConnect, onDisconnect, status, isLoading })
=> {
    const [creds, setCreds] = useState<Record<string, string>>({});

    const handleConnect = () => {
        onConnect(creds);
    };

    const isConnected = status.startsWith('Connected');
    return (
```

```
        <div className="bg-surface border border-border rounded-lg p-6">
            <div className="flex items-center justify-between">
                <div className="flex items-center gap-4">
                    <div className="w-10 h-10">{icon}</div>
                    <div>
                        <h3 className="text-lg font-bold text-text-primary">{serviceName}</h3>
                        <p className={`text-sm ${isConnected ? 'text-green-600' : 'text-text-
                        secondary'}`}>{status}</p>
                    </div>
                </div>
                {isConnected && (
                    <button onClick={onDisconnect} className="px-4 py-2 bg-red-500/10 text-red-600
                    font-semibold rounded-lg hover:bg-red-500/20">
                        Disconnect
                    </button>
                )}
            </div>
            {!isConnected && (
                <div className="mt-4 pt-4 border-t border-border space-y-2">
                    {fields.map(field => (
                        <div key={field.id}>
                            <label className="text-xs text-text-secondary">{field.label}</label>
                            <input
                                type={field.id.includes('token') || field.id.includes('pat') ? 'password' :
                                'text'}
                                value={creds[field.id] || ''}
                                onChange={e => setCreds(prev => ({ ...prev, [field.id]: e.target.value }))}
                                placeholder={field.placeholder}
                                className="w-full mt-1 p-2 bg-background border border-border rounded-md text-
                                sm"
                            />
                        </div>
                    ))}
                    <button onClick={handleConnect} disabled={isLoading} className="btn-primary
                    w-full mt-2 py-2 flex items-center justify-center">
                        {isLoading ? <LoadingSpinner /> : 'Connect'}
                    </button>
                </div>
            )}
        </div>
    );
};


export const WorkspaceConnectorHub: React.FC = () => {
    const { state, dispatch } = useGlobalState();
    const { user, githubUser, vaultState } = state;
    const { addNotification } = useNotification();
    const { requestUnlock, requestCreation } = useVaultModal();
    const [loadingStates, setLoadingStates] = useState<Record<string, boolean>>({});
    const [connectionStatuses, setConnectionStatuses] = useState<Record<string,
    string>>({});

    // Manual action state
    const [selectedActionId, setSelectedActionId] =
    useState<string>([...ACTION_REGISTRY.keys()][0]);
    const [actionParams, setActionParams] = useState<Record<string, any>>({});
    const [isExecuting, setIsExecuting] = useState(false);
    const [actionResult, setActionResult] = useState<string>('');

    const services = useMemo(() => {
        const serviceMap = new Map();
```

```
    ACTION_REGISTRY.forEach(action => {
        if (!serviceMap.has(action.service)) {
            serviceMap.set(action.service, {
                name: action.service,
                actions: [],
            });
        }
        serviceMap.get(action.service).actions.push(action);
    });
    return Array.from(serviceMap.values());
}, []);

const checkConnections = useCallback(async () => {
    if (!user || !vaultState.isUnlocked) return;

    const checkCred = async (credId: string, serviceName: string, successMessage:
    string) => {
         const token = await vaultService.getDecryptedCredential(credId);
         setConnectionStatuses(s => ({ ...s, [serviceName]: token ? successMessage : 'Not
         Connected' }));
    };

    await checkCred('github_pat', 'GitHub', githubUser ? `Connected as
    ${githubUser.login}`: 'Connected');
    await checkCred('jira_pat', 'Jira', 'Connected');
    await checkCred('slack_bot_token', 'Slack', 'Connected');

}, [user, vaultState.isUnlocked, githubUser]);

useEffect(() => {
    checkConnections();
}, [checkConnections]);

const withVault = useCallback(async (callback: () => Promise<void>) => {
    if (!vaultState.isInitialized) {
        const created = await requestCreation();
        if (!created) { addNotification('Vault setup is required.', 'error'); return; }
    }
    if (!vaultState.isUnlocked) {
        const unlocked = await requestUnlock();
        if (!unlocked) { addNotification('Vault must be unlocked to manage
        connections.', 'error'); return; }
    }
    await callback();
}, [vaultState, requestCreation, requestUnlock, addNotification]);


const handleConnect = async (serviceName: string, credentials: Record<string,
string>) => {
    await withVault(async () => {
        setLoadingStates(s => ({ ...s, [serviceName]: true }));
        try {
            for (const [key, value] of Object.entries(credentials)) {
                if (value) await vaultService.saveCredential(key, value);
            }
            if (serviceName === 'GitHub' && credentials.github_pat) {
                 const githubProfile = await validateToken(credentials.github_pat);
                 dispatch({ type: 'SET_GITHUB_USER', payload: githubProfile });
                 await vaultService.saveCredential('github_user', JSON.stringify(githubProfile));
            }
            addNotification(`${serviceName} connected successfully!`, 'success');
            checkConnections();
```

```
        } catch (e) {
            addNotification(`Failed to connect ${serviceName}: ${e instanceof Error ?
            e.message : 'Unknown error'}`, 'error');
        } finally {
            setLoadingStates(s => ({ ...s, [serviceName]: false }));
        }
    });
};

const handleDisconnect = async (serviceName: string, credIds: string[]) => {
   await withVault(async () => {
        setLoadingStates(s => ({ ...s, [serviceName]: true }));
        try {
            for (const id of credIds) {
                 await vaultService.saveCredential(id, ''); // Overwrite with empty string
            }
             if (serviceName === 'GitHub') {
                 dispatch({ type: 'SET_GITHUB_USER', payload: null });
                 await vaultService.saveCredential('github_user', '');
            }
            addNotification(`${serviceName} disconnected.`, 'info');
            checkConnections();
        } catch(e) {
            addNotification(`Failed to disconnect ${serviceName}.`, 'error');
        } finally {
             setLoadingStates(s => ({ ...s, [serviceName]: false }));
        }
   });
};

const handleExecuteAction = async () => {
    await withVault(async () => {
        setIsExecuting(true);
        setActionResult('');
        try {
            const result = await executeWorkspaceAction(selectedActionId, actionParams);
            setActionResult(JSON.stringify(result, null, 2));
            addNotification('Action executed successfully!', 'success');
        } catch(e) {
            setActionResult(`Error: ${e instanceof Error ? e.message : 'Unknown Error'}`);
            addNotification('Action failed.', 'error');
        } finally {
            setIsExecuting(false);
        }
    });
};

const handleSignIn = () => {
    signInWithGoogle();
    // The result is handled by the global callback set in App.tsx
};

const selectedAction = ACTION_REGISTRY.get(selectedActionId);
const actionParameters = selectedAction ? selectedAction.getParameters() : {};

if (!user) {
    return (
        <div className="h-full flex items-center justify-center">
            <div className="text-center bg-surface p-8 rounded-lg border border-border max-
            w-md">
                <h2 className="text-xl font-bold">Sign In Required</h2>
                <p className="text-text-secondary my-4">Please sign in with your Google account
```

```
                to manage workspace connections.</p>
                <button onClick={handleSignIn} disabled={loadingStates.google} className="btn-
                primary px-6 py-3 flex items-center justify-center gap-2 mx-auto">
                    {loadingStates.google ? <LoadingSpinner/> : 'Sign in with Google'}
                </button>
            </div>
        </div>
    );
}

return (
    <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
         <header className="mb-8">
            <h1 className="text-4xl font-extrabold tracking-tight flex items-
            center"><RectangleGroupIcon /><span className="ml-3">Workspace Connector
            Hub</span></h1>
            <p className="mt-2 text-lg text-text-secondary">Connect to your development
            services to unlock cross-platform AI actions.</p>
        </header>
        <div className="flex-grow grid grid-cols-1 lg:grid-cols-2 gap-8 min-h-0">
            <div className="flex flex-col gap-6 overflow-y-auto pr-4">
                <h2 className="text-2xl font-bold">Service Connections</h2>
                <ServiceConnectionCard
                    serviceName="GitHub"
                    icon={<GithubIcon />}
                    fields={[{ id: 'github_pat', label: 'Personal Access Token', placeholder:
                    'ghp_...' }]}
                    onConnect={(creds) => handleConnect('GitHub', creds)}
                    onDisconnect={() => handleDisconnect('GitHub', ['github_pat'])}
                    status={connectionStatuses.GitHub || 'Checking...'}
                    isLoading={loadingStates.GitHub}
                />
                 {/* Placeholder cards for Jira and Slack */}
                <ServiceConnectionCard
                    serviceName="Jira"
                    icon={<div className="w-10 h-10 bg-[#0052CC] rounded flex items-center justify-
                    center text-white font-bold text-xl">J</div>}
                    fields={[
                        { id: 'jira_domain', label: 'Jira Domain', placeholder: 'your-
                        company.atlassian.net' },
                        { id: 'jira_email', label: 'Your Jira Email', placeholder: 'you@example.com' },
                        { id: 'jira_pat', label: 'API Token', placeholder: 'Your API Token' },
                    ]}
                    onConnect={(creds) => handleConnect('Jira', creds)}
                    onDisconnect={() => handleDisconnect('Jira', ['jira_domain', 'jira_email',
                    'jira_pat'])}
                    status={connectionStatuses.Jira || 'Checking...'}
                    isLoading={loadingStates.Jira}
                />
                <ServiceConnectionCard
                    serviceName="Slack"
                    icon={<div className="w-10 h-10 bg-[#4A154B] rounded flex items-center justify-
                    center text-white font-bold text-2xl">#</div>}
                    fields={[{ id: 'slack_bot_token', label: 'Bot User OAuth Token', placeholder:
                    'xoxb-...' }]}
                    onConnect={(creds) => handleConnect('Slack', creds)}
                    onDisconnect={() => handleDisconnect('Slack', ['slack_bot_token'])}
                    status={connectionStatuses.Slack || 'Checking...'}
                    isLoading={loadingStates.Slack}
                />
            </div>
            <div className="flex flex-col gap-6 bg-surface p-6 border border-border rounded-
```

```
                lg">
                    <h2 className="text-2xl font-bold">Manual Action Runner</h2>
                    <div className="space-y-4">
                         <div>
                            <label className="text-sm font-medium">Action</label>
                            <select value={selectedActionId} onChange={e =>
                            setSelectedActionId(e.target.value)} className="w-full mt-1 p-2 bg-background
                            border rounded">
                                {services.map(service => (
                                    <optgroup label={service.name} key={service.name}>
                                        {service.actions.map((action: any) => (
                                            <option key={action.id} value={action.id}>{action.description}</option>
                                        ))}
                                    </optgroup>
                                ))}
                            </select>
                        </div>
                        {Object.entries(actionParameters).map(([key, param]: [string, any]) => (
                            <div key={key}>
                                <label className="text-sm font-medium">{key} {param.required && '*'}</label>
                                <input
                                    type={param.type}
                                    value={actionParams[key] || ''}
                                    onChange={e => setActionParams(p => ({...p, [key]: e.target.value}))}
                                    placeholder={param.default || ''}
                                    className="w-full mt-1 p-2 bg-background border rounded"
                                />
                            </div>
                        ))}
                        <button onClick={handleExecuteAction} disabled={isExecuting} className="btn-
                        primary w-full py-2 flex items-center justify-center gap-2">
                           {isExecuting ? <LoadingSpinner/> : <><SparklesIcon /> Execute Action</>}
                        </button>
                    </div>
                    <div>
                        <label className="text-sm font-medium">Result</label>
                        <pre className="w-full h-48 mt-1 p-2 bg-background border rounded overflow-auto
                        text-xs">{actionResult || 'Action results will appear here.'}</pre>
                    </div>
                </div>
            </div>
        </div>
    );
};

// ===== MarkdownSlides_51.tsx =====



import React, { useState, useMemo, useEffect, useRef, useCallback } from
'react';
import { marked } from 'marked';
import { PhotoIcon } from '../icons.tsx';

const exampleMarkdown = `# Slide 1: Welcome

This is a slide deck generated from Markdown.

- Use standard markdown syntax
- Like lists, headers, and **bold** text.

---
```

```
# Slide 2: Features

Navigate using the buttons below.

\`\`\`javascript
console.log("Code blocks work too!");
\`\`\`

---

# Slide 3: The End

Easy to create and present.
`;

export const MarkdownSlides: React.FC = () => {
    const [markdown, setMarkdown] = useState(exampleMarkdown);
    const [currentSlide, setCurrentSlide] = useState(0);
    const [slideHtml, setSlideHtml] = useState<string | TrustedHTML>('');
    const presentationRef = useRef<HTMLDivElement>(null);

    const slides = useMemo(() => markdown.split(/^-{3,}\s*$/m), [markdown]);

    useEffect(() => {
        const parse = async () => {
            const currentSlideContent = slides[currentSlide] || '';
            const html = await marked.parse(currentSlideContent);
            setSlideHtml(html);
        };
        parse();
    }, [slides, currentSlide]);

    const goToNext = useCallback(() => setCurrentSlide(s => Math.min(s + 1,
    slides.length - 1)), [slides.length]);
    const goToPrev = useCallback(() => setCurrentSlide(s => Math.max(s - 1, 0)),
    []);

    const handleFullscreen = () => {
        presentationRef.current?.requestFullscreen();
    };

    useEffect(() => {
        const handleKeyDown = (e: KeyboardEvent) => {
            if (document.fullscreenElement === presentationRef.current) {
                if (e.key === 'ArrowRight' || e.key === ' ') goToNext();
                if (e.key === 'ArrowLeft') goToPrev();
            }
        };
        document.addEventListener('keydown', handleKeyDown);
        return () => document.removeEventListener('keydown', handleKeyDown);
    }, [goToNext, goToPrev]);

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
            <header className="mb-6">
                <h1 className="text-3xl font-bold flex items-center"><PhotoIcon /><span
                classNamе="ml-3">Markdown to Slides</span></h1>
                <p className="text-text-secondary mt-1">Write markdown, present it as a
                slideshow. Use '---' to separate slides.</p>
            </header>
            <div className="flex-grow grid grid-cols-1 lg:grid-cols-2 gap-6 h-full overflow-
            hidden">
```

```
                <div className="flex flex-col h-full">
                     <label htmlFor="md-input" className="text-sm font-medium text-text-secondary
                     mb-2">Markdown Editor</label>
                     <textarea id="md-input" value={markdown} onChange={e =>
                     setMarkdown(e.target.value)} className="flex-grow p-4 bg-surface border border-
                     border rounded-md resize-none font-mono text-sm focus:ring-2 focus:ring-primary
                     focus:outline-none"/>
                </div>
                 <div ref={presentationRef} className="flex flex-col h-full bg-surface
                 fullscreen:bg-background border border-border rounded-md">
                    <div className="flex-shrink-0 flex justify-end items-center p-2 border-b border-
                    border gap-2">
                        <button onClick={handleFullscreen} className="px-3 py-1 bg-gray-100 dark:bg-
                        slate-700 rounded-md text-xs hover:bg-gray-200 dark:hover:bg-
                        slate-600">Fullscreen</button>
                    </div>
                    <div className="relative flex-grow flex flex-col justify-center items-center p-8
                    overflow-y-auto">
                        <div className="prose prose-lg max-w-none w-full" dangerouslySetInnerHTML={{
                        __html: slideHtml }} />
                         <button onClick={goToPrev} disabled={currentSlide === 0} className="absolute
                         left-4 top-1/2 -translate-y-1/2 p-2 bg-gray-200/50 dark:bg-slate-700/50 rounded-
                         full disabled:opacity-30 hover:bg-gray-300/50 dark:hover:bg-
                         slate-600/50">■</button>
                         <button onClick={goToNext} disabled={currentSlide === slides.length - 1}
                         className="absolute right-4 top-1/2 -translate-y-1/2 p-2 bg-gray-200/50 dark:bg-
                         slate-700/50 rounded-full disabled:opacity-30 hover:bg-gray-300/50
                         dark:hover:bg-slate-600/50">■</button>
                         <div className="absolute bottom-4 right-4 text-xs bg-black/50 px-2 py-1 rounded-
                         md text-white">
                            {currentSlide + 1} / {slides.length}
                         </div>
                    </div>
                </div>
            </div>
        </div>
    );
};

// ===== WeeklyDigestGenerator_30.tsx =====

import React, { useState, useCallback } from 'react';
import { generateWeeklyDigest } from '../../services/index.ts';
import { useNotification } from '../../contexts/NotificationContext.tsx';
import { MailIcon, SparklesIcon } from '../icons.tsx';
import { LoadingSpinner } from '../shared/index.tsx';

// Dummy data for demonstration purposes
const dummyCommitLogs = `
feat: implement user authentication
fix: resolve issue with button alignment
feat: add dark mode toggle
chore: update dependencies
refactor: simplify data fetching logic
`;
const dummyTelemetry = {
    avgPageLoad: 120,
    errorRate: '0.5%',
    uptime: '99.98%'
};

export const WeeklyDigestGenerator: React.FC = () => {
```

```
    const { addNotification } = useNotification();
    const [emailHtml, setEmailHtml] = useState('');
    const [isLoading, setIsLoading] = useState(false);

    const handleGenerate = useCallback(async () => {
        setIsLoading(true);
        setEmailHtml('');
        try {
            const html = await generateWeeklyDigest(dummyCommitLogs, dummyTelemetry);
            setEmailHtml(html);
            addNotification('Digest content generated!', 'success');
        } catch (e) {
            addNotification(e instanceof Error ? e.message : 'Failed to generate digest',
            'error');
        } finally {
            setIsLoading(false);
        }
    }, [addNotification]);


    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
            <header className="mb-6">
                <h1 className="text-3xl font-bold flex items-center"><MailIcon /><span
                className="ml-3">Weekly Digest Generator</span></h1>
                <p className="text-text-secondary mt-1">Generate an AI-powered weekly summary
                based on project data.</p>
            </header>

            <div className="flex-grow grid grid-cols-1 lg:grid-cols-2 gap-6 min-h-0">
                <div className="bg-surface p-4 border border-border rounded-lg flex flex-col
                items-center justify-center text-center">
                    <h3 className="text-lg font-bold">Generate Digest</h3>
                    <p className="text-sm text-text-secondary my-4">This tool will use dummy data to
                    generate a project summary. The send functionality has been removed due to
                    updated permissions.</p>
                    <div className="flex flex-col gap-4 w-full max-w-xs">
                        <button onClick={handleGenerate} disabled={isLoading} className="btn-primary
                        flex items-center justify-center gap-2 py-3">
                            {isLoading ? <LoadingSpinner /> : <><SparklesIcon /> Generate Digest</>}
                        </button>
                    </div>
                </div>

                <div className="bg-surface p-4 border border-border rounded-lg flex flex-col">
                    <h3 className="text-lg font-bold mb-2">Email Preview</h3>
                    <div className="flex-grow bg-white border rounded overflow-hidden">
                        {isLoading && <div className="flex justify-center items-center
                        h-full"><LoadingSpinner /></div>}
                        {emailHtml && <iframe srcDoc={emailHtml} title="Email Preview" className="w-full
                        h-full" />}
                        {!isLoading && !emailHtml && <div className="flex justify-center items-center
                        h-full text-text-secondary">Preview will appear here.</div>}
                    </div>
                </div>
            </div>
        </div>
    );
};

// ===== WorkspaceConnectorHub_18.tsx =====
import React, { useState, useEffect, useMemo, useCallback } from 'react';
```

```
import { useGlobalState } from '../../contexts/GlobalStateContext.tsx';
import * as vaultService from '../../services/vaultService.ts';
import { useNotification } from '../../contexts/NotificationContext.tsx';
import { validateToken } from '../../services/authService.ts';
import { ACTION_REGISTRY, executeWorkspaceAction } from
'../../services/workspaceConnectorService.ts';
import { RectangleGroupIcon, GithubIcon, SparklesIcon } from '../icons.tsx';
import { LoadingSpinner } from '../shared/index.tsx';
import { signInWithGoogle } from '../../services/googleAuthService.ts';
import { useVaultModal } from '../../contexts/VaultModalContext.tsx';

const ServiceConnectionCard: React.FC<{
    serviceName: string;
    icon: React.ReactNode;
    fields: { id: string; label: string; placeholder: string }[];
    onConnect: (credentials: Record<string, string>) => Promise<void>;
    onDisconnect: () => Promise<void>;
    status: string;
    isLoading: boolean;
}> = ({ serviceName, icon, fields, onConnect, onDisconnect, status, isLoading })
=> {
    const [creds, setCreds] = useState<Record<string, string>>({});

    const handleConnect = () => {
        onConnect(creds);
    };

    const isConnected = status.startsWith('Connected');

    return (
        <div className="bg-surface border border-border rounded-lg p-6">
            <div className="flex items-center justify-between">
                <div className="flex items-center gap-4">
                    <div className="w-10 h-10">{icon}</div>
                    <div>
                        <h3 className="text-lg font-bold text-text-primary">{serviceName}</h3>
                        <p className={`text-sm ${isConnected ? 'text-green-600' : 'text-text-
                        secondary'}`}>{status}</p>
                    </div>
                </div>
                {isConnected && (
                    <button onClick={onDisconnect} className="px-4 py-2 bg-red-500/10 text-red-600
                    font-semibold rounded-lg hover:bg-red-500/20">
                        Disconnect
                    </button>
                )}
            </div>
            {!isConnected && (
                <div className="mt-4 pt-4 border-t border-border space-y-2">
                    {fields.map(field => (
                        <div key={field.id}>
                            <label className="text-xs text-text-secondary">{field.label}</label>
                            <input
                                type={field.id.includes('token') || field.id.includes('pat') ? 'password' :
                                'text'}
                                value={creds[field.id] || ''}
                                onChange={e => setCreds(prev => ({ ...prev, [field.id]: e.target.value }))}
                                placeholder={field.placeholder}
                                className="w-full mt-1 p-2 bg-background border border-border rounded-md text-
                                sm"
                            />
                        </div>
```

```
                    ))}
                    <button onClick={handleConnect} disabled={isLoading} className="btn-primary
                    w-full mt-2 py-2 flex items-center justify-center">
                        {isLoading ? <LoadingSpinner /> : 'Connect'}
                    </button>
                </div>
            )}
        </div>
    );
};


export const WorkspaceConnectorHub: React.FC = () => {
    const { state, dispatch } = useGlobalState();
    const { user, githubUser, vaultState } = state;
    const { addNotification } = useNotification();
    const { requestUnlock, requestCreation } = useVaultModal();
    const [loadingStates, setLoadingStates] = useState<Record<string, boolean>>({});
    const [connectionStatuses, setConnectionStatuses] = useState<Record<string,
    string>>({});

    // Manual action state
    const [selectedActionId, setSelectedActionId] =
    useState<string>([...ACTION_REGISTRY.keys()][0]);
    const [actionParams, setActionParams] = useState<Record<string, any>>({});
    const [isExecuting, setIsExecuting] = useState(false);
    const [actionResult, setActionResult] = useState<string>('');

    const services = useMemo(() => {
        const serviceMap = new Map();
        ACTION_REGISTRY.forEach(action => {
            if (!serviceMap.has(action.service)) {
                serviceMap.set(action.service, {
                    name: action.service,
                    actions: [],
                });
            }
            serviceMap.get(action.service).actions.push(action);
        });
        return Array.from(serviceMap.values());
    }, []);

    const checkConnections = useCallback(async () => {
        if (!user || !vaultState.isUnlocked) return;

        const checkCred = async (credId: string, serviceName: string, successMessage:
        string) => {
             const token = await vaultService.getDecryptedCredential(credId);
             setConnectionStatuses(s => ({ ...s, [serviceName]: token ? successMessage : 'Not
             Connected' }));
        };

        await checkCred('github_pat', 'GitHub', githubUser ? `Connected as
        ${githubUser.login}`: 'Connected');
        await checkCred('jira_pat', 'Jira', 'Connected');
        await checkCred('slack_bot_token', 'Slack', 'Connected');

    }, [user, vaultState.isUnlocked, githubUser]);

    useEffect(() => {
        checkConnections();
    }, [checkConnections]);
```

```
const withVault = useCallback(async (callback: () => Promise<void>) => {
    if (!vaultState.isInitialized) {
        const created = await requestCreation();
        if (!created) { addNotification('Vault setup is required.', 'error'); return; }
    }
    if (!vaultState.isUnlocked) {
        const unlocked = await requestUnlock();
        if (!unlocked) { addNotification('Vault must be unlocked to manage
        connections.', 'error'); return; }
    }
    await callback();
}, [vaultState, requestCreation, requestUnlock, addNotification]);


const handleConnect = async (serviceName: string, credentials: Record<string,
string>) => {
    await withVault(async () => {
        setLoadingStates(s => ({ ...s, [serviceName]: true }));
        try {
            for (const [key, value] of Object.entries(credentials)) {
                if (value) await vaultService.saveCredential(key, value);
            }
            if (serviceName === 'GitHub' && credentials.github_pat) {
                 const githubProfile = await validateToken(credentials.github_pat);
                 dispatch({ type: 'SET_GITHUB_USER', payload: githubProfile });
                 await vaultService.saveCredential('github_user', JSON.stringify(githubProfile));
            }
            addNotification(`${serviceName} connected successfully!`, 'success');
            checkConnections();
        } catch (e) {
            addNotification(`Failed to connect ${serviceName}: ${e instanceof Error ?
            e.message : 'Unknown error'}`, 'error');
        } finally {
            setLoadingStates(s => ({ ...s, [serviceName]: false }));
        }
    });
};

const handleDisconnect = async (serviceName: string, credIds: string[]) => {
   await withVault(async () => {
        setLoadingStates(s => ({ ...s, [serviceName]: true }));
        try {
            for (const id of credIds) {
                 await vaultService.saveCredential(id, ''); // Overwrite with empty string
            }
             if (serviceName === 'GitHub') {
                 dispatch({ type: 'SET_GITHUB_USER', payload: null });
                 await vaultService.saveCredential('github_user', '');
            }
            addNotification(`${serviceName} disconnected.`, 'info');
            checkConnections();
        } catch(e) {
            addNotification(`Failed to disconnect ${serviceName}.`, 'error');
        } finally {
             setLoadingStates(s => ({ ...s, [serviceName]: false }));
        }
   });
};

const handleExecuteAction = async () => {
    await withVault(async () => {
        setIsExecuting(true);
```

```
        setActionResult('');
        try {
            const result = await executeWorkspaceAction(selectedActionId, actionParams);
            setActionResult(JSON.stringify(result, null, 2));
            addNotification('Action executed successfully!', 'success');
        } catch(e) {
            setActionResult(`Error: ${e instanceof Error ? e.message : 'Unknown Error'}`);
            addNotification('Action failed.', 'error');
        } finally {
            setIsExecuting(false);
        }
    });
};

const handleSignIn = () => {
    signInWithGoogle();
    // The result is handled by the global callback set in App.tsx
};

const selectedAction = ACTION_REGISTRY.get(selectedActionId);
const actionParameters = selectedAction ? selectedAction.getParameters() : {};

if (!user) {
    return (
        <div className="h-full flex items-center justify-center">
            <div className="text-center bg-surface p-8 rounded-lg border border-border max-
            w-md">
                <h2 className="text-xl font-bold">Sign In Required</h2>
                <p className="text-text-secondary my-4">Please sign in with your Google account
                to manage workspace connections.</p>
                <button onClick={handleSignIn} className="btn-primary px-6 py-3 flex items-
                center justify-center gap-2 mx-auto">
                    Sign in with Google
                </button>
            </div>
        </div>
    );
}

return (
    <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
         <header className="mb-8">
            <h1 className="text-4xl font-extrabold tracking-tight flex items-
            center"><RectangleGroupIcon /><span className="ml-3">Workspace Connector
            Hub</span></h1>
            <p className="mt-2 text-lg text-text-secondary">Connect to your development
            services to unlock cross-platform AI actions.</p>
        </header>
        <div className="flex-grow grid grid-cols-1 lg:grid-cols-2 gap-8 min-h-0">
            <div className="flex flex-col gap-6 overflow-y-auto pr-4">
                <h2 className="text-2xl font-bold">Service Connections</h2>
                <ServiceConnectionCard
                    serviceName="GitHub"
                    icon={<GithubIcon />}
                    fields={[{ id: 'github_pat', label: 'Personal Access Token', placeholder:
                    'ghp_...' }]}
                    onConnect={(creds) => handleConnect('GitHub', creds)}
                    onDisconnect={() => handleDisconnect('GitHub', ['github_pat'])}
                    status={connectionStatuses.GitHub || 'Checking...'}
                    isLoading={loadingStates.GitHub}
                />
                 {/* Placeholder cards for Jira and Slack */}
```

```
    <ServiceConnectionCard
        serviceName="Jira"
        icon={<div className="w-10 h-10 bg-[#0052CC] rounded flex items-center justify-
        center text-white font-bold text-xl">J</div>}
        fields={[
            { id: 'jira_domain', label: 'Jira Domain', placeholder: 'your-
            company.atlassian.net' },
            { id: 'jira_email', label: 'Your Jira Email', placeholder: 'you@example.com' },
            { id: 'jira_pat', label: 'API Token', placeholder: 'Your API Token' },
        ]}
        onConnect={(creds) => handleConnect('Jira', creds)}
        onDisconnect={() => handleDisconnect('Jira', ['jira_domain', 'jira_email',
        'jira_pat'])}
        status={connectionStatuses.Jira || 'Checking...'}
        isLoading={loadingStates.Jira}
    />
    <ServiceConnectionCard
        serviceName="Slack"
        icon={<div className="w-10 h-10 bg-[#4A154B] rounded flex items-center justify-
        center text-white font-bold text-2xl">#</div>}
        fields={[{ id: 'slack_bot_token', label: 'Bot User OAuth Token', placeholder:
        'xoxb-...' }]}
        onConnect={(creds) => handleConnect('Slack', creds)}
        onDisconnect={() => handleDisconnect('Slack', ['slack_bot_token'])}
        status={connectionStatuses.Slack || 'Checking...'}
        isLoading={loadingStates.Slack}
    />
</div>
<div className="flex flex-col gap-6 bg-surface p-6 border border-border rounded-
lg">
    <h2 className="text-2xl font-bold">Manual Action Runner</h2>
    <div className="space-y-4">
         <div>
            <label className="text-sm font-medium">Action</label>
            <select value={selectedActionId} onChange={e =>
            setSelectedActionId(e.target.value)} className="w-full mt-1 p-2 bg-background
            border rounded">
                {services.map(service => (
                    <optgroup label={service.name} key={service.name}>
                        {service.actions.map((action: any) => (
                            <option key={action.id} value={action.id}>{action.description}</option>
                        ))}
                    </optgroup>
                ))}
            </select>
        </div>
        {Object.entries(actionParameters).map(([key, param]: [string, any]) => (
            <div key={key}>
                <label className="text-sm font-medium">{key} {param.required && '*'}</label>
                <input
                    type={param.type}
                    value={actionParams[key] || ''}
                    onChange={e => setActionParams(p => ({...p, [key]: e.target.value}))}
                    placeholder={param.default || ''}
                    className="w-full mt-1 p-2 bg-background border rounded"
                />
            </div>
        ))}
        <button onClick={handleExecuteAction} disabled={isExecuting} className="btn-
        primary w-full py-2 flex items-center justify-center gap-2">
           {isExecuting ? <LoadingSpinner/> : <><SparklesIcon /> Execute Action</>}
        </button>
```

```
                    </div>
                    <div>
                        <label className="text-sm font-medium">Result</label>
                        <pre className="w-full h-48 mt-1 p-2 bg-background border rounded overflow-auto
                        text-xs">{actionResult || 'Action results will appear here.'}</pre>
                    </div>
                </div>
            </div>
        </div>
    );
};

// ===== GmailAddonSimulator.tsx =====

import React, { useState, useCallback } from 'react';
import { streamContent } from '../../services/aiService.ts';
import { MailIcon, SparklesIcon, XMarkIcon } from '../icons.tsx';
import { LoadingSpinner, MarkdownRenderer } from '../shared/index.tsx';

const mockEmail = {
    from: 'Alice <alice@example.com>',
    to: 'Me <me@example.com>',
    subject: 'Project Update & Question',
    body: `Hey,

Just wanted to give you a quick update. The new user authentication flow is
complete and pushed to the staging server.

I had a question about the next task regarding the database migration. The
ticket says we need to migrate the 'users' table, but it's not clear on the
required schema changes. Should I just add the new 'last_login' column or are
there other modifications needed?

Let me know when you have a chance.

Thanks,
Alice`
};

export const GmailAddonSimulator: React.FC = () => {
    const [isComposeOpen, setComposeOpen] = useState(false);
    const [generatedReply, setGeneratedReply] = useState('');
    const [isGenerating, setIsGenerating] = useState(false);

    const handleGenerateReply = useCallback(async () => {
        setIsGenerating(true);
        setGeneratedReply('');
        setComposeOpen(true);
        try {
            const prompt = `Generate a professional and friendly reply to the following
            email. Acknowledge the update and answer the question by stating that only the
            'last_login' column (as a DATETIME) is needed for
            now.\n\nEMAIL:\n${mockEmail.body}`;
            const stream = streamContent(prompt, "You are a helpful assistant writing a
            professional email reply.", 0.7);
            let fullResponse = '';
            for await (const chunk of stream) {
                fullResponse += chunk;
                setGeneratedReply(fullResponse);
            }
        } catch(e) {
            setGeneratedReply(`Error: ${e instanceof Error ? e.message : 'Could not generate
```

```
        reply.'}`);
    }
    finally {
        setIsGenerating(false);
    }
}, []);

const ComposeModal = () => (
    <div className="absolute inset-0 bg-black/30 backdrop-blur-sm flex justify-
    center items-center p-4 z-20">
        <div className="w-full max-w-2xl h-[70vh] bg-surface rounded-lg shadow-2xl flex
        flex-col animate-pop-in">
            <header className="flex justify-between items-center p-3 bg-gray-100 dark:bg-
            slate-700 rounded-t-lg">
                <h3 className="font-semibold text-sm">New Message</h3>
                <button onClick={() => setComposeOpen(false)}><XMarkIcon /></button>
            </header>
            <div className="p-3 text-sm border-b border-border">
                <p><span className="text-text-secondary">To:</span> {mockEmail.from}</p>
            </div>
            <div className="p-3 text-sm border-b border-border">
                 <p><span className="text-text-secondary">Subject:</span> Re:
                 {mockEmail.subject}</p>
            </div>
            <div className="flex-grow p-3 overflow-y-auto">
                {isGenerating ? <div className="flex justify-center items-center
                h-full"><LoadingSpinner /></div> : <MarkdownRenderer content={generatedReply}
                />}
            </div>
             <footer className="p-3 border-t border-border">
                <button className="btn-primary px-6 py-2">Send</button>
             </footer>
        </div>
    </div>
);

return (
    <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
        <header className="mb-6">
            <h1 className="text-3xl font-bold flex items-center"><MailIcon /><span
            className="ml-3">Gmail Add-on Simulator</span></h1>
            <p className="text-text-secondary mt-1">A simulation of how contextual add-on
            scopes would work inside Gmail.</p>
        </header>
        <div className="relative flex-grow bg-surface border-2 border-dashed border-
        border rounded-lg p-6 flex items-center justify-center">
            {isComposeOpen && <ComposeModal />}
            <div className="w-full max-w-4xl h-full bg-white dark:bg-slate-800 rounded-xl
            shadow-2xl flex flex-col overflow-hidden">
                {/* Header */}
                <div className="flex-shrink-0 p-4 border-b border-border">
                    <h2 className="text-xl font-bold">{mockEmail.subject}</h2>
                    <div className="flex items-center gap-2 text-sm mt-2">
                         <img src="https://avatar.vercel.sh/alice" alt="Alice" className="w-8 h-8
                         rounded-full" />
                         <div>
                            <p className="font-semibold">{mockEmail.from.split('<')[0].trim()}</p>
                            <p className="text-text-secondary text-xs">to
                            {mockEmail.to.split('<')[0].trim()}</p>
                         </div>
                    </div>
                </div>
```

```
                    {/* Body */}
                    <div className="flex-grow p-4 overflow-y-auto">
                        <pre className="whitespace-pre-wrap font-sans text-sm">{mockEmail.body}</pre>
                    </div>
                    {/* Actions */}
                    <div className="flex-shrink-0 p-4 border-t border-border bg-gray-50 dark:bg-
                    slate-900/50 flex justify-between items-center">
                        <div className="text-xs text-text-secondary">
                            <strong>Disclaimer:</strong> This is a simulation. The requested scopes allow
                            this app to read the current email and compose replies <strong>if it were
                            running inside Gmail.</strong>
                        </div>
                        <button onClick={handleGenerateReply} disabled={isGenerating} className="btn-
                        primary flex items-center justify-center gap-2 px-4 py-2">
                           <SparklesIcon /> AI Reply
                        </button>
                    </div>
                </div>
            </div>
        </div>
    );
};


// ===== IamPolicyVisualizer.tsx =====

import React, { useState, useCallback, useMemo } from 'react';
import { testIamPermissions } from '../../services/gcpService.ts';
import { useGlobalState } from '../../contexts/GlobalStateContext.tsx';
import { GcpIcon, SparklesIcon, XMarkIcon } from '../icons.tsx';
import { LoadingSpinner } from '../shared/index.tsx';

type SimulationStatus = 'idle' | 'running' | 'completed' | 'error';
type NodeStatus = 'neutral' | 'pending' | 'success' | 'fail' | 'partial';

interface ResourceNode {
    id: string; // The full resource name
    name: string;
    type: 'project' | 'bucket' | 'instance' | 'unknown';
    status: NodeStatus;
    results?: { permission: string; granted: boolean }[];
}

const COMMON_ROLES = {
    'Viewer': ['resourcemanager.projects.get', 'storage.objects.list',
    'compute.instances.list'],
    'Editor': ['storage.objects.create', 'storage.objects.delete',
    'compute.instances.start', 'compute.instances.stop'],
    'Storage Object Admin': ['storage.objects.create', 'storage.objects.delete',
    'storage.objects.get', 'storage.objects.list', 'storage.objects.update'],
};

const getResourceType = (resourceId: string): ResourceNode['type'] => {
    if (resourceId.includes('/projects/')) return 'project';
    if (resourceId.includes('/b/')) return 'bucket';
    if (resourceId.includes('/instances/')) return 'instance';
    return 'unknown';
};

export const IamPolicyVisualizer: React.FC = () => {
    const { state } = useGlobalState();
    const [resources, setResources] = useState<ResourceNode[]>([]);
```

```
const [newResource, setNewResource] =
useState('//cloudresourcemanager.googleapis.com/projects/your-gcp-project-id');
const [permissions, setPermissions] =
useState('storage.objects.get\nstorage.objects.create');
const [simulationStatus, setSimulationStatus] =
useState<SimulationStatus>('idle');
const [error, setError] = useState('');
const [selectedNode, setSelectedNode] = useState<ResourceNode | null>(null);

const permissionList = useMemo(() => permissions.split('\n').map(p =>
p.trim()).filter(Boolean), [permissions]);

const handleAddResource = () => {
    if (newResource.trim() && !resources.find(r => r.id === newResource)) {
        setResources(prev => [...prev, {
            id: newResource,
            name: newResource.split('/').pop() || newResource,
            type: getResourceType(newResource),
            status: 'neutral',
        }]);
        setNewResource('');
    }
};

const handleRunSimulation = useCallback(async () => {
    if (!state.user) {
        setError('You must be signed in to run a simulation.');
        return;
    }
    if (resources.length === 0 || permissionList.length === 0) {
        setError('Please add at least one resource and one permission.');
        return;
    }

    setSimulationStatus('running');
    setError('');
    setSelectedNode(null);
    setResources(r => r.map(res => ({ ...res, status: 'pending', results: [] })));

    const promises = resources.map(resource =>
        testIamPermissions(resource.id, permissionList)
            .then(result => ({ id: resource.id, success: true, data: result }))
            .catch(err => ({ id: resource.id, success: false, error: err }))
    );

    const results = await Promise.allSettled(promises);

    setResources(prevResources => prevResources.map(resource => {
        const result: any = results.find((r: any) => r.value?.id === resource.id);
        if (!result || !result.value.success) {
            return { ...resource, status: 'fail' as NodeStatus };
        }

        const grantedPermissions = result.value.data.permissions || [];
        const permissionResults = permissionList.map(p => ({ permission: p, granted:
        grantedPermissions.includes(p) }));
        const allGranted = permissionResults.every(r => r.granted);
        const noneGranted = permissionResults.every(r => !r.granted);

        let status: NodeStatus = 'partial';
        if (allGranted) status = 'success';
        if (noneGranted) status = 'fail';
```

```
        return { ...resource, status, results: permissionResults };
    }));

    setSimulationStatus('completed');

}, [resources, permissionList, state.user]);

const nodeColorClass: Record<NodeStatus, string> = {
    neutral: 'border-slate-600',
    pending: 'border-yellow-500 animate-pulse',
    success: 'border-green-500',
    fail: 'border-red-500',
    partial: 'border-orange-500',
};

return (
    <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary bg-
    background">
        {selectedNode && (
            <div className="fixed inset-0 bg-black/60 z-50 flex items-center justify-center"
            onClick={() => setSelectedNode(null)}>
                <div className="bg-surface rounded-lg shadow-xl p-6 w-full max-w-lg animate-pop-
                in" onClick={e => e.stopPropagation()}>
                    <h3 className="text-lg font-bold truncate">{selectedNode.name}</h3>
                    <p className="text-xs text-text-secondary font-mono mb-4">{selectedNode.id}</p>
                    <ul className="space-y-2 max-h-96 overflow-y-auto">
                        {selectedNode.results?.map(res => (
                            <li key={res.permission} className={`flex items-center justify-between p-2
                            rounded text-sm ${res.granted ? 'bg-green-500/10' : 'bg-red-500/10'}`}>
                                <span className="font-mono">{res.permission}</span>
                                <span className={`font-bold ${res.granted ? 'text-green-500' : 'text-
                                red-500'}`}>{res.granted ? 'GRANTED' : 'DENIED'}</span>
                            </li>
                        ))}
                    </ul>
                </div>
            </div>
        )}
        <header className="mb-6"><h1 className="text-3xl font-bold flex items-
        center"><GcpIcon /><span className="ml-3">GCP IAM Policy
        Visualizer</span></h1><p className="text-text-secondary mt-1">Visually test and
        audit GCP IAM permissions in real-time across your resources.</p></header>
        <div className="flex-grow grid grid-cols-1 lg:grid-cols-3 gap-6 min-h-0">
            <aside className="lg:col-span-1 bg-surface p-4 rounded-lg border border-border
            flex flex-col gap-4">
                <h3 className="text-xl font-bold">Simulation Controls</h3>
                <div><label className="text-sm font-semibold">1. Add Resource</label><div
                className="flex gap-1 mt-1"><input value={newResource} onChange={e =>
                setNewResource(e.target.value)} placeholder="Full GCP resource name..."
                className="flex-grow p-2 bg-background border rounded text-xs" /><button
                onClick={handleAddResource} className="btn-primary px-3 text-
                sm">+</button></div></div>
                <div><label className="text-sm font-semibold">2. Define Permission
                Set</label><select onChange={e => setPermissions(COMMON_ROLES[e.target.value as
                keyof typeof COMMON_ROLES]?.join('\n') || '')} className="w-full mt-1 p-2 bg-
                background border rounded text-xs mb-1"><option>Load common
                role...</option>{Object.keys(COMMON_ROLES).map(r => <option
                key={r}>{r}</option>)}</select><textarea value={permissions} onChange={e =>
                setPermissions(e.target.value)} placeholder="One permission per line..."
                className="w-full h-32 p-2 bg-background border rounded text-xs font-
                mono"/></div>
                <button onClick={handleRunSimulation} disabled={simulationStatus === 'running'}
```

```
                    className="btn-primary py-3 flex items-center justify-center
                    gap-2"><SparklesIcon /> {simulationStatus === 'running' ? 'Simulating...' : 'Run
                    Simulation'}</button>
                    {error && <p className="text-red-500 text-xs text-center">{error}</p>}
                </aside>
                <main className="lg:col-span-2 bg-gray-50 dark:bg-slate-900/50 rounded-lg p-4
                border-2 border-dashed border-border overflow-auto relative">
                    <div className="grid grid-cols-2 md:grid-cols-3 lg:grid-cols-4 gap-4">
                        {resources.map(res => (
                            <div key={res.id} onClick={() => res.results && setSelectedNode(res)}
                            className={`p-4 bg-surface rounded-lg border-4 transition-colors duration-500
                            ${nodeColorClass[res.status]} ${res.results ? 'cursor-pointer hover:scale-105' :
                            ''}`}>
                                <h4 className="font-bold truncate">{res.name}</h4>
                                <p className="text-xs text-text-secondary capitalize">{res.type}</p>
                            </div>
                        ))}
                    </div>
                    {resources.length === 0 && <div className="absolute inset-0 flex items-center
                    justify-center text-text-secondary">Add resources to begin your
                    simulation.</div>}
                </main>
            </div>
        </div>
    );
};

// ===== ProjectExplorer_50.tsx =====

import React, { useState, useEffect, useCallback } from 'react';
import { useGlobalState } from '../../contexts/GlobalStateContext.tsx';
import { useNotification } from '../../contexts/NotificationContext.tsx';
import { initializeOctokit } from '../../services/authService.ts';
import { getDecryptedCredential } from '../../services/vaultService.ts';
import { getRepos, getRepoTree, getFileContent, commitFiles } from
'../../services/githubService.ts';
import { generateCommitMessageStream, answerProjectQuestion,
generateNewFilesForProject } from '../../services/index.ts';
import type { Repo, FileNode, GeneratedFile } from '../../types.ts';
import { FolderIcon, DocumentIcon, SparklesIcon, XMarkIcon } from
'../icons.tsx';
import { LoadingSpinner, MarkdownRenderer } from '../shared/index.tsx';
import * as Diff from 'diff';

const FileTree: React.FC<{ node: FileNode, onFileSelect: (path: string, name:
string) => void, activePath: string | null }> = ({ node, onFileSelect,
activePath }) => {
    const [isOpen, setIsOpen] = useState(true);

    if (node.type === 'file') {
        const isActive = activePath === node.path;
        return (
            <div
                className={`flex items-center space-x-2 pl-4 py-1 cursor-pointer rounded
                ${isActive ? 'bg-primary/10 text-primary' : 'hover:bg-gray-100 dark:hover:bg-
                slate-700'}`}
                onClick={() => onFileSelect(node.path, node.name)}
            >
                <DocumentIcon />
                <span>{node.name}</span>
            </div>
        );
```

```
    }

    return (
        <div>
            <div
                className="flex items-center space-x-2 py-1 cursor-pointer hover:bg-gray-100
                dark:hover:bg-slate-700 rounded"
                onClick={() => setIsOpen(!isOpen)}
            >
                <div className={`transform transition-transform ${isOpen ? 'rotate-90' :
                ''}`}>■</div>
                <FolderIcon />
                <span className="font-semibold">{node.name}</span>
            </div>
            {isOpen && node.children && (
                <div className="pl-4 border-l border-border ml-3">
                    {node.children.map(child => <FileTree key={child.path} node={child}
                    onFileSelect={onFileSelect} activePath={activePath} />)}
                </div>
            )}
        </div>
    );
};

const GeneratedFilesModal: React.FC<{
    files: GeneratedFile[];
    onClose: () => void;
    onCommit: (commitMessage: string) => void;
    isCommitting: boolean;
}> = ({ files, onClose, onCommit, isCommitting }) => {
    const [commitMessage, setCommitMessage] = useState('');
    const [activeFile, setActiveFile] = useState(files[0]);

    useEffect(() => {
        const generateMessage = async () => {
            const diffContext = files.map(f => `File:
            ${f.filePath}\n\n${f.content}`).join('\n---\n');
            const stream = generateCommitMessageStream(diffContext);
            let message = '';
            for await (const chunk of stream) {
                message += chunk;
                setCommitMessage(message);
            }
        };
        generateMessage();
    }, [files]);

    return (
         <div className="fixed inset-0 bg-black/60 z-50 flex items-center justify-center"
         onClick={onClose}>
            <div className="bg-surface rounded-lg shadow-xl w-full max-w-4xl h-[80vh] flex
            flex-col" onClick={e => e.stopPropagation()}>
                <header className="flex justify-between items-center p-4 border-b">
                    <h2 className="text-lg font-bold">Generated Files</h2>
                    <button onClick={onClose}><XMarkIcon/></button>
                </header>
                <div className="flex-grow flex min-h-0">
                    <aside className="w-1/3 border-r p-2 overflow-y-auto">
                        <ul>
                            {files.map(f => (
                                <li key={f.filePath} onClick={() => setActiveFile(f)} className={`p-2 rounded
                                cursor-pointer ${activeFile.filePath === f.filePath ? 'bg-primary/10' :
```

```
                                ''}`}>{f.filePath}</li>
                            ))}
                        </ul>
                    </aside>
                    <main className="w-2/3 overflow-y-auto">
                        <MarkdownRenderer content={'```\n' + activeFile.content + '\n```'} />
                    </main>
                </div>
                <footer className="p-4 border-t flex gap-4 items-center">
                    <input type="text" value={commitMessage} onChange={e =>
                    setCommitMessage(e.target.value)} placeholder="Commit message..."
                    className="flex-grow p-2 bg-background border rounded"/>
                    <button onClick={() => onCommit(commitMessage)} disabled={isCommitting}
                    className="btn-primary px-4 py-2 flex items-center justify-center
                    min-w-[120px]">{isCommitting ? <LoadingSpinner/> : 'Commit to Repo'}</button>
                </footer>
            </div>
        </div>
    )
}

const AiAssistantPanel: React.FC<{
    projectFiles: FileNode | null;
    onFilesGenerated: (files: GeneratedFile[]) => void;
}> = ({ projectFiles, onFilesGenerated }) => {
    const [isOpen, setIsOpen] = useState(false);
    const [tab, setTab] = useState<'ask' | 'generate'>('ask');
    const [prompt, setPrompt] = useState('');
    const [isLoading, setIsLoading] = useState(false);
    const [result, setResult] = useState('');

    const handleSubmit = async () => {
        if (!prompt.trim() || !projectFiles) return;
        setIsLoading(true);
        setResult('');

        try {
            if (tab === 'ask') {
                const stream = answerProjectQuestion(prompt, projectFiles);
                let fullResponse = '';
                for await (const chunk of stream) {
                    fullResponse += chunk;
                    setResult(fullResponse);
                }
            } else {
                const files = await generateNewFilesForProject(prompt, projectFiles);
                onFilesGenerated(files);
            }
        } catch (e) {
            setResult(`Error: ${e instanceof Error ? e.message : 'Unknown error'}`);
        } finally {
            setIsLoading(false);
            setPrompt('');
        }
    };

    return (
        <div className="flex-shrink-0 bg-surface border-t border-border">
            <button onClick={() => setIsOpen(!isOpen)} className="w-full p-2 text-left text-
            sm font-semibold flex items-center justify-between">
                <span><SparklesIcon/> AI Project Assistant</span>
                <span>{isOpen ? '▼' : '▲'}</span>
```

```
            </button>
            {isOpen && (
                <div className="p-4 border-t">
                    <div className="flex border-b mb-2">
                        <button onClick={() => setTab('ask')} className={`px-3 py-1 text-sm ${tab ===
                        'ask' ? 'border-b-2 border-primary' : ''}`}>Ask AI</button>
                        <button onClick={() => setTab('generate')} className={`px-3 py-1 text-sm ${tab
                        === 'generate' ? 'border-b-2 border-primary' : ''}`}>Generate Files</button>
                    </div>
                    {result && tab === 'ask' && <div className="p-2 bg-background rounded mb-2
                    max-h-48 overflow-y-auto"><MarkdownRenderer content={result} /></div>}
                    <div className="flex gap-2">
                        <input value={prompt} onChange={e => setPrompt(e.target.value)} onKeyDown={e =>
                        e.key === 'Enter' && handleSubmit()} placeholder={tab === 'ask' ? 'e.g., Where
                        is the auth logic?' : 'e.g., Create a new utility file with a date formatting
                        function.'} className="flex-grow p-2 text-sm bg-background border rounded"/>
                        <button onClick={handleSubmit} disabled={isLoading} className="btn-primary px-4
                        py-1 text-sm">{isLoading ? <LoadingSpinner/> : 'Send'}</button>
                    </div>
                </div>
            )}
        </div>
    )
};


export const ProjectExplorer: React.FC = () => {
    const { state, dispatch } = useGlobalState();
    const { user, githubUser, selectedRepo, projectFiles } = state;
    const { addNotification } = useNotification();
    const [repos, setRepos] = useState<Repo[]>([]);
    const [isLoading, setIsLoading] = useState<'repos' | 'tree' | 'file' | 'commit'
    | null>(null);
    const [error, setError] = useState('');
    const [activeFile, setActiveFile] = useState<{ path: string; name: string;
    originalContent: string; editedContent: string} | null>(null);
    const [generatedFiles, setGeneratedFiles] = useState<GeneratedFile[] |
    null>(null);

    const getApiClient = useCallback(async () => {
        if (!user) {
            throw new Error("You must be logged in to use the Project Explorer.");
        }
        const token = await getDecryptedCredential('github_pat');
        if (!token) {
            throw new Error("GitHub token not found. Please add it on the Connections
            page.");
        }
        return initializeOctokit(token);
    }, [user]);


    useEffect(() => {
        const loadRepos = async () => {
            if (user && githubUser) {
                setIsLoading('repos');
                setError('');
                try {
                    const octokit = await getApiClient();
                    const userRepos = await getRepos(octokit);
                    setRepos(userRepos);
                } catch (err) {
```

```
                setError(err instanceof Error ? err.message : 'Failed to load repositories');
            } finally {
                setIsLoading(null);
            }
        } else {
            setRepos([]);
        }
    };
    loadRepos();
}, [user, githubUser, getApiClient]);

useEffect(() => {
    const loadTree = async () => {
         if (selectedRepo && user && githubUser) {
            setIsLoading('tree');
            setError('');
            setActiveFile(null);
            try {
                const octokit = await getApiClient();
                const tree = await getRepoTree(octokit, selectedRepo.owner, selectedRepo.repo);
                dispatch({ type: 'LOAD_PROJECT_FILES', payload: tree });
            } catch (err) {
                 setError(err instanceof Error ? err.message : 'Failed to load repository tree');
            } finally {
                setIsLoading(null);
            }
        }
    };
    loadTree();
}, [selectedRepo, user, githubUser, dispatch, getApiClient]);

const handleFileSelect = async (path: string, name: string) => {
    if (!selectedRepo) return;
    setIsLoading('file');
    try {
        const octokit = await getApiClient();
        const content = await getFileContent(octokit, selectedRepo.owner,
        selectedRepo.repo, path);
        setActiveFile({ path, name, originalContent: content, editedContent: content });
    } catch (err) {
        setError((err as Error).message);
    } finally {
        setIsLoading(null);
    }
};

const handleCommit = async () => {
    if (!activeFile || !selectedRepo || activeFile.originalContent ===
    activeFile.editedContent) return;

    setIsLoading('commit');
    setError('');
    try {
        const diff = Diff.createPatch(activeFile.path, activeFile.originalContent,
        activeFile.editedContent);

        const stream = generateCommitMessageStream(diff);
        let commitMessage = '';
        for await (const chunk of stream) { commitMessage += chunk; }

        const finalMessage = window.prompt("Confirm or edit commit message:",
        commitMessage);
```

```
        if (!finalMessage) {
            setIsLoading(null);
            return;
        }

        const octokit = await getApiClient();
        await commitFiles(
            octokit,
            selectedRepo.owner,
            selectedRepo.repo,
            [{ path: activeFile.path, content: activeFile.editedContent }],
            finalMessage
        );

        addNotification(`Successfully committed to ${selectedRepo.repo}`, 'success');
        setActiveFile(prev => prev ? { ...prev, originalContent: prev.editedContent } :
        null);

    } catch (err) {
        const message = err instanceof Error ? err.message : 'Failed to commit changes';
        setError(message);
        addNotification(message, 'error');
    } finally {
        setIsLoading(null);
    }
};

 const handleCommitGeneratedFiles = async (commitMessage: string) => {
    if (!generatedFiles || !selectedRepo) return;
    setIsLoading('commit');
    try {
         const octokit = await getApiClient();
         await commitFiles(
            octokit,
            selectedRepo.owner,
            selectedRepo.repo,
            generatedFiles.map(f => ({ path: f.filePath, content: f.content })),
            commitMessage
         );
         addNotification(`Successfully committed ${generatedFiles.length} new files!`,
         'success');
         setGeneratedFiles(null);
         // Reload tree
         const tree = await getRepoTree(octokit, selectedRepo.owner, selectedRepo.repo);
         dispatch({ type: 'LOAD_PROJECT_FILES', payload: tree });
    } catch (err) {
        addNotification(err instanceof Error ? err.message : 'Failed to commit',
        'error');
    } finally {
        setIsLoading(null);
    }
};

if (!user) {
    return (
        <div className="h-full flex flex-col items-center justify-center text-center
        text-text-secondary p-4">
            <FolderIcon />
            <h2 className="text-lg font-semibold mt-2">Please Sign In</h2>
            <p>Sign in via the "Connections" tab to explore your repositories.</p>
        </div>
    );
```

```
}

if (!githubUser) {
     return (
        <div className="h-full flex flex-col items-center justify-center text-center
        text-text-secondary p-4">
            <FolderIcon />
            <h2 className="text-lg font-semibold mt-2">Connect to GitHub</h2>
            <p>Please go to the "Connections" tab and provide a Personal Access Token to
            explore your repositories.</p>
        </div>
    );
}

const hasChanges = activeFile ? activeFile.originalContent !==
activeFile.editedContent : false;

return (
    <div className="h-full flex flex-col text-text-primary">
        {generatedFiles && <GeneratedFilesModal files={generatedFiles} onClose={() =>
        setGeneratedFiles(null)} onCommit={handleCommitGeneratedFiles}
        isCommitting={isLoading === 'commit'}/>}
        <header className="p-4 border-b border-border flex-shrink-0">
            <h1 className="text-xl font-bold flex items-center"><FolderIcon /><span
            className="ml-3">Project Explorer</span></h1>
            <div className="mt-2">
                <select
                    value={selectedRepo ? `${selectedRepo.owner}/${selectedRepo.repo}` : ''}
                    onChange={e => {
                        const [owner, repo] = e.target.value.split('/');
                        dispatch({ type: 'SET_SELECTED_REPO', payload: { owner, repo } });
                    }}
                    className="w-full p-2 bg-surface border border-border rounded-md text-sm"
                >
                    <option value="" disabled>{isLoading === 'repos' ? 'Loading...' : 'Select a
                    repository'}</option>
                    {repos.map(r => <option key={r.id} value={r.full_name}>{r.full_name}</option>)}
                </select>
            </div>
            {error && <p className="text-red-500 text-xs mt-2">{error}</p>}
        </header>
        <div className="flex-grow flex min-h-0">
            <aside className="w-1/3 bg-background border-r border-border p-4 overflow-y-
            auto">
                {isLoading === 'tree' && <div className="flex justify-center"><LoadingSpinner
                /></div>}
                {projectFiles && <FileTree node={projectFiles} onFileSelect={handleFileSelect}
                activePath={activeFile?.path ?? null} />}
            </aside>
            <main className="flex-1 bg-surface flex flex-col">
                 <div className="flex justify-between items-center p-2 border-b border-border bg-
                 gray-50 dark:bg-slate-800">
                    <span className="text-sm font-semibold">{activeFile?.name || 'No file
                    selected'}</span>
                    <button onClick={handleCommit} disabled={!hasChanges || isLoading === 'commit'}
                    className="btn-primary px-4 py-1 text-sm flex items-center justify-center
                    min-w-[100px]">
                       {isLoading === 'commit' ? <LoadingSpinner/> : 'Commit'}
                    </button>
                 </div>
                 {isLoading === 'file' ? <div className="flex items-center justify-center
                 h-full"><LoadingSpinner /></div> :
```

```
                        <textarea
                            value={activeFile?.editedContent ?? 'Select a file to view its content.'}
                            onChange={e => setActiveFile(prev => prev ? { ...prev, editedContent:
                            e.target.value } : null)}
                            disabled={!activeFile}
                            className="w-full h-full p-4 text-sm font-mono bg-transparent resize-none
                            focus:outline-none"
                        />
                    }
                </main>
            </div>
            <AiAssistantPanel projectFiles={projectFiles}
            onFilesGenerated={setGeneratedFiles} />
        </div>
    );
};

// ===== SchemaDesigner_52.tsx =====

import React, { useState, useRef, useMemo, useCallback } from 'react';
import { MapIcon, ArrowDownTrayIcon, PlusIcon, TrashIcon, PencilIcon, LinkIcon }
from '../icons.tsx';
import { downloadFile } from '../../services/fileUtils.ts';

// --- TYPES ---
type Constraint = 'PK' | 'NN' | 'UQ' | string; // string for FK like
'FK_table_column'
type SQLDataType = 'INTEGER' | 'VARCHAR(255)' | 'TEXT' | 'DATE' | 'TIMESTAMP' |
'DECIMAL(18, 2)' | 'BOOLEAN';

interface Column {
    id: number;
    name: string;
    type: SQLDataType;
    constraints: Constraint[];
}
interface Table {
    id: number;
    name: string;
    columns: Column[];
    x: number;
    y: number;
}
interface Relationship {
    fromTable: number;
    fromColumn: number;
    toTable: number;
    toColumn: number;
}

// --- TEMPLATES ---
const bankingTemplates: Record<string, { name: string; tables: Table[] }> = {
    'core-banking': {
        name: 'Core Banking',
        tables: [
            { id: 1, name: 'Customers', x: 50, y: 50, columns: [
                { id: 1, name: 'customer_id', type: 'INTEGER', constraints: ['PK', 'NN'] },
                { id: 2, name: 'first_name', type: 'VARCHAR(255)', constraints: ['NN'] },
                { id: 3, name: 'last_name', type: 'VARCHAR(255)', constraints: ['NN'] },
                { id: 4, name: 'address', type: 'TEXT', constraints: [] },
                { id: 5, name: 'email', type: 'VARCHAR(255)', constraints: ['UQ', 'NN'] },
                { id: 6, name: 'phone', type: 'VARCHAR(255)', constraints: [] },
```

```
            { id: 7, name: 'created_at', type: 'TIMESTAMP', constraints: ['NN'] }
        ]},
        { id: 2, name: 'Accounts', x: 400, y: 50, columns: [
            { id: 1, name: 'account_id', type: 'INTEGER', constraints: ['PK', 'NN'] },
            { id: 2, name: 'customer_id', type: 'INTEGER', constraints:
            ['FK_Customers_customer_id'] },
            { id: 3, name: 'account_type', type: 'VARCHAR(255)', constraints: ['NN'] },
            { id: 4, name: 'balance', type: 'DECIMAL(18, 2)', constraints: ['NN'] },
            { id: 5, name: 'currency', type: 'VARCHAR(255)', constraints: ['NN'] },
            { id: 6, name: 'opened_at', type: 'TIMESTAMP', constraints: ['NN'] },
            { id: 7, name: 'status', type: 'VARCHAR(255)', constraints: ['NN'] }
        ]},
        { id: 3, name: 'Transactions', x: 400, y: 350, columns: [
            { id: 1, name: 'transaction_id', type: 'INTEGER', constraints: ['PK', 'NN'] },
            { id: 2, name: 'account_id', type: 'INTEGER', constraints:
            ['FK_Accounts_account_id'] },
            { id: 3, name: 'transaction_type', type: 'VARCHAR(255)', constraints: ['NN'] },
            { id: 4, name: 'amount', type: 'DECIMAL(18, 2)', constraints: ['NN'] },
            { id: 5, name: 'currency', type: 'VARCHAR(255)', constraints: ['NN'] },
            { id: 6, name: 'transaction_date', type: 'TIMESTAMP', constraints: ['NN'] },
            { id: 7, name: 'description', type: 'TEXT', constraints: [] }
        ]}
    ]
},
'wealth-management': {
    name: 'Wealth Management',
    tables: [
        { id: 1, name: 'Clients', x: 50, y: 150, columns: [ { id: 1, name: 'client_id',
        type: 'INTEGER', constraints: ['PK'] }, { id: 2, name: 'client_name', type:
        'VARCHAR(255)', constraints: ['NN'] } ]},
        { id: 2, name: 'Portfolios', x: 350, y: 50, columns: [ { id: 1, name:
        'portfolio_id', type: 'INTEGER', constraints: ['PK'] }, { id: 2, name:
        'client_id', type: 'INTEGER', constraints: ['FK_Clients_client_id'] }, { id: 3,
        name: 'name', type: 'VARCHAR(255)', constraints: ['NN'] } ]},
        { id: 3, name: 'Assets', x: 350, y: 250, columns: [ { id: 1, name: 'asset_id',
        type: 'INTEGER', constraints: ['PK'] }, { id: 2, name: 'ticker', type:
        'VARCHAR(255)', constraints: ['NN'] } ]},
        { id: 4, name: 'Holdings', x: 650, y: 150, columns: [ { id: 1, name:
        'holding_id', type: 'INTEGER', constraints: ['PK'] }, { id: 2, name:
        'portfolio_id', type: 'INTEGER', constraints: ['FK_Portfolios_portfolio_id'] },
        { id: 3, name: 'asset_id', type: 'INTEGER', constraints: ['FK_Assets_asset_id']
        }, { id: 4, name: 'quantity', type: 'DECIMAL(18, 2)', constraints: ['NN'] } ]}
    ]
},
'lending': {
    name: 'Lending',
    tables: [
        { id: 1, name: 'Borrowers', x: 50, y: 200, columns: [ { id: 1, name:
        'borrower_id', type: 'INTEGER', constraints: ['PK'] }, { id: 2, name: 'name',
        type: 'VARCHAR(255)', constraints: ['NN'] } ]},
        { id: 2, name: 'Loans', x: 350, y: 50, columns: [ { id: 1, name: 'loan_id',
        type: 'INTEGER', constraints: ['PK'] }, { id: 2, name: 'borrower_id', type:
        'INTEGER', constraints: ['FK_Borrowers_borrower_id'] }, { id: 3, name: 'amount',
        type: 'DECIMAL(18, 2)', constraints: ['NN'] } ]},
        { id: 3, name: 'Collateral', x: 350, y: 250, columns: [ { id: 1, name:
        'collateral_id', type: 'INTEGER', constraints: ['PK'] }, { id: 2, name:
        'loan_id', type: 'INTEGER', constraints: ['FK_Loans_loan_id'] }, { id: 3, name:
        'type', type: 'VARCHAR(255)', constraints: ['NN'] } ]},
        { id: 4, name: 'Repayments', x: 650, y: 150, columns: [ { id: 1, name:
        'repayment_id', type: 'INTEGER', constraints: ['PK'] }, { id: 2, name:
        'loan_id', type: 'INTEGER', constraints: ['FK_Loans_loan_id'] }, { id: 3, name:
        'amount', type: 'DECIMAL(18, 2)', constraints: ['NN'] } ]}
```

```
        ]
    }
};

// --- HELPERS ---
const exportToSQL = (tables: Table[]) => {
    let sql = '';
    const foreignKeys: string[] = [];

    tables.forEach(table => {
        const columnsSQL = table.columns.map(col => {
            let line = `  "${col.name}" ${col.type}`;
            if (col.constraints.includes('PK')) line += ' PRIMARY KEY';
            if (col.constraints.includes('NN')) line += ' NOT NULL';
            if (col.constraints.includes('UQ')) line += ' UNIQUE';
            return line;
        }).join(',\n');

        const tableFks = table.columns.filter(c => c.constraints.some(cons =>
        cons.startsWith('FK_')));
        tableFks.forEach(col => {
            const fkConstraint = col.constraints.find(c => c.startsWith('FK_'))!;
            const [, targetTable, targetCol] = fkConstraint.split('_');
            foreignKeys.push(`ALTER TABLE "${table.name}" ADD FOREIGN KEY ("${col.name}")
            REFERENCES "${targetTable}" ("${targetCol}");`);
        });

        sql += `CREATE TABLE "${table.name}" (\n${columnsSQL}\n);\n\n`;
    });

    sql += '-- Foreign Keys\n';
    sql += foreignKeys.join('\n');

    return sql;
};

// --- COMPONENTS ---
const TableComponent: React.FC<{ table: Table; onMouseDown: (e:
React.MouseEvent, id: number) => void; isSelected: boolean; }> = ({ table,
onMouseDown, isSelected }) => (
    <div
        className={`absolute w-64 bg-surface rounded-lg shadow-xl border-2 cursor-grab
        active:cursor-grabbing ${isSelected ? 'border-primary shadow-primary/20' :
        'border-border'}`}
        style={{ top: table.y, left: table.x, transform: 'translateZ(0)' }}
        onMouseDown={e => onMouseDown(e, table.id)}
    >
        <h3 className="font-bold text-lg p-2 bg-gray-50 dark:bg-slate-700/50 rounded-t-
        lg border-b border-border text-center text-text-primary">{table.name}</h3>
        <div className="p-2 space-y-1 font-mono text-xs">
            {table.columns.map(col => (
                <div key={col.id} className="flex justify-between items-center p-1 rounded
                hover:bg-gray-100 dark:hover:bg-slate-700">
                    <div>
                        <span className="text-text-primary font-semibold">{col.name}</span>
                        {col.constraints.includes('PK') && <span className="text-yellow-600 ml-1"
                        title="Primary Key">PK</span>}
                        {col.constraints.some(c => c.startsWith('FK_')) && <span className="text-
                        blue-500 ml-1" title="Foreign Key">FK</span>}
                    </div>
                    <span className="text-text-secondary">{col.type}</span>
                </div>
```

```
            ))}
        </div>
    </div>
);

export const SchemaDesigner: React.FC = () => {
    const [tables, setTables] = useState<Table[]>(bankingTemplates['core-
    banking'].tables);
    const [selectedTableId, setSelectedTableId] = useState<number | null>(null);
    const [dragging, setDragging] = useState<{ id: number; offsetX: number; offsetY:
    number } | null>(null);
    const canvasRef = useRef<HTMLDivElement>(null);

    const relationships = useMemo(() => {
        const rels: Relationship[] = [];
        const tableMap = new Map(tables.map(t => [t.name, t]));
        tables.forEach(table => {
            table.columns.forEach(col => {
                const fk = col.constraints.find(c => c.startsWith('FK_'));
                if (fk) {
                    const [, targetTableName, targetColName] = fk.split('_');
                    const targetTable = tableMap.get(targetTableName);
                    if (targetTable) {
                        const targetCol = targetTable.columns.find(c => c.name === targetColName);
                        if (targetCol) {
                            rels.push({ fromTable: table.id, fromColumn: col.id, toTable: targetTable.id,
                            toColumn: targetCol.id });
                        }
                    }
                }
            });
        });
        return rels;
    }, [tables]);

    const onMouseDown = (e: React.MouseEvent, id: number) => {
        e.stopPropagation();
        setSelectedTableId(id);
        const tableElement = e.currentTarget as HTMLDivElement;
        const rect = tableElement.getBoundingClientRect();
        const canvasRect = canvasRef.current!.getBoundingClientRect();
        setDragging({ id, offsetX: e.clientX - rect.left, offsetY: e.clientY - rect.top
        });
    };

    const onMouseMove = (e: React.MouseEvent<HTMLDivElement>) => {
        if (!dragging || !canvasRef.current) return;
        const canvasRect = canvasRef.current.getBoundingClientRect();
        setTables(tables.map(t => t.id === dragging.id ? { ...t, x: e.clientX -
        dragging.offsetX - canvasRect.left, y: e.clientY - dragging.offsetY -
        canvasRect.top } : t));
    };

    const onMouseUp = () => setDragging(null);

    const selectedTable = useMemo(() => tables.find(t => t.id === selectedTableId),
    [tables, selectedTableId]);

    const handleLoadTemplate = (key: string) => {
        setTables(bankingTemplates[key].tables);
        setSelectedTableId(null);
    };
```

```
return (
    <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
        <header className="mb-6"><h1 className="text-3xl font-bold flex items-
        center"><MapIcon /><span className="ml-3">Interactive Schema
        Designer</span></h1><p className="text-text-secondary mt-1">Visually design your
        database schema, load banking templates, and export to SQL.</p></header>
        <div className="flex-grow flex gap-6 min-h-0">
            <main ref={canvasRef} className="flex-grow relative bg-background rounded-lg
            border-2 border-dashed border-border overflow-auto" onMouseMove={onMouseMove}
            onMouseUp={onMouseUp} onMouseLeave={onMouseUp} onClick={() =>
            setSelectedTableId(null)}>
                 <svg className="absolute inset-0 w-full h-full pointer-events-none" width="100%"
                 height="100%">
                    {relationships.map((rel, i) => {
                        const fromTable = tables.find(t => t.id === rel.fromTable);
                        const toTable = tables.find(t => t.id === rel.toTable);
                        if (!fromTable || !toTable) return null;
                        return <line key={i} x1={fromTable.x + 128} y1={fromTable.y + 20} x2={toTable.x
                        + 128} y2={toTable.y + 20} stroke="var(--color-primary)" strokeWidth="2"
                        strokeDasharray="5,5" />;
                    })}
                </svg>
                {tables.map(table => (
                    <div key={table.id} onClick={e => {e.stopPropagation();
                    setSelectedTableId(table.id)}}>
                        <TableComponent table={table} onMouseDown={onMouseDown}
                        isSelected={selectedTableId === table.id} />
                    </div>
                ))}
            </main>
            <aside className="w-96 flex-shrink-0 flex flex-col gap-4">
                 <div className="bg-surface border border-border p-4 rounded-lg">
                    <label className="font-bold mb-2 block">Load Template</label>
                    <select onChange={(e) => handleLoadTemplate(e.target.value)} className="w-full
                    p-2 bg-background border border-border rounded">
                        {Object.entries(bankingTemplates).map(([key, {name}]) => <option key={key}
                        value={key}>{name}</option>)}
                    </select>
                </div>
                <div className="flex-grow bg-surface border border-border p-4 rounded-lg
                overflow-y-auto">
                    <h3 className="font-bold mb-2 text-lg">Editor</h3>
                    {selectedTable ? (
                        <div className="text-sm">Editing <span className="font-mono text-
                        primary">{selectedTable.name}</span> coming soon...</div>
                    ) : (
                        <p className="text-xs text-text-secondary">Select a table to see details or add
                        a new one.</p>
                    )}
                </div>
                <div className="flex flex-col gap-2">
                     <button onClick={() => downloadFile(JSON.stringify(tables, null, 2),
                     'schema.json', 'application/json')} className="flex-1 text-sm py-2 bg-gray-100
                     dark:bg-slate-700 border border-border rounded-md flex items-center justify-
                     center gap-2 hover:bg-gray-200 dark:hover:bg-slate-600">
                        <ArrowDownTrayIcon className="w-4 h-4"/> Download JSON
                    </button>
                     <button onClick={() => downloadFile(exportToSQL(tables), 'schema.sql',
                     'application/sql')} className="btn-primary flex-1 text-sm py-2 flex items-center
                     justify-center gap-2">
                        <ArrowDownTrayIcon className="w-4 h-4"/> Export to SQL
                     </button>
```

```
                    </div>
                </aside>
            </div>
        </div>
    );
};


// ===== ApiClientGenerator.tsx =====

import React from 'react';
import { CodeBracketSquareIcon } from '../icons.tsx';

export const ApiClientGenerator: React.FC = () => {
    return (
        <div className="h-full flex flex-col items-center justify-center p-4 text-
        center">
            <div className="text-4xl text-primary mb-4"><CodeBracketSquareIcon /></div>
            <h1 className="text-2xl font-bold">API Client Generator</h1>
            <p className="text-text-secondary mt-2">This feature is under construction.</p>
        </div>
    );
};


// ===== SqlToApiGenerator.tsx =====

import React from 'react';
import { ServerStackIcon } from '../icons.tsx';

export const SqlToApiGenerator: React.FC = () => {
    return (
        <div className="h-full flex flex-col items-center justify-center p-4 text-
        center">
            <div className="text-4xl text-primary mb-4"><ServerStackIcon /></div>
            <h1 className="text-2xl font-bold">SQL to API Generator</h1>
            <p className="text-text-secondary mt-2">This feature is under construction.</p>
        </div>
    );
};


// ===== BlamelessPostmortemGenerator.tsx =====

import React from 'react';
import { DocumentTextIcon } from '../icons.tsx';

export const BlamelessPostmortemGenerator: React.FC = () => {
    return (
        <div className="h-full flex flex-col items-center justify-center p-4 text-
        center">
            <div className="text-4xl text-primary mb-4"><DocumentTextIcon /></div>
            <h1 className="text-2xl font-bold">Blameless Post-mortem Generator</h1>
            <p className="text-text-secondary mt-2">This feature is under construction.</p>
        </div>
    );
};


// ===== DataAnonymizer.tsx =====

import React from 'react';
```

```
import { ShieldCheckIcon } from '../icons.tsx';

export const DataAnonymizer: React.FC = () => {
    return (
        <div className="h-full flex flex-col items-center justify-center p-4 text-
        center">
            <div className="text-4xl text-primary mb-4"><ShieldCheckIcon /></div>
            <h1 className="text-2xl font-bold">Data Anonymizer</h1>
            <p className="text-text-secondary mt-2">This feature is under construction.</p>
        </div>
    );
};


// ===== ABTestAssistant.tsx =====

import React from 'react';
import { BeakerIcon } from '../icons.tsx';

export const ABTestAssistant: React.FC = () => {
    return (
        <div className="h-full flex flex-col items-center justify-center p-4 text-
        center">
            <div className="text-4xl text-primary mb-4"><BeakerIcon /></div>
            <h1 className="text-2xl font-bold">A/B Test Assistant</h1>
            <p className="text-text-secondary mt-2">This feature is under construction.</p>
        </div>
    );
};


// ===== I18nHelper.tsx =====

import React from 'react';
import { ProjectExplorerIcon } from '../icons.tsx';

export const I18nHelper: React.FC = () => {
    return (
        <div className="h-full flex flex-col items-center justify-center p-4 text-
        center">
            <div className="text-4xl text-primary mb-4"><ProjectExplorerIcon /></div>
            <h1 className="text-2xl font-bold">i18n Helper</h1>
            <p className="text-text-secondary mt-2">This feature is under construction.</p>
        </div>
    );
};


// ===== TokenUsageEstimator.tsx =====

import React from 'react';
import { CpuChipIcon } from '../icons.tsx';

export const TokenUsageEstimator: React.FC = () => {
    return (
        <div className="h-full flex flex-col items-center justify-center p-4 text-
        center">
            <div className="text-4xl text-primary mb-4"><CpuChipIcon /></div>
            <h1 className="text-2xl font-bold">Token Usage Estimator</h1>
            <p className="text-text-secondary mt-2">This feature is under construction.</p>
        </div>
    );
```

```
};


// ===== FinancialChartGenerator.tsx =====

import React from 'react';
import { ChartBarIcon } from '../icons.tsx';

export const FinancialChartGenerator: React.FC = () => {
    return (
        <div className="h-full flex flex-col items-center justify-center p-4 text-
        center">
            <div className="text-4xl text-primary mb-4"><ChartBarIcon /></div>
            <h1 className="text-2xl font-bold">Financial Chart Generator</h1>
            <p className="text-text-secondary mt-2">This feature is under construction.</p>
        </div>
    );
};


// ===== ComplianceReportHelper.tsx =====

import React from 'react';
import { ShieldCheckIcon } from '../icons.tsx';

export const ComplianceReportHelper: React.FC = () => {
    return (
        <div className="h-full flex flex-col items-center justify-center p-4 text-
        center">
            <div className="text-4xl text-primary mb-4"><ShieldCheckIcon /></div>
            <h1 className="text-2xl font-bold">Compliance Report Helper</h1>
            <p className="text-text-secondary mt-2">This feature is under construction.</p>
        </div>
    );
};


// ===== EcommerceComponentGenerator.tsx =====

import React from 'react';
import { ArchiveBoxIcon } from '../icons.tsx';

export const EcommerceComponentGenerator: React.FC = () => {
    return (
        <div className="h-full flex flex-col items-center justify-center p-4 text-
        center">
            <div className="text-4xl text-primary mb-4"><ArchiveBoxIcon /></div>
            <h1 className="text-2xl font-bold">E-commerce Component Generator</h1>
            <p className="text-text-secondary mt-2">This feature is under construction.</p>
        </div>
    );
};


// ===== ApiEndpointTester.tsx =====

import React from 'react';
import { PaperAirplaneIcon } from '../icons.tsx';

export const ApiEndpointTester: React.FC = () => {
    return (
        <div className="h-full flex flex-col items-center justify-center p-4 text-
```

```
        center">
            <div className="text-4xl text-primary mb-4"><PaperAirplaneIcon /></div>
            <h1 className="text-2xl font-bold">API Endpoint Tester</h1>
            <p className="text-text-secondary mt-2">This feature is under construction.</p>
        </div>
    );
};


// ===== StoryboardGenerator.tsx =====

import React from 'react';
import { PhotoIcon } from '../icons.tsx';

export const StoryboardGenerator: React.FC = () => {
    return (
        <div className="h-full flex flex-col items-center justify-center p-4 text-
        center">
            <div className="text-4xl text-primary mb-4"><PhotoIcon /></div>
            <h1 className="text-2xl font-bold">Storyboard Generator</h1>
            <p className="text-text-secondary mt-2">This feature is under construction.</p>
        </div>
    );
};


// ===== UserPersonaGenerator.tsx =====

import React from 'react';
import { DocumentIcon } from '../icons.tsx';

export const UserPersonaGenerator: React.FC = () => {
    return (
        <div className="h-full flex flex-col items-center justify-center p-4 text-
        center">
            <div className="text-4xl text-primary mb-4"><DocumentIcon /></div>
            <h1 className="text-2xl font-bold">User Persona Generator</h1>
            <p className="text-text-secondary mt-2">This feature is under construction.</p>
        </div>
    );
};


// ===== CompetitiveAnalysisBot.tsx =====

import React from 'react';
import { MagnifyingGlassIcon } from '../icons.tsx';

export const CompetitiveAnalysisBot: React.FC = () => {
    return (
        <div className="h-full flex flex-col items-center justify-center p-4 text-
        center">
            <div className="text-4xl text-primary mb-4"><MagnifyingGlassIcon /></div>
            <h1 className="text-2xl font-bold">Competitive Analysis Bot</h1>
            <p className="text-text-secondary mt-2">This feature is under construction.</p>
        </div>
    );
};


// ===== CodeDocumentationWriter.tsx =====
import React from 'react';
```

```
import { DocumentTextIcon } from '../icons.tsx';

export const CodeDocumentationWriter: React.FC = () => {
    return (
        <div className="h-full flex flex-col items-center justify-center p-4 text-
        center">
            <div className="text-4xl text-primary mb-4"><DocumentTextIcon /></div>
            <h1 className="text-2xl font-bold">Code Documentation Writer</h1>
            <p className="text-text-secondary mt-2">This feature is under construction.</p>
        </div>
    );
};


// ===== DependencyUpdateExplainer.tsx =====

import React from 'react';
import { GitBranchIcon } from '../icons.tsx';

export const DependencyUpdateExplainer: React.FC = () => {
    return (
        <div className="h-full flex flex-col items-center justify-center p-4 text-
        center">
            <div className="text-4xl text-primary mb-4"><GitBranchIcon /></div>
            <h1 className="text-2xl font-bold">Dependency Update Explainer</h1>
            <p className="text-text-secondary mt-2">This feature is under construction.</p>
        </div>
    );
};


// ===== AiVideoGenerator.tsx =====

import React from 'react';
import { VideoCameraIcon } from '../icons.tsx';

export const AiVideoGenerator: React.FC = () => {
    return (
        <div className="h-full flex flex-col items-center justify-center p-4 text-
        center">
            <div className="text-4xl text-primary mb-4"><VideoCameraIcon /></div>
            <h1 className="text-2xl font-bold">AI Video Generator</h1>
            <p className="text-text-secondary mt-2">This feature is under construction.</p>
        </div>
    );
};


// ===== CloudCostEstimator.tsx =====

import React from 'react';
import { GcpIcon } from '../icons.tsx';

export const CloudCostEstimator: React.FC = () => {
    return (
        <div className="h-full flex flex-col items-center justify-center p-4 text-
        center">
            <div className="text-4xl text-primary mb-4"><GcpIcon /></div>
            <h1 className="text-2xl font-bold">Cloud Cost Estimator</h1>
            <p className="text-text-secondary mt-2">This feature is under construction.</p>
        </div>
    );
```

```
};


// ===== SmartLogger.tsx =====

import React from 'react';
import { TerminalIcon } from '../icons.tsx';

export const SmartLogger: React.FC = () => {
    return (
        <div className="h-full flex flex-col items-center justify-center p-4 text-
        center">
            <div className="text-4xl text-primary mb-4"><TerminalIcon /></div>
            <h1 className="text-2xl font-bold">Smart Logger</h1>
            <p className="text-text-secondary mt-2">This feature is under construction.</p>
        </div>
    );
};


// ===== AccessibilityAnnotation.tsx =====

import React from 'react';
import { EyeIcon } from '../icons.tsx';

export const AccessibilityAnnotation: React.FC = () => {
    return (
        <div className="h-full flex flex-col items-center justify-center p-4 text-
        center">
            <div className="text-4xl text-primary mb-4"><EyeIcon /></div>
            <h1 className="text-2xl font-bold">Accessibility Annotation</h1>
            <p className="text-text-secondary mt-2">This feature is under construction.</p>
        </div>
    );
};


// ===== AiCommandCenter_52.tsx =====

import React, { useState, useCallback } from 'react';
import { Type, FunctionDeclaration } from "@google/genai";
import { logError, getInferenceFunction, CommandResponse, FEATURE_TAXONOMY,
executeWorkspaceAction, ACTION_REGISTRY } from '../../services/index.ts';
import { useGlobalState } from '../../contexts/GlobalStateContext.tsx';
import { CommandLineIcon } from '../icons.tsx';
import { LoadingSpinner } from '../shared/index.tsx';
import { ALL_FEATURE_IDS } from '../../constants.tsx';

const baseFunctionDeclarations: FunctionDeclaration[] = [
    {
        name: 'navigateTo',
        description: 'Navigates to a specific feature page.',
        parameters: {
            type: Type.OBJECT,
            properties: {
                featureId: {
                    type: Type.STRING,
                    description: 'The ID of the feature to navigate to.',
                    enum: ALL_FEATURE_IDS
                },
            },
            required: ['featureId'],
```

```
        },
    },
    {
        name: 'runFeatureWithInput',
        description: 'Navigates to a feature and passes initial data to it.',
        parameters: {
            type: Type.OBJECT,
            properties: {
                 featureId: {
                    type: Type.STRING,
                    description: 'The ID of the feature to run.',
                    enum: ALL_FEATURE_IDS
                },
                props: {
                    type: Type.OBJECT,
                    description: 'An object containing the initial properties for the feature, based
                    on its required inputs.',
                    properties: {
                        initialCode: { type: Type.STRING },
                        initialPrompt: { type: Type.STRING },
                        beforeCode: { type: Type.STRING },
                        afterCode: { type: Type.STRING },
                        logInput: { type: Type.STRING },
                        diff: { type: Type.STRING },
                        codeInput: { type: Type.STRING },
                        jsonInput: { type: Type.STRING },
                    }
                }
            },
            required: ['featureId', 'props']
        }
    }
];

// Dynamically add the workspace action
const functionDeclarations: FunctionDeclaration[] = [
    ...baseFunctionDeclarations,
    {
        name: 'runWorkspaceAction',
        description: 'Executes a defined action on a connected workspace service like
        Jira, Slack, or GitHub.',
        parameters: {
            type: Type.OBJECT,
            properties: {
                 actionId: {
                    type: Type.STRING,
                    description: 'The unique identifier for the action to execute.',
                    enum: [ ...ACTION_REGISTRY.keys() ]
                },
                params: {
                    type: Type.OBJECT,
                    description: 'An object containing the parameters for the action, matching its
                    required inputs.'
                }
            },
            required: ['actionId', 'params']
        }
    }
]

const knowledgeBase = FEATURE_TAXONOMY.map(f => `- ${f.name} (${f.id}):
${f.description} Inputs: ${f.inputs}`).join('\n');
```

```
const ExamplePromptButton: React.FC<{ text: string, onClick: (text: string) =>
void }> = ({ text, onClick }) => (
    <button
        onClick={() => onClick(text)}
        className="px-3 py-1.5 bg-surface border border-border rounded-full text-xs
        hover:bg-gray-100 dark:hover:bg-slate-700 transition-colors"
    >
        {text}
    </button>
)

export const AiCommandCenter: React.FC = () => {
    const { dispatch } = useGlobalState();
    const [prompt, setPrompt] = useState('');
    const [isLoading, setIsLoading] = useState(false);
    const [lastResponse, setLastResponse] = useState('');

    const handleCommand = useCallback(async () => {
        if (!prompt.trim()) return;

        setIsLoading(true);
        setLastResponse('');

        try {
            const response: CommandResponse = await getInferenceFunction(prompt,
            functionDeclarations, knowledgeBase);

            if (response.functionCalls && response.functionCalls.length > 0) {
                const call = response.functionCalls[0];
                const { name, args } = call;

                setLastResponse(`Understood! Executing command: ${name}`);

                switch (name) {
                    case 'navigateTo':
                        dispatch({ type: 'SET_VIEW', payload: { view: args.featureId }});
                        break;
                    case 'runFeatureWithInput':
                         dispatch({ type: 'SET_VIEW', payload: { view: args.featureId, props: args.props
                         } });
                        break;
                    case 'runWorkspaceAction':
                        try {
                            const result = await executeWorkspaceAction(args.actionId, args.params);
                            setLastResponse(`Action '${args.actionId}' executed successfully.\n\nResult:
                            \`\`\`json\n${JSON.stringify(result, null, 2)}\n\`\`\``);
                        } catch (e) {
                            setLastResponse(`Action failed: ${e instanceof Error ? e.message : 'Unknown
                            error'}`);
                        }
                        break;
                    default:
                        setLastResponse(`Unknown command: ${name}`);
                }
                 setPrompt('');
            } else {
                 setLastResponse(response.text);
            }

        } catch (err) {
            logError(err as Error, { prompt });
            setLastResponse(err instanceof Error ? err.message : 'An unknown error
```

```
        occurred.');
    } finally {
        setIsLoading(false);
    }
}, [prompt, dispatch]);

const handleKeyDown = (e: React.KeyboardEvent) => {
    if (e.key === 'Enter' && !e.shiftKey) {
        e.preventDefault();
        handleCommand();
    }
};

const handleExampleClick = (text: string) => {
    setPrompt(text);
}

return (
    <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
        <header className="mb-6 text-center">
            <h1 className="text-4xl font-extrabold tracking-tight flex items-center justify-
            center">
                <CommandLineIcon />
                <span className="ml-3">AI Command Center</span>
            </h1>
            <p className="mt-2 text-lg text-text-secondary">What would you like to do?</p>
        </header>

        <div className="flex-grow flex flex-col justify-end max-w-3xl w-full mx-auto">
            {lastResponse && (
                <div className="mb-4 p-4 bg-surface rounded-lg text-text-primary border border-
                border">
                    <p><strong>AI:</strong> {lastResponse}</p>
                </div>
            )}
             <div className="relative">
                <textarea
                    value={prompt}
                    onChange={e => setPrompt(e.target.value)}
                    onKeyDown={handleKeyDown}
                    disabled={isLoading}
                    placeholder='Try "explain this code: const a = 1;" or "open the theme designer"'
                    className="w-full p-4 pr-28 rounded-lg bg-surface border border-border
                    focus:ring-2 focus:ring-primary focus:outline-none resize-none shadow-sm"
                    rows={2}
                />
                <button
                    onClick={handleCommand}
                    disabled={isLoading}
                    className="btn-primary absolute right-3 top-1/2 -translate-y-1/2 px-4 py-2"
                >
                   {isLoading ? <LoadingSpinner/> : 'Send'}
                </button>
            </div>
             <div className="flex flex-wrap items-center justify-center gap-2 mt-4">
                <ExamplePromptButton text="Open Theme Designer" onClick={handleExampleClick} />
                <ExamplePromptButton text="Generate a commit for a bug fix"
                onClick={handleExampleClick} />
                <ExamplePromptButton text="Create a regex for email validation"
                onClick={handleExampleClick} />
            </div>
             <p className="text-xs text-text-secondary text-center mt-2">Press Enter to send,
```

```
                Shift+Enter for new line.</p>
            </div>
        </div>
    );
};

// ===== AiPullRequestAssistant_52.tsx =====



import React, { useState, useMemo, useCallback } from 'react';
import * as Diff from 'diff';
import { generatePrSummaryStructured, generateTechnicalSpecFromDiff,
downloadFile, createDocument, insertText } from '../../services/index.ts';
import type { StructuredPrSummary } from '../../types.ts';
import { AiPullRequestAssistantIcon, DocumentIcon } from '../icons.tsx';
import { LoadingSpinner } from '../shared/index.tsx';
import { useNotification } from '../../contexts/NotificationContext.tsx';
import { useGlobalState } from '../../contexts/GlobalStateContext.tsx';

const exampleBefore = `function Greeter(props) {
  return <h1>Hello, {props.name}!</h1>;
}`;
const exampleAfter = `function Greeter({ name, enthusiasmLevel = 1 }) {
  const punctuation = '!'.repeat(enthusiasmLevel);
  return <h1>Hello, {name}{punctuation}</h1>;
}`;

export const AiPullRequestAssistant: React.FC = () => {
    const [beforeCode, setBeforeCode] = useState<string>(exampleBefore);
    const [afterCode, setAfterCode] = useState<string>(exampleAfter);
    const [isLoading, setIsLoading] = useState<boolean>(false);
    const [isExporting, setIsExporting] = useState<boolean>(false);
    const [error, setError] = useState<string>('');
    const [summary, setSummary] = useState<StructuredPrSummary | null>(null);

    const { addNotification } = useNotification();
    const { state } = useGlobalState();
    const { user } = state;

    const diff = useMemo(() => Diff.createPatch('component.tsx', beforeCode,
    afterCode), [beforeCode, afterCode]);

    const handleGenerateSummary = useCallback(async () => {
        if (!beforeCode.trim() && !afterCode.trim()) {
            setError('Please provide code to generate a summary.');
            return;
        }
        setIsLoading(true);
        setError('');
        setSummary(null);

        try {
            const result: StructuredPrSummary = await generatePrSummaryStructured(diff);
            setSummary(result);
        } catch (err) {
            const errorMessage = err instanceof Error ? err.message : 'An unknown error
            occurred.';
            setError(`Failed to generate summary: ${errorMessage}`);
        } finally {
            setIsLoading(false);
        }
```

```
}, [diff, beforeCode, afterCode]);

const handleExportToDocs = async () => {
    if (!summary || !user) {
        addNotification('Please generate a summary first and ensure you are signed in.',
        'error');
        return;
    }
    setIsExporting(true);
    try {
        const specContent = await generateTechnicalSpecFromDiff(diff, summary);
        const doc = await createDocument(`Tech Spec: ${summary.title}`);
        await insertText(doc.documentId, specContent);
        addNotification('Successfully exported to Google Docs!', 'success');
        window.open(doc.webViewLink, '_blank');
    } catch (err) {
        const errorMessage = err instanceof Error ? err.message : 'An unknown error
        occurred.';
        addNotification(`Failed to export: ${errorMessage}`, 'error');
    } finally {
        setIsExporting(false);
    }
};

return (
    <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
        <header className="mb-6">
            <h1 className="text-3xl font-bold flex items-center">
                <AiPullRequestAssistantIcon />
                <span className="ml-3">AI Pull Request Assistant</span>
            </h1>
            <p className="text-text-secondary mt-1">Generate a PR summary from code changes
            and export a full tech spec.</p>
        </header>
        <div className="flex-grow grid grid-cols-1 lg:grid-cols-2 gap-6 min-h-0">
            {/* Left side: Inputs and Generator */}
            <div className="flex flex-col gap-4 min-h-0">
                <div className="flex flex-col flex-1 min-h-0">
                    <label htmlFor="before-code" className="text-sm font-medium text-text-secondary
                    mb-2">Before</label>
                    <textarea id="before-code" value={beforeCode} onChange={e =>
                    setBeforeCode(e.target.value)} className="flex-grow p-4 bg-surface border
                    border-border rounded-md resize-none font-mono text-sm" />
                </div>
                <div className="flex flex-col flex-1 min-h-0">
                    <label htmlFor="after-code" className="text-sm font-medium text-text-secondary
                    mb-2">After</label>
                    <textarea id="after-code" value={afterCode} onChange={e =>
                    setAfterCode(e.target.value)} className="flex-grow p-4 bg-surface border border-
                    border rounded-md resize-none font-mono text-sm" />
                </div>
                <button
                    onClick={handleGenerateSummary}
                    disabled={isLoading}
                    className="btn-primary w-full mt-4 flex items-center justify-center gap-2 py-2"
                >
                    {isLoading ? <LoadingSpinner/> : 'Generate Summary'}
                </button>
            </div>
            {/* Right side: Summary and actions */}
            <div className="flex flex-col gap-4 min-h-0">
                <div className="flex justify-between items-center">
```

```
                        <h3 className="text-lg font-bold">Generated Summary</h3>
                        {summary && (
                             <button
                                onClick={() => downloadFile(`${summary.title}\n\n${summary.summary}\n\n${summary
                                .changes.join('\n')}`, `pr_${summary.title.slice(0,10)}.md`, 'text/markdown')}
                                disabled={!summary}
                                className="btn-primary px-3 py-1 text-sm disabled:bg-gray-400"
                            >
                                Download .md
                           </button>
                        )}
                    </div>
                    <div className="flex-grow bg-surface p-4 border rounded-lg overflow-y-auto">
                        {isLoading && <div className="flex justify-center items-center
                        h-full"><LoadingSpinner/></div>}
                        {error && <p className="text-red-500">{error}</p>}
                        {summary && (
                            <div className="space-y-4">
                                <input type="text" value={summary.title} readOnly className="w-full p-2 bg-
                                background border rounded font-bold"/>
                                <div className="p-2 bg-background border rounded space-y-2">
                                    <h4 className="font-semibold">Summary</h4>
                                    <p className="text-sm">{summary.summary}</p>
                                    <h4 className="font-semibold">Changes</h4>
                                    <ul className="list-disc list-inside text-sm">
                                        {summary.changes.map((c, i) => <li key={i}>{c}</li>)}
                                    </ul>
                                </div>
                            </div>
                        )}
                    </div>
                     <button
                        onClick={handleExportToDocs}
                        disabled={isExporting || !summary || !user}
                        className="btn-primary w-full mt-4 flex items-center justify-center gap-2 py-2"
                        title={!user ? "Sign in to export to Google Docs" : !summary ? "Generate summary
                        first" : "Export a full tech spec to Google Docs"}
                    >
                        {isExporting ? <LoadingSpinner/> : <DocumentIcon/>} Export Tech Spec to Docs
                    </button>
                </div>
            </div>
        </div>
    );
};


// ===== OneClickRefactor_27.tsx =====

import React, { useState, useCallback } from 'react';
import * as Diff from 'diff';
import { refactorForPerformance, refactorForReadability, generateJsDoc,
convertToFunctionalComponent } from '../../services/aiService.ts';
import { SparklesIcon } from '../icons.tsx';
import { LoadingSpinner } from '../shared/index.tsx';

type RefactorAction = 'readability' | 'performance' | 'jsdoc' | 'functional' |
'custom';

const exampleCode = `const MyComponent = ({ data }) => {
  // A less readable component
  let transformedData = [];
```

```
  for (let i = 0; i < data.length; i++) {
    if (data[i].value > 50) {
      let item = { ...data[i], status: 'high' };
      transformedData.push(item);
    }
  }
  return (
    <div>
      {transformedData.map(d => <p key={d.id}>{d.name}</p>)}
    </div>
  );
}`;

const DiffViewer: React.FC<{ oldCode: string, newCode: string }> = ({ oldCode,
newCode }) => {
    const diff = Diff.diffLines(oldCode, newCode);

    return (
        <pre className="whitespace-pre-wrap font-mono text-xs">
            {diff.map((part, index) => {
                const color = part.added ? 'bg-green-500/20' : part.removed ? 'bg-red-500/20' :
                'bg-transparent';
                return <div key={index} className={color}>{part.value}</div>;
            })}
        </pre>
    );
};


export const OneClickRefactor: React.FC = () => {
    const [code, setCode] = useState(exampleCode);
    const [refactoredCode, setRefactoredCode] = useState('');
    const [loadingAction, setLoadingAction] = useState<RefactorAction | null>(null);

    const handleRefactor = useCallback(async (action: RefactorAction) => {
        if (!code.trim()) return;
        setLoadingAction(action);
        setRefactoredCode('');

        let stream;
        switch(action) {
            case 'readability':
                stream = refactorForReadability(code);
                break;
            case 'performance':
                stream = refactorForPerformance(code);
                break;
            case 'jsdoc':
                stream = generateJsDoc(code);
                break;
            case 'functional':
                stream = convertToFunctionalComponent(code);
                break;
            default:
                setLoadingAction(null);
                return;
        }

        try {
            let fullResponse = '';
            for await (const chunk of stream) {
                fullResponse += chunk;
```

```
                setRefactoredCode(fullResponse.replace(/^```(?:\w+\n)?/, '').replace(/```$/,
                ''));
            }
        } catch (e) {
            console.error(e);
            setRefactoredCode(`// Error during refactoring: ${e instanceof Error ? e.message
            : 'Unknown error'}`);
        } finally {
            setLoadingAction(null);
        }
    }, [code]);

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
            <header className="mb-6">
                <h1 className="text-3xl font-bold flex items-center">
                    <SparklesIcon />
                    <span className="ml-3">One-Click Refactor</span>
                </h1>
                <p className="text-text-secondary mt-1">Apply common refactoring patterns to
                your code with a single click.</p>
            </header>
            <div className="flex items-center justify-center flex-wrap gap-2 mb-4 p-4 bg-
            surface rounded-lg border border-border">
                <button onClick={() => handleRefactor('readability')} disabled={!!loadingAction}
                className="btn-primary px-3 py-1.5 text-sm">{loadingAction === 'readability' ?
                <LoadingSpinner/> : 'Improve Readability'}</button>
                <button onClick={() => handleRefactor('performance')} disabled={!!loadingAction}
                className="btn-primary px-3 py-1.5 text-sm">{loadingAction === 'performance' ?
                <LoadingSpinner/> : 'Boost Performance'}</button>
                <button onClick={() => handleRefactor('jsdoc')} disabled={!!loadingAction}
                className="btn-primary px-3 py-1.5 text-sm">{loadingAction === 'jsdoc' ?
                <LoadingSpinner/> : 'Add JSDoc'}</button>
                <button onClick={() => handleRefactor('functional')} disabled={!!loadingAction}
                className="btn-primary px-3 py-1.5 text-sm">{loadingAction === 'functional' ?
                <LoadingSpinner/> : 'To Functional Component'}</button>
            </div>
            <div className="flex-grow grid grid-cols-1 lg:grid-cols-2 gap-6 min-h-0">
                <div className="flex flex-col">
                    <label className="text-sm font-medium mb-2">Original Code</label>
                    <textarea value={code} onChange={e => setCode(e.target.value)} className="flex-
                    grow p-2 bg-surface border rounded font-mono text-xs"/>
                </div>
                 <div className="flex flex-col">
                    <label className="text-sm font-medium mb-2">Refactored Code</label>
                    <div className="flex-grow p-2 bg-background border rounded overflow-auto">
                        {loadingAction ? <div className="flex justify-center items-center
                        h-full"><LoadingSpinner/></div> : <DiffViewer oldCode={code}
                        newCode={refactoredCode} />}
                    </div>
                </div>
            </div>
        </div>
    );
};

// ===== ApiClientGenerator_2.tsx =====

import React, { useState } from 'react';
import JSZip from 'jszip';
import { generateClientFromApiSchema } from '../../services/aiService.ts';
import { CodeBracketSquareIcon, SparklesIcon } from '../icons.tsx';
```

```
import { LoadingSpinner } from '../shared/index.tsx';
import { useNotification } from '../../contexts/NotificationContext.tsx';
import type { GeneratedFile } from '../../types.ts';
import { MarkdownRenderer } from '../shared/index.tsx';

const exampleSchema = `{
  "openapi": "3.0.0",
  "info": { "title": "Simple API", "version": "1.0.0" },
  "paths": {
    "/users": {
      "get": {
        "summary": "Get all users",
        "responses": { "200": { "description": "A list of users" } }
      }
    }
  }
}`;

export const ApiClientGenerator: React.FC = () => {
    const [schema, setSchema] = useState(exampleSchema);
    const [language, setLanguage] = useState('TypeScript/Fetch');
    const [isLoading, setIsLoading] = useState(false);
    const [error, setError] = useState('');
    const [generatedFiles, setGeneratedFiles] = useState<GeneratedFile[]>([]);
    const [activeFile, setActiveFile] = useState<GeneratedFile | null>(null);
    const { addNotification } = useNotification();

    const handleGenerate = async () => {
        if (!schema.trim()) {
            setError('Please provide a schema.');
            return;
        }
        setIsLoading(true);
        setError('');
        setGeneratedFiles([]);
        setActiveFile(null);
        try {
            const files = await generateClientFromApiSchema(schema, language);
            setGeneratedFiles(files);
            if (files.length > 0) {
                setActiveFile(files[0]);
            }
            addNotification('API client generated!', 'success');
        } catch (err) {
            const msg = err instanceof Error ? err.message : 'An unknown error occurred';
            setError(msg);
            addNotification(msg, 'error');
        } finally {
            setIsLoading(false);
        }
    };

    const handleDownloadZip = async () => {
        if (generatedFiles.length === 0) return;
        const zip = new JSZip();
        generatedFiles.forEach(file => {
            zip.file(file.filePath, file.content);
        });

        const zipBlob = await zip.generateAsync({ type: 'blob' });
        const link = document.createElement('a');
        link.href = URL.createObjectURL(zipBlob);
```

```
        link.download = 'api-client.zip';
        document.body.appendChild(link);
        link.click();
        document.body.removeChild(link);
    };

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
            <header className="mb-6">
                <h1 className="text-3xl font-bold flex items-center"><CodeBracketSquareIcon
                /><span className="ml-3">API Client Generator</span></h1>
                <p className="text-text-secondary mt-1">Generate a typed client library from an
                OpenAPI or GraphQL schema.</p>
            </header>
            <div className="flex-grow grid grid-cols-1 lg:grid-cols-2 gap-6 min-h-0">
                <div className="flex flex-col gap-4">
                    <div className="flex flex-col flex-1 min-h-0">
                        <label className="text-sm font-medium mb-2">OpenAPI / GraphQL Schema</label>
                        <textarea value={schema} onChange={e => setSchema(e.target.value)}
                        className="flex-grow p-2 bg-surface border rounded font-mono text-xs"/>
                    </div>
                    <div>
                        <label className="text-sm font-medium">Target Language & Library</label>
                        <select value={language} onChange={e => setLanguage(e.target.value)}
                        className="w-full mt-1 p-2 bg-surface border rounded">
                            <option>TypeScript/Fetch</option>
                            <option>Python/requests</option>
                        </select>
                    </div>
                    <button onClick={handleGenerate} disabled={isLoading} className="btn-primary
                    w-full py-3">{isLoading ? <LoadingSpinner/> : 'Generate Client'}</button>
                    {error && <p className="text-red-500 text-xs text-center">{error}</p>}
                </div>
                <div className="flex flex-col min-h-0">
                    <div className="flex justify-between items-center mb-2">
                        <label className="text-sm font-medium">Generated Files</label>
                        {generatedFiles.length > 0 && <button onClick={handleDownloadZip}
                        className="btn-primary px-3 py-1 text-sm">Download as ZIP</button>}
                    </div>
                    <div className="flex-grow grid grid-cols-3 gap-2 min-h-0 bg-background border
                    rounded-lg p-2">
                        <div className="col-span-1 overflow-y-auto">
                            {generatedFiles.map(file => (
                                <div key={file.filePath} onClick={() => setActiveFile(file)} className={`p-2
                                text-sm rounded cursor-pointer ${activeFile?.filePath === file.filePath ? 'bg-
                                primary/10 text-primary' : ''}`}>{file.filePath}</div>
                            ))}
                        </div>
                        <div className="col-span-2 bg-surface rounded overflow-y-auto">
                            {activeFile && <MarkdownRenderer content={'```\n' + activeFile.content +
                            '\n```'} />}
                        </div>
                    </div>
                </div>
            </div>
        </div>
    );
};


// ===== SqlToApiGenerator_2.tsx =====
import React, { useState } from 'react';
```

```
import { sqlToApiEndpoints } from '../../services/aiService.ts';
import type { GeneratedFile } from '../../types.ts';
import { ServerStackIcon } from '../icons.tsx';
import { LoadingSpinner, MarkdownRenderer } from '../shared/index.tsx';
import { useNotification } from '../../contexts/NotificationContext.tsx';

const exampleSchema = `CREATE TABLE users (
    id SERIAL PRIMARY KEY,
    username VARCHAR(255) UNIQUE NOT NULL,
    email VARCHAR(255) UNIQUE NOT NULL,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP
);`;

export const SqlToApiGenerator: React.FC = () => {
    const [schema, setSchema] = useState(exampleSchema);
    const [framework, setFramework] = useState<'express' | 'fastify'>('express');
    const [files, setFiles] = useState<GeneratedFile[]>([]);
    const [activeTab, setActiveTab] = useState<string | null>(null);
    const [isLoading, setIsLoading] = useState(false);
    const { addNotification } = useNotification();

    const handleGenerate = async () => {
        if (!schema.trim()) {
            addNotification('Please provide a SQL schema.', 'error');
            return;
        }
        setIsLoading(true);
        setFiles([]);
        setActiveTab(null);
        try {
            const result = await sqlToApiEndpoints(schema, framework);
            setFiles(result);
            if (result.length > 0) {
                setActiveTab(result[0].filePath);
            }
            addNotification('API files generated!', 'success');
        } catch (err) {
            addNotification(err instanceof Error ? err.message : 'Failed to generate API',
            'error');
        } finally {
            setIsLoading(false);
        }
    };

    const activeFile = files.find(f => f.filePath === activeTab);

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
            <header className="mb-6">
                <h1 className="text-3xl font-bold flex items-center"><ServerStackIcon /><span
                className="ml-3">SQL to API Generator</span></h1>
                <p className="text-text-secondary mt-1">Generate boilerplate CRUD API endpoints
                from a SQL schema.</p>
            </header>
            <div className="flex-grow grid grid-cols-1 lg:grid-cols-2 gap-6 min-h-0">
                <div className="flex flex-col gap-4">
                    <div className="flex flex-col flex-1 min-h-0">
                        <label className="text-sm font-medium mb-2">SQL Schema</label>
                        <textarea value={schema} onChange={e => setSchema(e.target.value)}
                        className="flex-grow p-2 bg-surface border rounded font-mono text-xs"/>
                    </div>
                    <div>
```

```
                        <label className="text-sm font-medium">Framework</label>
                        <select value={framework} onChange={e => setFramework(e.target.value as
                        'express' | 'fastify')} className="w-full mt-1 p-2 bg-surface border rounded">
                            <option value="express">Express</option>
                            <option value="fastify">Fastify</option>
                        </select>
                    </div>
                    <button onClick={handleGenerate} disabled={isLoading} className="btn-primary
                    w-full py-3">{isLoading ? <LoadingSpinner/> : 'Generate API Files'}</button>
                </div>
                <div className="flex flex-col">
                    <div className="flex border-b border-border">
                        {files.map(file => (
                            <button key={file.filePath} onClick={() => setActiveTab(file.filePath)}
                            className={`px-4 py-2 text-sm ${activeTab === file.filePath ? 'bg-background
                            border-b-2 border-primary text-text-primary' : 'text-text-secondary'}`}>
                                {file.filePath}
                            </button>
                        ))}
                    </div>
                    <div className="flex-grow bg-background border border-t-0 rounded-b-md overflow-
                    auto">
                        {isLoading ? <div className="flex justify-center items-center
                        h-full"><LoadingSpinner /></div> : (
                            activeFile ? <MarkdownRenderer content={'```javascript\n' + activeFile.content +
                            '\n```'} /> : <div className="p-4 text-text-secondary">Generated code will
                            appear here.</div>
                        )}
                    </div>
                </div>
            </div>
        </div>
    );
};


// ===== BlamelessPostmortemGenerator_2.tsx =====

import React, { useState } from 'react';
import { generatePostmortem } from '../../services/aiService.ts';
import { DocumentTextIcon } from '../icons.tsx';
import { LoadingSpinner, MarkdownRenderer } from '../shared/index.tsx';

interface ActionItem {
    id: number;
    description: string;
    assignee: string;
}

export const BlamelessPostmortemGenerator: React.FC = () => {
    const [title, setTitle] = useState('Production Database Outage');
    const [timeline, setTimeline] = useState('2024-07-15 14:30 UTC - PagerDuty alert
    received.\n2024-07-15 14:35 UTC - On-call engineer acknowledges.\n2024-07-15
    14:50 UTC - Root cause identified as a bad migration script.\n2024-07-15 15:00
    UTC - Rollback initiated.\n2024-07-15 15:10 UTC - Service restored.');
    const [rootCause, setRootCause] = useState('A database migration script
    contained a locking issue that caused a cascading failure under heavy load.');
    const [impact, setImpact] = useState('All API services were unavailable for 40
    minutes, affecting 100% of users.');
    const [actionItems, setActionItems] = useState<ActionItem[]>([{ id: 1,
    description: 'Add stricter linting rules for migration scripts.', assignee: 'DB
    Team' }]);
```

```
const [generatedDoc, setGeneratedDoc] = useState('');
const [isLoading, setIsLoading] = useState(false);

const handleGenerate = async () => {
    setIsLoading(true);
    setGeneratedDoc('');
    try {
        const doc = await generatePostmortem({
            title,
            timeline,
            rootCause,
            impact,
            actionItems: actionItems.filter(a => a.description)
        });
        setGeneratedDoc(doc);
    } catch (e) {
        setGeneratedDoc(`Error generating document: ${e instanceof Error ? e.message :
        'Unknown error'}`);
    } finally {
        setIsLoading(false);
    }
};

const handleActionItemChange = (id: number, field: 'description' | 'assignee',
value: string) => {
    setActionItems(items => items.map(item => item.id === id ? { ...item, [field]:
    value } : item));
};

const addActionItem = () => {
    setActionItems(items => [...items, { id: Date.now(), description: '', assignee:
    '' }]);
};

return (
    <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
        <header className="mb-6">
            <h1 className="text-3xl font-bold flex items-center"><DocumentTextIcon /><span
            className="ml-3">Blameless Post-mortem Generator</span></h1>
            <p className="text-text-secondary mt-1">A wizard to guide you through creating a
            formal, blameless post-mortem report.</p>
        </header>
        <div className="flex-grow grid grid-cols-1 lg:grid-cols-2 gap-6 min-h-0">
            <div className="flex flex-col gap-3 overflow-y-auto pr-2">
                <h3 className="text-xl font-bold">Incident Details</h3>
                <div><label className="text-sm font-medium">1. Incident Title</label><input
                type="text" value={title} onChange={e => setTitle(e.target.value)}
                className="w-full mt-1 p-2 bg-surface border rounded"/></div>
                <div><label className="text-sm font-medium">2. Timeline (one event per
                line)</label><textarea value={timeline} onChange={e =>
                setTimeline(e.target.value)} className="w-full mt-1 p-2 bg-surface border
                rounded h-24"/></div>
                <div><label className="text-sm font-medium">3. Root Cause
                Analysis</label><textarea value={rootCause} onChange={e =>
                setRootCause(e.target.value)} className="w-full mt-1 p-2 bg-surface border
                rounded h-20"/></div>
                <div><label className="text-sm font-medium">4. Impact</label><textarea
                value={impact} onChange={e => setImpact(e.target.value)} className="w-full mt-1
                p-2 bg-surface border rounded h-16"/></div>
                <div>
                    <label className="text-sm font-medium">5. Action Items</label>
                    {actionItems.map(item => (
```

```
                            <div key={item.id} className="flex gap-2 mt-1">
                                <input value={item.description} onChange={e => handleActionItemChange(item.id,
                                'description', e.target.value)} placeholder="Description" className="flex-grow
                                p-2 bg-surface border rounded text-sm"/>
                                <input value={item.assignee} onChange={e => handleActionItemChange(item.id,
                                'assignee', e.target.value)} placeholder="Assignee" className="w-32 p-2 bg-
                                surface border rounded text-sm"/>
                            </div>
                        ))}
                        <button onClick={addActionItem} className="text-sm text-primary mt-1">+ Add
                        Item</button>
                    </div>
                    <button onClick={handleGenerate} disabled={isLoading} className="btn-primary
                    w-full py-3 mt-2">{isLoading ? <LoadingSpinner/> : 'Generate Report'}</button>
                </div>
                <div className="flex flex-col">
                    <label className="text-sm font-medium mb-2">Generated Report</label>
                    <div className="flex-grow p-4 bg-background border rounded overflow-auto">
                        {isLoading ? <div className="flex justify-center items-center
                        h-full"><LoadingSpinner /></div> : <MarkdownRenderer content={generatedDoc} />}
                    </div>
                </div>
            </div>
        </div>
    );
};

// ===== DataAnonymizer_2.tsx =====

import React, { useState } from 'react';
import { anonymizeData } from '../../services/aiService.ts';
import { ShieldCheckIcon } from '../icons.tsx';
import { LoadingSpinner } from '../shared/index.tsx';
import { useNotification } from '../../contexts/NotificationContext.tsx';

const exampleJson = `[
  { "id": 1, "name": "Alice Smith", "email": "alice.smith@example.com",
  "ip_address": "192.168.1.10" },
  { "id": 2, "name": "Bob Johnson", "email": "bob.j@workplace.net", "ip_address":
  "10.0.0.54" }
]`;

export const DataAnonymizer: React.FC = () => {
    const [data, setData] = useState(exampleJson);
    const [anonymizedData, setAnonymizedData] = useState('');
    const [targets, setTargets] = useState({ name: true, email: true, ip_address:
    true, phone: true });
    const [isLoading, setIsLoading] = useState(false);
    const { addNotification } = useNotification();

    const handleAnonymize = async () => {
        const selectedTargets = Object.entries(targets).filter(([, value]) =>
        value).map(([key]) => key);
        if (selectedTargets.length === 0) {
            addNotification('Please select at least one field to anonymize.', 'error');
            return;
        }
        setIsLoading(true);
        setAnonymizedData('');
        try {
            const result = await anonymizeData(data, selectedTargets);
            setAnonymizedData(result.anonymizedData);
```

```
            addNotification('Data anonymized successfully!', 'success');
        } catch (err) {
            addNotification(err instanceof Error ? err.message : 'Anonymization failed.',
            'error');
        } finally {
            setIsLoading(false);
        }
    };

    const handleTargetChange = (key: string) => {
        setTargets(prev => ({ ...prev, [key]: !prev[key] }));
    };

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
            <header className="mb-6">
                <h1 className="text-3xl font-bold flex items-center"><ShieldCheckIcon /><span
                className="ml-3">Data Anonymizer</span></h1>
                <p className="text-text-secondary mt-1">Anonymize sensitive data in JSON or CSV
                formats using AI.</p>
            </header>
            <div className="flex-grow grid grid-cols-1 lg:grid-cols-2 gap-6 min-h-0">
                <div className="flex flex-col gap-4">
                    <div className="flex flex-col flex-1 min-h-0">
                        <label className="text-sm font-medium mb-2">Original Data (JSON or CSV)</label>
                        <textarea value={data} onChange={e => setData(e.target.value)} className="flex-
                        grow p-2 bg-surface border rounded font-mono text-xs"/>
                    </div>
                    <div>
                        <label className="text-sm font-medium mb-2">Fields to Anonymize</label>
                        <div className="flex flex-wrap gap-2">
                            {Object.keys(targets).map(key => (
                                <label key={key} className="flex items-center gap-2 p-2 bg-surface border
                                rounded-md text-sm cursor-pointer">
                                    <input type="checkbox" checked={targets[key as keyof typeof targets]}
                                    onChange={() => handleTargetChange(key)} className="rounded text-primary
                                    focus:ring-primary" />
                                    {key}
                                </label>
                            ))}
                        </div>
                    </div>
                    <button onClick={handleAnonymize} disabled={isLoading} className="btn-primary
                    w-full py-3">{isLoading ? <LoadingSpinner/> : 'Anonymize Data'}</button>
                </div>
                <div className="flex flex-col">
                    <label className="text-sm font-medium mb-2">Anonymized Data</label>
                    <div className="flex-grow p-2 bg-background border rounded overflow-auto">
                        {isLoading ? <div className="flex justify-center items-center
                        h-full"><LoadingSpinner /></div> : <pre className="font-mono text-xs whitespace-
                        pre-wrap">{anonymizedData}</pre>}
                    </div>
                </div>
            </div>
        </div>
    );
};


// ===== ABTestAssistant_2.tsx =====

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

```
import { generateABTestWrapper } from '../../services/aiService.ts';
import { BeakerIcon } from '../icons.tsx';
import { LoadingSpinner, MarkdownRenderer } from '../shared/index.tsx';

const variantA_example = `<button className="bg-blue-500 text-white p-2
rounded">Sign Up</button>`;
const variantB_example = `<button className="bg-green-500 text-white p-2
rounded">Get Started</button>`;

export const ABTestAssistant: React.FC = () => {
    const [variantA, setVariantA] = useState(variantA_example);
    const [variantB, setVariantB] = useState(variantB_example);
    const [service, setService] = useState('LaunchDarkly');
    const [generatedCode, setGeneratedCode] = useState('');
    const [isLoading, setIsLoading] = useState(false);
    const [error, setError] = useState('');

    const handleGenerate = async () => {
        setIsLoading(true);
        setError('');
        setGeneratedCode('');
        try {
            const code = await generateABTestWrapper(variantA, variantB, service);
            setGeneratedCode(code);
        } catch (err) {
            setError(err instanceof Error ? err.message : 'Failed to generate wrapper
            component');
        } finally {
            setIsLoading(false);
        }
    };

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
            <header className="mb-6">
                <h1 className="text-3xl font-bold flex items-center"><BeakerIcon /><span
                className="ml-3">A/B Test Assistant</span></h1>
                <p className="text-text-secondary mt-1">Generate code snippets for A/B tests
                using a feature flagging service.</p>
            </header>
            <div className="flex-grow grid grid-cols-1 lg:grid-cols-2 gap-6 min-h-0">
                <div className="flex flex-col gap-4">
                    <div className="flex flex-col flex-1 min-h-0">
                        <label className="text-sm font-medium mb-2">Variant A Code</label>
                        <textarea value={variantA} onChange={e => setVariantA(e.target.value)}
                        className="flex-grow p-2 bg-surface border rounded font-mono text-xs"/>
                    </div>
                     <div className="flex flex-col flex-1 min-h-0">
                        <label className="text-sm font-medium mb-2">Variant B Code</label>
                        <textarea value={variantB} onChange={e => setVariantB(e.target.value)}
                        className="flex-grow p-2 bg-surface border rounded font-mono text-xs"/>
                    </div>
                     <div>
                        <label className="text-sm font-medium">Feature Flag Service</label>
                        <select value={service} onChange={e => setService(e.target.value)}
                        className="w-full mt-1 p-2 bg-surface border rounded">
                            <option>LaunchDarkly</option>
                            <option>PostHog</option>
                            <option>Generic</option>
                        </select>
                    </div>
                    <button onClick={handleGenerate} disabled={isLoading} className="btn-primary
```

```
                    w-full py-3">{isLoading ? <LoadingSpinner/> : 'Generate Wrapper
                    Component'}</button>
                </div>
                <div className="flex flex-col">
                    <label className="text-sm font-medium mb-2">Generated Wrapper Component</label>
                    <div className="flex-grow p-1 bg-background border rounded overflow-auto">
                        {isLoading ? <div className="flex justify-center items-center
                        h-full"><LoadingSpinner /></div> : (
                            <>
                                {error && <p className="text-red-500 p-4">{error}</p>}
                                {generatedCode && <MarkdownRenderer content={generatedCode} />}
                            </>
                        )}
                    </div>
                </div>
            </div>
        </div>
    );
};


// ===== I18nHelper_2.tsx =====

import React, { useState } from 'react';
import { extractStringsForI18n } from '../../services/aiService.ts';
import { ProjectExplorerIcon } from '../icons.tsx';
import { LoadingSpinner, MarkdownRenderer } from '../shared/index.tsx';
import { useNotification } from '../../contexts/NotificationContext.tsx';

const exampleCode = `import React from 'react';

function WelcomeCard() {
  return (
    <div>
      <h1>Welcome to our App!</h1>
      <p>Click the button below to get started.</p>
      <button>Get Started</button>
    </div>
  );
}`;

export const I18nHelper: React.FC = () => {
    const [code, setCode] = useState(exampleCode);
    const [i18nJson, setI18nJson] = useState<Record<string, string> | null>(null);
    const [refactoredCode, setRefactoredCode] = useState('');
    const [isLoading, setIsLoading] = useState(false);
    const { addNotification } = useNotification();

    const handleExtract = async () => {
        setIsLoading(true);
        setI18nJson(null);
        setRefactoredCode('');
        try {
            const result = await extractStringsForI18n(code);
            setI18nJson(result.i18nJson);
            setRefactoredCode(result.refactoredCode);
            addNotification('Strings extracted successfully!', 'success');
        } catch (err) {
            addNotification(err instanceof Error ? err.message : 'Extraction failed.',
            'error');
        } finally {
            setIsLoading(false);
```

```
        }
    };

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
            <header className="mb-6">
                <h1 className="text-3xl font-bold flex items-center"><ProjectExplorerIcon
                /><span className="ml-3">i18n Helper</span></h1>
                <p className="text-text-secondary mt-1">Extract strings from a component and
                generate JSON language files.</p>
            </header>
            <div className="flex-grow grid grid-cols-1 lg:grid-cols-2 gap-6 min-h-0">
                <div className="flex flex-col">
                    <label className="text-sm font-medium mb-2">Component Code</label>
                    <textarea value={code} onChange={e => setCode(e.target.value)} className="flex-
                    grow p-2 bg-surface border rounded font-mono text-xs"/>
                    <button onClick={handleExtract} disabled={isLoading} className="btn-primary
                    w-full mt-4 py-3">{isLoading ? <LoadingSpinner/> : 'Extract Strings'}</button>
                </div>
                <div className="flex flex-col gap-4">
                    <div className="flex flex-col flex-1 min-h-0">
                        <label className="text-sm font-medium mb-2">Generated i18n JSON</label>
                        <div className="flex-grow p-1 bg-background border rounded overflow-auto">
                            {isLoading && !i18nJson ? <div className="flex justify-center items-center
                            h-full"><LoadingSpinner /></div> : (
                                i18nJson && <MarkdownRenderer content={'```json\n' + JSON.stringify(i18nJson,
                                null, 2) + '\n```'} />
                            )}
                        </div>
                    </div>
                     <div className="flex flex-col flex-1 min-h-0">
                        <label className="text-sm font-medium mb-2">Refactored Component</label>
                        <div className="flex-grow p-1 bg-background border rounded overflow-auto">
                            {refactoredCode && <MarkdownRenderer content={'```tsx\n' + refactoredCode +
                            '\n```'} />}
                        </div>
                    </div>
                </div>
            </div>
        </div>
    );
};


// ===== TokenUsageEstimator_2.tsx =====

import React, { useState } from 'react';
import { CpuChipIcon } from '../icons.tsx';
import { estimateTokenCount } from '../../services/index.ts';
import { useNotification } from '../../contexts/NotificationContext.tsx';
import { LoadingSpinner } from '../shared/index.tsx';

export const TokenUsageEstimator: React.FC = () => {
    const [prompt, setPrompt] = useState('');
    const [estimate, setEstimate] = useState(0);
    const [isLoading, setIsLoading] = useState(false);
    const { addNotification } = useNotification();

    const handleEstimate = async () => {
        if (!prompt.trim()) {
            addNotification('Please enter some text to estimate.', 'info');
            return;
```

```
        }
        setIsLoading(true);
        setEstimate(0);
        try {
            const { count } = await estimateTokenCount(prompt);
            setEstimate(count);
        } catch (e) {
            addNotification('Failed to estimate tokens via API.', 'error');
        } finally {
            setIsLoading(false);
        }
    };

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
             <header className="mb-6">
                <h1 className="text-3xl font-bold flex items-center"><CpuChipIcon /><span
                className="ml-3">Token Usage Estimator</span></h1>
                <p className="text-text-secondary mt-1">Estimate Gemini token usage for a given
                prompt/task.</p>
            </header>
            <div className="flex-grow flex flex-col items-center justify-center gap-4">
                <div className="w-full max-w-2xl">
                    <label className="text-sm font-medium mb-2">Prompt</label>
                    <textarea
                        value={prompt}
                        onChange={e => setPrompt(e.target.value)}
                        className="w-full h-48 p-4 bg-surface border rounded-lg"
                        placeholder="Enter your prompt here..."
                    />
                </div>
                <button onClick={handleEstimate} disabled={isLoading} className="btn-primary
                px-6 py-3 min-w-[180px] flex items-center justify-center">
                    {isLoading ? <LoadingSpinner /> : 'Estimate Tokens via API'}
                </button>
                {estimate > 0 && (
                    <div className="mt-4 text-center">
                        <p className="text-4xl font-bold text-primary">{estimate}</p>
                        <p className="text-text-secondary">Estimated Tokens</p>
                    </div>
                )}
            </div>
        </div>
    );
};

// ===== FinancialChartGenerator_2.tsx =====

import React, { useState } from 'react';
import { generateChartComponent } from '../../services/aiService.ts';
import { ChartBarIcon } from '../icons.tsx';
import { LoadingSpinner, MarkdownRenderer } from '../shared/index.tsx';
import { useNotification } from '../../contexts/NotificationContext.tsx';

export const FinancialChartGenerator: React.FC = () => {
    const [csvData, setCsvData] = useState('date,price\n2024-01-01,150\n2024-02-
    01,155\n2024-03-01,162\n2024-04-01,158');
    const [chartType, setChartType] = useState<'line' | 'bar'>('line');
    const [generatedCode, setGeneratedCode] = useState('');
    const [isLoading, setIsLoading] = useState(false);
    const { addNotification } = useNotification();
    const handleGenerate = async () => {
```

```
        setIsLoading(true);
        setGeneratedCode('');
        try {
            const lines = csvData.trim().split('\n');
            const headers = lines[0].split(',');
            const samples = lines.slice(1, 4).map(line => line.split(','));

            const code = await generateChartComponent({ headers, samples }, chartType);
            setGeneratedCode(code);
            addNotification('Chart component generated!', 'success');
        } catch (err) {
            addNotification(err instanceof Error ? err.message : 'Failed to generate chart',
            'error');
        } finally {
            setIsLoading(false);
        }
    };

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
            <header className="mb-6">
                <h1 className="text-3xl font-bold flex items-center"><ChartBarIcon /><span
                className="ml-3">Financial Chart Generator</span></h1>
                <p className="text-text-secondary mt-1">Generate chart components from financial
                data (e.g., CSV).</p>
            </header>
            <div className="flex-grow grid grid-cols-1 lg:grid-cols-2 gap-6 min-h-0">
                <div className="flex flex-col gap-4">
                    <div className="flex flex-col flex-1 min-h-0">
                        <label className="text-sm font-medium mb-2">CSV Data</label>
                        <textarea value={csvData} onChange={e => setCsvData(e.target.value)}
                        className="flex-grow p-2 bg-surface border rounded font-mono text-xs"/>
                    </div>
                     <div>
                        <label className="text-sm font-medium">Chart Type</label>
                        <select value={chartType} onChange={e => setChartType(e.target.value as 'line' |
                        'bar')} className="w-full mt-1 p-2 bg-surface border rounded">
                            <option value="line">Line Chart</option>
                            <option value="bar">Bar Chart</option>
                        </select>
                    </div>
                    <button onClick={handleGenerate} disabled={isLoading} className="btn-primary
                    w-full py-3">{isLoading ? <LoadingSpinner/> : 'Generate Chart
                    Component'}</button>
                </div>
                <div className="flex flex-col">
                    <label className="text-sm font-medium mb-2">Generated React Component (using
                    Recharts)</label>
                    <div className="flex-grow p-1 bg-background border rounded overflow-auto">
                        {isLoading ? <div className="flex justify-center items-center
                        h-full"><LoadingSpinner /></div> : (
                            generatedCode && <MarkdownRenderer content={generatedCode} />
                        )}
                    </div>
                </div>
            </div>
        );
};


// ===== ComplianceReportHelper_2.tsx =====
```

```
import React, { useState } from 'react';
import { generateComplianceReport } from '../../services/aiService.ts';
import { ShieldCheckIcon } from '../icons.tsx';
import { LoadingSpinner, MarkdownRenderer } from '../shared/index.tsx';
import { useNotification } from '../../contexts/NotificationContext.tsx';

const exampleCode = `function saveUserData(user) {
  // Storing user data in localStorage for session persistence
  localStorage.setItem('user_session', JSON.stringify(user));

  // Sending analytics data
  fetch('https://analytics.example.com/track', {
    method: 'POST',
    body: JSON.stringify({ userId: user.id, event: 'login' })
  });
}`;

export const ComplianceReportHelper: React.FC = () => {
    const [code, setCode] = useState(exampleCode);
    const [standard, setStandard] = useState('GDPR');
    const [report, setReport] = useState('');
    const [isLoading, setIsLoading] = useState(false);
    const { addNotification } = useNotification();

    const handleGenerate = async () => {
        setIsLoading(true);
        setReport('');
        try {
            const result = await generateComplianceReport(code, standard);
            setReport(result);
            addNotification('Compliance report generated!', 'success');
        } catch (err) {
            addNotification(err instanceof Error ? err.message : 'Failed to generate
            report', 'error');
        } finally {
            setIsLoading(false);
        }
    };

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
            <header className="mb-6">
                <h1 className="text-3xl font-bold flex items-center"><ShieldCheckIcon /><span
                className="ml-3">Compliance Report Helper</span></h1>
                <p className="text-text-secondary mt-1">Check code for compliance with standards
                like GDPR.</p>
            </header>
            <div className="flex-grow grid grid-cols-1 lg:grid-cols-2 gap-6 min-h-0">
                <div className="flex flex-col gap-4">
                    <div className="flex flex-col flex-1 min-h-0">
                        <label className="text-sm font-medium mb-2">Code to Analyze</label>
                        <textarea value={code} onChange={e => setCode(e.target.value)} className="flex-
                        grow p-2 bg-surface border rounded font-mono text-xs"/>
                    </div>
                    <div>
                        <label className="text-sm font-medium">Compliance Standard</label>
                        <select value={standard} onChange={e => setStandard(e.target.value)}
                        className="w-full mt-1 p-2 bg-surface border rounded">
                            <option>GDPR</option>
                            <option>SOX</option>
                        </select>
                    </div>
```

```
                    <button onClick={handleGenerate} disabled={isLoading} className="btn-primary
                    w-full py-3">{isLoading ? <LoadingSpinner/> : 'Generate Report'}</button>
                </div>
                <div className="flex flex-col">
                    <label className="text-sm font-medium mb-2">Generated Report</label>
                    <div className="flex-grow p-4 bg-background border rounded overflow-auto">
                        {isLoading ? <div className="flex justify-center items-center
                        h-full"><LoadingSpinner /></div> : (
                            report && <MarkdownRenderer content={report} />
                        )}
                    </div>
                </div>
            </div>
        </div>
    );
};


// ===== EcommerceComponentGenerator_2.tsx =====

import React, { useState } from 'react';
import { generateEcommerceComponent } from '../../services/aiService.ts';
import { ArchiveBoxIcon } from '../icons.tsx';
import { LoadingSpinner, MarkdownRenderer } from '../shared/index.tsx';
import { useNotification } from '../../contexts/NotificationContext.tsx';

export const EcommerceComponentGenerator: React.FC = () => {
    const [description, setDescription] = useState('a product page for a red shoe
    with three images and a price of $99');
    const [generatedCode, setGeneratedCode] = useState('');
    const [isLoading, setIsLoading] = useState(false);
    const { addNotification } = useNotification();

    const handleGenerate = async () => {
        setIsLoading(true);
        setGeneratedCode('');
        try {
            const code = await generateEcommerceComponent(description);
            setGeneratedCode(code);
            addNotification('Component generated!', 'success');
        } catch (err) {
            addNotification(err instanceof Error ? err.message : 'Failed to generate
            component', 'error');
        } finally {
            setIsLoading(false);
        }
    };

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
            <header className="mb-6">
                <h1 className="text-3xl font-bold flex items-center"><ArchiveBoxIcon /><span
                className="ml-3">E-commerce Component Generator</span></h1>
                <p className="text-text-secondary mt-1">Generate a product display component
                with schema.org markup.</p>
            </header>
            <div className="flex-grow flex flex-col gap-4 min-h-0">
                <div className="flex flex-col flex-1 min-h-0">
                    <label className="text-sm font-medium mb-2">Product Description</label>
                    <textarea value={description} onChange={e => setDescription(e.target.value)}
                    className="flex-grow p-2 bg-surface border rounded"/>
                </div>
```

```
                <button onClick={handleGenerate} disabled={isLoading} className="btn-primary
                w-full max-w-sm mx-auto py-3">{isLoading ? <LoadingSpinner/> : 'Generate
                Component'}</button>
                <div className="flex flex-col flex-grow min-h-0 mt-4">
                    <label className="text-sm font-medium mb-2">Generated React Component</label>
                    <div className="flex-grow p-1 bg-background border rounded overflow-auto">
                        {isLoading ? <div className="flex justify-center items-center
                        h-full"><LoadingSpinner /></div> : (
                            generatedCode && <MarkdownRenderer content={generatedCode} />
                        )}
                    </div>
                </div>
            </div>
        </div>
    );
};


// ===== ApiEndpointTester_2.tsx =====

import React, { useState } from 'react';
import { PaperAirplaneIcon } from '../icons.tsx';
import { LoadingSpinner } from '../shared/index.tsx';
import { useNotification } from '../../contexts/NotificationContext.tsx';

export const ApiEndpointTester: React.FC = () => {
    const [url, setUrl] = useState('https://jsonplaceholder.typicode.com/posts/1');
    const [method, setMethod] = useState('GET');
    const [headers, setHeaders] = useState('{\n  "Content-Type":
    "application/json"\n}');
    const [body, setBody] = useState('');
    const [response, setResponse] = useState<any>(null);
    const [status, setStatus] = useState<number | null>(null);
    const [time, setTime] = useState<number | null>(null);
    const [isLoading, setIsLoading] = useState(false);
    const { addNotification } = useNotification();

    const handleSend = async () => {
        setIsLoading(true);
        setResponse(null);
        setStatus(null);
        setTime(null);

        const startTime = performance.now();
        try {
            const parsedHeaders = headers ? JSON.parse(headers) : {};
            const res = await fetch(url, {
                method,
                headers: parsedHeaders,
                body: method !== 'GET' && body ? body : undefined,
            });

            const endTime = performance.now();
            setTime(endTime - startTime);
            setStatus(res.status);

            const resBody = await res.json();
            setResponse(JSON.stringify(resBody, null, 2));
            addNotification('Request successful!', 'success');
        } catch (err) {
            const endTime = performance.now();
            setTime(endTime - startTime);
```

```
            const msg = err instanceof Error ? err.message : 'An unknown error occurred';
            setResponse(msg);
            addNotification(msg, 'error');
        } finally {
            setIsLoading(false);
        }
    };

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
            <header className="mb-6">
                <h1 className="text-3xl font-bold flex items-center"><PaperAirplaneIcon /><span
                className="ml-3">API Endpoint Tester</span></h1>
                <p className="text-text-secondary mt-1">A simple UI to make requests to API
                endpoints (like Postman lite).</p>
            </header>
            <div className="flex-grow grid grid-cols-1 lg:grid-cols-2 gap-6 min-h-0">
                <div className="flex flex-col gap-4">
                    <div className="flex gap-2">
                        <select value={method} onChange={e => setMethod(e.target.value)} className="p-2
                        bg-surface border rounded">
                            <option>GET</option><option>POST</option><option>PUT</option><option>DELETE</opt
                            ion>
                        </select>
                        <input value={url} onChange={e => setUrl(e.target.value)}
                        placeholder="https://api.example.com/data" className="flex-grow p-2 bg-surface
                        border rounded"/>
                    </div>
                    <div className="flex flex-col flex-1 min-h-0">
                        <label className="text-sm mb-1">Headers (JSON)</label>
                        <textarea value={headers} onChange={e => setHeaders(e.target.value)}
                        className="flex-grow p-2 bg-surface border rounded font-mono text-xs"/>
                    </div>
                    <div className="flex flex-col flex-1 min-h-0">
                        <label className="text-sm mb-1">Body (JSON)</label>
                        <textarea value={body} onChange={e => setBody(e.target.value)} className="flex-
                        grow p-2 bg-surface border rounded font-mono text-xs"/>
                    </div>
                    <button onClick={handleSend} disabled={isLoading} className="btn-primary w-full
                    py-3">{isLoading ? <LoadingSpinner/> : 'Send Request'}</button>
                </div>
                <div className="flex flex-col">
                    <div className="flex justify-between items-center mb-2">
                        <label className="text-sm font-medium">Response</label>
                        {status && <div className="flex gap-4 text-sm"><span className={status >= 200 &&
                        status < 300 ? 'text-green-500' : 'text-red-500'}>Status:
                        {status}</span><span>Time: {time?.toFixed(0)}ms</span></div>}
                    </div>
                    <div className="flex-grow p-2 bg-background border rounded overflow-auto">
                        {isLoading ? <div className="flex justify-center items-center
                        h-full"><LoadingSpinner /></div> : <pre className="font-mono text-
                        xs">{response}</pre>}
                    </div>
                </div>
            </div>
        </div>
    );
};


// ===== StoryboardGenerator_2.tsx =====
import React, { useState } from 'react';
```

```
import { decomposeUserFlow, generateImage } from '../../services/aiService.ts';
import { PhotoIcon } from '../icons.tsx';
import { LoadingSpinner } from '../shared/index.tsx';
import { useNotification } from '../../contexts/NotificationContext.tsx';

interface StoryboardFrame {
    description: string;
    imageUrl: string;
}

export const StoryboardGenerator: React.FC = () => {
    const [flow, setFlow] = useState('User logs in, sees a dashboard with three
    widgets, and clicks a button to open a settings modal.');
    const [frames, setFrames] = useState<StoryboardFrame[]>([]);
    const [isLoading, setIsLoading] = useState(false);
    const [progress, setProgress] = useState('');
    const { addNotification } = useNotification();

    const handleGenerate = async () => {
        setIsLoading(true);
        setFrames([]);
        setProgress('Decomposing user flow...');
        try {
            const { steps } = await decomposeUserFlow(flow);
            const newFrames: StoryboardFrame[] = [];
            for (let i = 0; i < steps.length; i++) {
                setProgress(`Generating wireframe for: "${steps[i]}" (${i +
                1}/${steps.length})`);
                const imageUrl = await generateImage(`A clean UI wireframe of: ${steps[i]}, user
                interface, UX design, simple, clean lines.`);
                newFrames.push({ description: steps[i], imageUrl });
                setFrames([...newFrames]);
            }
            addNotification('Storyboard generated!', 'success');
        } catch (err) {
            addNotification(err instanceof Error ? err.message : 'Generation failed',
            'error');
        } finally {
            setIsLoading(false);
            setProgress('');
        }
    };

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
            <header className="mb-6">
                <h1 className="text-3xl font-bold flex items-center"><PhotoIcon /><span
                className="ml-3">Storyboard Generator</span></h1>
                <p className="text-text-secondary mt-1">Create a sequence of UI mockups from a
                user flow description.</p>
            </header>
            <div className="flex flex-col gap-4">
                <div className="flex flex-col">
                    <label className="text-sm font-medium mb-2">User Flow Description</label>
                    <textarea value={flow} onChange={e => setFlow(e.target.value)} className="w-full
                    p-2 bg-surface border rounded text-sm"/>
                </div>
                <button onClick={handleGenerate} disabled={isLoading} className="btn-primary
                w-full max-w-sm mx-auto py-3">{isLoading ? <LoadingSpinner/> : 'Generate
                Storyboard'}</button>
            </div>
            <div className="flex-grow mt-6 bg-background border rounded-lg p-4 overflow-x-
```

```
            auto">
                {isLoading && <div className="text-center"><LoadingSpinner /><p className="mt-2
                text-sm text-text-secondary">{progress}</p></div>}
                <div className="flex gap-4 h-full">
                    {frames.map((frame, i) => (
                        <div key={i} className="flex-shrink-0 w-72 h-full bg-surface border rounded-md
                        flex flex-col p-2">
                            <div className="flex-grow bg-gray-200 rounded-sm flex items-center justify-
                            center">
                                <img src={frame.imageUrl} alt={frame.description} className="max-w-full max-h-
                                full object-contain"/>
                            </div>
                            <p className="text-xs text-center mt-2 p-1">{frame.description}</p>
                        </div>
                    ))}
                </div>
            </div>
        </div>
    );
};


// ===== UserPersonaGenerator_2.tsx =====

import React, { useState } from 'react';
import { generateUserPersona, generateImage } from
'../../services/aiService.ts';
import { DocumentTextIcon } from '../icons.tsx';
import { LoadingSpinner } from '../shared/index.tsx';
import { useNotification } from '../../contexts/NotificationContext.tsx';

interface Persona {
    name: string;
    photoUrl: string;
    demographics: string;
    goals: string[];
    frustrations: string[];
    techStack: string;
}

export const UserPersonaGenerator: React.FC = () => {
    const [description, setDescription] = useState('A busy project manager at a mid-
    sized tech company who needs to track team progress.');
    const [persona, setPersona] = useState<Persona | null>(null);
    const [isLoading, setIsLoading] = useState(false);
    const { addNotification } = useNotification();

    const handleGenerate = async () => {
        setIsLoading(true);
        setPersona(null);
        try {
            const personaData = await generateUserPersona(description);
            const photoUrl = await generateImage(`A photorealistic portrait of
            ${personaData.photoDescription}`);
            setPersona({ ...personaData, photoUrl });
            addNotification('Persona generated!', 'success');
        } catch (err) {
            addNotification(err instanceof Error ? err.message : 'Failed to generate
            persona', 'error');
        } finally {
            setIsLoading(false);
        }
```

```
    };

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
            <header className="mb-6">
                <h1 className="text-3xl font-bold flex items-center"><DocumentTextIcon /><span
                className="ml-3">User Persona Generator</span></h1>
                <p className="text-text-secondary mt-1">Create detailed user personas from a
                brief description of a target audience.</p>
            </header>
            <div className="flex-grow grid grid-cols-1 lg:grid-cols-2 gap-6 min-h-0">
                <div className="flex flex-col gap-4">
                    <div className="flex flex-col flex-1 min-h-0">
                        <label className="text-sm font-medium mb-2">Target Audience Description</label>
                        <textarea value={description} onChange={e => setDescription(e.target.value)}
                        className="flex-grow p-2 bg-surface border rounded"/>
                    </div>
                    <button onClick={handleGenerate} disabled={isLoading} className="btn-primary
                    w-full py-3">{isLoading ? <LoadingSpinner/> : 'Generate Persona'}</button>
                </div>
                <div className="bg-surface p-6 border rounded-lg overflow-y-auto">
                    {isLoading && <div className="flex justify-center items-center
                    h-full"><LoadingSpinner /></div>}
                    {persona && (
                        <div className="flex flex-col md:flex-row gap-6">
                            <div className="flex-shrink-0 w-32 h-32">
                                <img src={persona.photoUrl} alt={persona.name} className="w-full h-full rounded-
                                full object-cover shadow-md"/>
                            </div>
                            <div className="flex-grow">
                                <h2 className="text-2xl font-bold">{persona.name}</h2>
                                <p className="text-sm text-text-secondary">{persona.demographics}</p>
                                <div className="mt-4">
                                    <h3 className="font-semibold">Goals</h3>
                                    <ul className="list-disc list-inside text-sm">
                                        {persona.goals.map((g, i) => <li key={i}>{g}</li>)}
                                    </ul>
                                </div>
                                 <div className="mt-4">
                                    <h3 className="font-semibold">Frustrations</h3>
                                    <ul className="list-disc list-inside text-sm">
                                        {persona.frustrations.map((f, i) => <li key={i}>{f}</li>)}
                                    </ul>
                                </div>
                                <div className="mt-4">
                                    <h3 className="font-semibold">Tech Stack</h3>
                                    <p className="text-sm">{persona.techStack}</p>
                                </div>
                            </div>
                        </div>
                    )}
                </div>
            </div>
        </div>
    );
};


// ===== CompetitiveAnalysisBot_2.tsx =====

import React, { useState } from 'react';
import { analyzeCompetitorUrl } from '../../services/aiService.ts';
```

```
import { MagnifyingGlassIcon } from '../icons.tsx';
import { LoadingSpinner, MarkdownRenderer } from '../shared/index.tsx';
import { useNotification } from '../../contexts/NotificationContext.tsx';

export const CompetitiveAnalysisBot: React.FC = () => {
    const [url, setUrl] = useState('https://www.stripe.com');
    const [analysis, setAnalysis] = useState('');
    const [isLoading, setIsLoading] = useState(false);
    const { addNotification } = useNotification();

    const handleAnalyze = async () => {
        if (!url.trim()) {
            addNotification('Please enter a URL.', 'error');
            return;
        }
        setIsLoading(true);
        setAnalysis('');
        try {
            const result = await analyzeCompetitorUrl(url);
            setAnalysis(result);
        } catch (err) {
            addNotification(err instanceof Error ? err.message : 'Analysis failed.',
            'error');
        } finally {
            setIsLoading(false);
        }
    };

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
            <header className="mb-6">
                <h1 className="text-3xl font-bold flex items-center"><MagnifyingGlassIcon
                /><span className="ml-3">Competitive Analysis Bot</span></h1>
                <p className="text-text-secondary mt-1">Given a URL, AI will summarize the
                likely tech stack and features.</p>
            </header>
            <div className="flex items-center gap-2 mb-4">
                <input value={url} onChange={e => setUrl(e.target.value)}
                placeholder="https://competitor.com" className="flex-grow p-2 bg-surface border
                rounded"/>
                <button onClick={handleAnalyze} disabled={isLoading} className="btn-primary px-6
                py-2">{isLoading ? <LoadingSpinner/> : 'Analyze'}</button>
            </div>
            <p className="text-xs text-center text-yellow-600 mb-4 bg-yellow-400/10 p-2
            rounded-md">
                <strong>Note:</strong> This is a simulation. The AI uses its training data to
                infer information about the site and does not perform a live scrape.
            </p>
            <div className="flex-grow p-4 bg-background border rounded overflow-auto">
                {isLoading ? <div className="flex justify-center items-center
                h-full"><LoadingSpinner /></div> : (
                    analysis && <MarkdownRenderer content={analysis} />
                )}
            </div>
        </div>
    );
};


// ===== CodeDocumentationWriter_2.tsx =====

import React, { useState, useCallback } from 'react';
```

```
import { generateDocumentationForFiles } from '../../services/aiService.ts';
import { useGlobalState } from '../../contexts/GlobalStateContext.tsx';
import type { FileNode } from '../../types.ts';
import { DocumentTextIcon, FolderIcon } from '../icons.tsx';
import { LoadingSpinner, MarkdownRenderer } from '../shared/index.tsx';
import { useNotification } from '../../contexts/NotificationContext.tsx';
import { getDecryptedCredential } from '../../services/vaultService.ts';
import { initializeOctokit } from '../../services/authService.ts';
import { getFileContent } from '../../services/githubService.ts';

const FileTreeSelector: React.FC<{ node: FileNode, selectedPaths: Set<string>,
onToggle: (path: string, isFolder: boolean) => void }> = ({ node, selectedPaths,
onToggle }) => {
    const [isOpen, setIsOpen] = useState(true);
    const isSelected = selectedPaths.has(node.path);

    const handleToggle = () => {
        onToggle(node.path, node.type === 'folder');
    };

    if (node.type === 'file') {
        return (
            <div className="flex items-center space-x-2 pl-4 py-1">
                <input type="checkbox" checked={isSelected} onChange={handleToggle}
                className="w-4 h-4 rounded text-primary focus:ring-primary" />
                <DocumentTextIcon />
                <span>{node.name}</span>
            </div>
        );
    }

    return (
        <div>
            <div className="flex items-center space-x-2 py-1">
                <input type="checkbox" checked={isSelected} onChange={handleToggle}
                className="w-4 h-4 rounded text-primary focus:ring-primary"/>
                <button onClick={() => setIsOpen(!isOpen)} className={`transform transition-
                transform ${isOpen ? 'rotate-90' : ''}`}>■</button>
                <FolderIcon />
                <span className="font-semibold">{node.name}</span>
            </div>
            {isOpen && node.children && (
                <div className="pl-4 border-l border-border ml-3">
                    {node.children.map(child => <FileTreeSelector key={child.path} node={child}
                    selectedPaths={selectedPaths} onToggle={onToggle} />)}
                </div>
            )}
        </div>
    );
};


export const CodeDocumentationWriter: React.FC = () => {
    const { state } = useGlobalState();
    const { projectFiles, selectedRepo, user } = state;
    const [selectedPaths, setSelectedPaths] = useState(new Set<string>());
    const [documentation, setDocumentation] = useState('');
    const [isLoading, setIsLoading] = useState(false);
    const { addNotification } = useNotification();

    const getApiClient = useCallback(async () => {
        if (!user) {
```

```
        throw new Error("You must be logged in.");
    }
    const token = await getDecryptedCredential('github_pat');
    if (!token) {
        throw new Error("GitHub token not found. Please connect on the Connections
        page.");
    }
    return initializeOctokit(token);
}, [user]);

const findNodeByPath = (node: FileNode, path: string): FileNode | null => {
    if (node.path === path) return node;
    if (node.children) {
        for (const child of node.children) {
            const found = findNodeByPath(child, path);
            if (found) return found;
        }
    }
    return null;
}

const handleGenerate = async () => {
    if (selectedPaths.size === 0) {
        addNotification('Please select files to document.', 'error');
        return;
    }
    setIsLoading(true);
    setDocumentation('');
    try {
        if (!selectedRepo) {
            throw new Error('Please select a repository first.');
        }

        const pathsToFetch = Array.from(selectedPaths)
            .filter(path => path && projectFiles && findNodeByPath(projectFiles, path)?.type
            === 'file');

        if (pathsToFetch.length === 0) {
            addNotification('Please select specific files to document.', 'info');
            setIsLoading(false);
            return;
        }

        if (pathsToFetch.length > 10) {
            addNotification('For performance reasons, please select 10 files or fewer.',
            'info');
            setIsLoading(false);
            return;
        }

        const octokit = await getApiClient();

        const filePromises = pathsToFetch.map(path =>
            getFileContent(octokit, selectedRepo.owner, selectedRepo.repo, path)
                .then(content => ({ path, content }))
                .catch(err => {
                    console.error(`Failed to fetch ${path}`, err);
                    return { path, content: `// Error fetching content for this file.` };
                })
        );

        const filesToDocument = await Promise.all(filePromises);
```

```
        const result = await generateDocumentationForFiles(filesToDocument);
        setDocumentation(result);
        addNotification('Documentation generated!', 'success');
    } catch (err) {
        addNotification(err instanceof Error ? err.message : 'Failed to generate
        documentation.', 'error');
    } finally {
        setIsLoading(false);
    }
};

const getAllChildPaths = (node: FileNode): string[] => {
    let paths = node.type === 'file' ? [node.path] : [];
    if (node.children) {
        paths = paths.concat(...node.children.map(getAllChildPaths));
    }
    return paths;
};

const handleToggle = (path: string, isFolder: boolean) => {
    const newSelected = new Set(selectedPaths);
    const isSelected = newSelected.has(path);

    let pathsToToggle: string[] = [path];
    if (isFolder && projectFiles) {
        const folderNode = findNodeByPath(projectFiles, path);
        if(folderNode) pathsToToggle = [path, ...getAllChildPaths(folderNode)];
    }

    if (isSelected) {
        pathsToToggle.forEach(p => newSelected.delete(p));
    } else {
        pathsToToggle.forEach(p => newSelected.add(p));
    }
    setSelectedPaths(newSelected);
};

return (
    <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
        <header className="mb-6">
            <h1 className="text-3xl font-bold flex items-center"><DocumentTextIcon /><span
            className="ml-3">Code Documentation Writer</span></h1>
            <p className="text-text-secondary mt-1">Select files from your project to
            generate comprehensive documentation.</p>
        </header>
        <div className="flex-grow grid grid-cols-1 lg:grid-cols-2 gap-6 min-h-0">
            <div className="flex flex-col">
                 <label className="text-sm font-medium mb-2">Select Files</label>
                 <div className="flex-grow p-2 bg-surface border rounded overflow-auto">
                    {projectFiles ? <FileTreeSelector node={projectFiles}
                    selectedPaths={selectedPaths} onToggle={handleToggle} /> : <p>Load a project in
                    the Project Explorer first.</p>}
                 </div>
                 <button onClick={handleGenerate} disabled={isLoading || selectedPaths.size ===
                 0} className="btn-primary w-full mt-4 py-3">{isLoading ? <LoadingSpinner/> :
                 'Generate Documentation'}</button>
            </div>
            <div className="flex flex-col">
                <label className="text-sm font-medium mb-2">Generated Documentation</label>
                <div className="flex-grow p-4 bg-background border rounded overflow-auto">
                    {isLoading ? <div className="flex justify-center items-center
                    h-full"><LoadingSpinner /></div> : <MarkdownRenderer content={documentation} />}
```

```
                    </div>
                </div>
            </div>
        </div>
    );
};

// ===== DependencyUpdateExplainer_2.tsx =====

import React, { useState } from 'react';
import * as Diff from 'diff';
import { explainDependencyChanges } from '../../services/aiService.ts';
import { GitBranchIcon } from '../icons.tsx';
import { LoadingSpinner, MarkdownRenderer } from '../shared/index.tsx';
import { useNotification } from '../../contexts/NotificationContext.tsx';

const oldLock = `{
  "name": "my-app",
  "version": "1.0.0",
  "dependencies": {
    "react": { "version": "17.0.2" }
  }
}`;
const newLock = `{
  "name": "my-app",
  "version": "1.0.0",
  "dependencies": {
    "react": { "version": "18.2.0" },
    "lodash": { "version": "4.17.21" }
  }
}`;

export const DependencyUpdateExplainer: React.FC = () => {
    const [oldFile, setOldFile] = useState(oldLock);
    const [newFile, setNewFile] = useState(newLock);
    const [explanation, setExplanation] = useState('');
    const [isLoading, setIsLoading] = useState(false);
    const { addNotification } = useNotification();

    const handleExplain = async () => {
        setIsLoading(true);
        setExplanation('');
        try {
            const diff = Diff.createPatch('package-lock.json', oldFile, newFile);
            const result = await explainDependencyChanges(diff);
            setExplanation(result);
            addNotification('Explanation generated!', 'success');
        } catch (err) {
            addNotification(err instanceof Error ? err.message : 'Failed to generate
            explanation', 'error');
        } finally {
            setIsLoading(false);
        }
    };

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
            <header className="mb-6">
                <h1 className="text-3xl font-bold flex items-center"><GitBranchIcon /><span
                className="ml-3">Dependency Update Explainer</span></h1>
                <p className="text-text-secondary mt-1">Analyze a package-lock.json diff and
                explain the risks/benefits.</p>
```

```
            </header>
            <div className="flex-grow grid grid-cols-1 lg:grid-cols-2 gap-6 min-h-0">
                <div className="flex flex-col gap-4">
                    <div className="flex flex-col flex-1 min-h-0">
                        <label className="text-sm font-medium mb-2">Old package-lock.json</label>
                        <textarea value={oldFile} onChange={e => setOldFile(e.target.value)}
                        className="flex-grow p-2 bg-surface border rounded font-mono text-xs"/>
                    </div>
                     <div className="flex flex-col flex-1 min-h-0">
                        <label className="text-sm font-medium mb-2">New package-lock.json</label>
                        <textarea value={newFile} onChange={e => setNewFile(e.target.value)}
                        className="flex-grow p-2 bg-surface border rounded font-mono text-xs"/>
                    </div>
                </div>
                <div className="flex flex-col">
                    <button onClick={handleExplain} disabled={isLoading} className="btn-primary
                    w-full py-3 mb-4">{isLoading ? <LoadingSpinner/> : 'Explain Changes'}</button>
                    <div className="flex-grow p-4 bg-background border rounded overflow-auto">
                        {isLoading && <div className="flex justify-center items-center
                        h-full"><LoadingSpinner /></div>}
                        {explanation && <MarkdownRenderer content={explanation} />}
                    </div>
                </div>
            </div>
        </div>
    );
};


// ===== AiVideoGenerator_2.tsx =====

import React, { useState, useEffect, useRef } from 'react';
import { decomposeUserFlow, generateImage } from '../../services/aiService.ts';
import { VideoCameraIcon } from '../icons.tsx';
import { LoadingSpinner } from '../shared/index.tsx';
import { useNotification } from '../../contexts/NotificationContext.tsx';

export const AiVideoGenerator: React.FC = () => {
    const [prompt, setPrompt] = useState('A cinematic shot of a robot skateboarding
    through a neon-lit city at night.');
    const [images, setImages] = useState<string[]>([]);
    const [currentImageIndex, setCurrentImageIndex] = useState(0);
    const [isLoading, setIsLoading] = useState(false);
    const [isPlaying, setIsPlaying] = useState(false);
    const intervalRef = useRef<number | null>(null);
    const { addNotification } = useNotification();

    const handleGenerate = async () => {
        setIsLoading(true);
        setImages([]);
        try {
            const { steps } = await decomposeUserFlow(`Create a short video clip based on
            this prompt: "${prompt}". Break it down into 4 keyframes.`);
            for (let i = 0; i < steps.length; i++) {
                const imageUrl = await generateImage(`A cinematic frame from a video of:
                ${steps[i]}.`);
                setImages(prev => [...prev, imageUrl]);
            }
            addNotification('Video frames generated!', 'success');
        } catch (err) {
            addNotification(err instanceof Error ? err.message : 'Generation failed',
            'error');
```

```
        } finally {
            setIsLoading(false);
        }
    };

    useEffect(() => {
        if (isPlaying) {
            intervalRef.current = window.setInterval(() => {
                setCurrentImageIndex(prev => (prev + 1) % images.length);
            }, 200); // 5 fps
        } else {
            if (intervalRef.current) clearInterval(intervalRef.current);
        }
        return () => {
            if (intervalRef.current) clearInterval(intervalRef.current);
        };
    }, [isPlaying, images.length]);

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
            <header className="mb-6">
                <h1 className="text-3xl font-bold flex items-center"><VideoCameraIcon /><span
                className="ml-3">AI Video Generator (Simulated)</span></h1>
                <p className="text-text-secondary mt-1">Generate a sequence of images from a
                prompt to simulate video creation.</p>
            </header>
            <div className="flex-grow grid grid-cols-1 lg:grid-cols-2 gap-6 min-h-0">
                <div className="flex flex-col gap-4">
                    <div className="flex flex-col flex-1 min-h-0">
                        <label className="text-sm font-medium mb-2">Prompt</label>
                        <textarea value={prompt} onChange={e => setPrompt(e.target.value)}
                        className="flex-grow p-2 bg-surface border rounded"/>
                    </div>
                    <button onClick={handleGenerate} disabled={isLoading} className="btn-primary
                    w-full py-3">{isLoading ? <LoadingSpinner/> : 'Generate Video Frames'}</button>
                    <p className="text-xs text-center text-yellow-600 bg-yellow-400/10 p-2 rounded-
                    md">
                        <strong>Note:</strong> This is a simulation. It generates a sequence of images
                        and plays them as a slideshow to mimic video generation, as client-side video
                        generation APIs are not yet standard.
                    </p>
                </div>
                <div className="flex flex-col">
                    <label className="text-sm font-medium mb-2">Generated Video</label>
                    <div className="flex-grow bg-background border rounded overflow-hidden
                    relative">
                        {isLoading && <div className="flex justify-center items-center
                        h-full"><LoadingSpinner /></div>}
                        {images.length > 0 && <img src={images[currentImageIndex]} alt="Generated frame"
                        className="w-full h-full object-contain"/>}
                        {images.length > 0 && !isLoading && (
                            <button onClick={() => setIsPlaying(!isPlaying)} className="absolute bottom-4
                            left-1/2 -translate-x-1/2 bg-black/50 text-white px-4 py-2 rounded-full">
                                {isPlaying ? 'Pause' : 'Play'}
                            </button>
                        )}
                    </div>
                </div>
            </div>
        </div>
    );
};
```

```
// ===== CloudCostEstimator_2.tsx =====

import React, { useState } from 'react';
import { estimateCloudCost } from '../../services/aiService.ts';
import { GcpIcon } from '../icons.tsx';
import { LoadingSpinner, MarkdownRenderer } from '../shared/index.tsx';
import { useNotification } from '../../contexts/NotificationContext.tsx';

export const CloudCostEstimator: React.FC = () => {
    const [description, setDescription] = useState('A web app with 2 vCPUs, a 50GB
    SQL database, and a load balancer in us-central1');
    const [estimate, setEstimate] = useState('');
    const [isLoading, setIsLoading] = useState(false);
    const { addNotification } = useNotification();

    const handleEstimate = async () => {
        setIsLoading(true);
        setEstimate('');
        try {
            const result = await estimateCloudCost(description);
            setEstimate(result);
            addNotification('Estimate generated!', 'success');
        } catch (err) {
            addNotification(err instanceof Error ? err.message : 'Failed to generate
            estimate', 'error');
        } finally {
            setIsLoading(false);
        }
    };

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
            <header className="mb-6">
                <h1 className="text-3xl font-bold flex items-center"><GcpIcon /><span
                className="ml-3">Cloud Cost Estimator</span></h1>
                <p className="text-text-secondary mt-1">Estimate GCP/AWS costs based on a
                description of services.</p>
            </header>
            <div className="flex-grow flex flex-col gap-4 min-h-0">
                <div className="flex flex-col flex-1 min-h-0">
                    <label className="text-sm font-medium mb-2">Architecture Description</label>
                    <textarea value={description} onChange={e => setDescription(e.target.value)}
                    className="flex-grow p-2 bg-surface border rounded"/>
                </div>
                <button onClick={handleEstimate} disabled={isLoading} className="btn-primary
                w-full max-w-sm mx-auto py-3">{isLoading ? <LoadingSpinner/> : 'Estimate Monthly
                Cost'}</button>
                <div className="flex flex-col flex-grow min-h-0 mt-4">
                    <label className="text-sm font-medium mb-2">AI-Generated Estimate</label>
                    <div className="flex-grow p-4 bg-background border rounded overflow-auto">
                        {isLoading ? <div className="flex justify-center items-center
                        h-full"><LoadingSpinner /></div> : (
                            estimate && (
                                <>
                                    <MarkdownRenderer content={estimate} />
                                    <p className="text-xs text-yellow-600 mt-4"><strong>Disclaimer:</strong> This is
                                    a rough, non-binding estimate based on public pricing data and should not be
                                    used for official budgeting.</p>
                                </>
                            )
                        )}
                    </div>
```

```
                </div>
            </div>
        </div>
    );
};


// ===== SmartLogger_2.tsx =====

import React, { useState } from 'react';
import * as Diff from 'diff';
import { insertSmartLogging } from '../../services/aiService.ts';
import { TerminalIcon } from '../icons.tsx';
import { LoadingSpinner } from '../shared/index.tsx';

const exampleCode = `function processData(data) {
  if (!data || data.length === 0) {
    return [];
  }
  const results = data.filter(item => item.value > 10);
  return results.map(item => ({ ...item, processed: true }));
}`;

const DiffViewer: React.FC<{ oldCode: string, newCode: string }> = React.memo(({
oldCode, newCode }) => {
    const diff = Diff.diffLines(oldCode, newCode);
    return (
        <pre className="whitespace-pre-wrap font-mono text-xs">
            {diff.map((part, index) => {
                const color = part.added ? 'bg-green-500/20 text-green-800 dark:text-green-300'
                : part.removed ? 'bg-red-500/20' : 'text-text-secondary';
                return <div key={index} className={color}>{part.value}</div>;
            })}
        </pre>
    );
});

export const SmartLogger: React.FC = () => {
    const [code, setCode] = useState(exampleCode);
    const [loggedCode, setLoggedCode] = useState('');
    const [isLoading, setIsLoading] = useState(false);

    const handleGenerate = async () => {
        setIsLoading(true);
        setLoggedCode('');
        try {
            const result = await insertSmartLogging(code);
            setLoggedCode(result.replace(/^```(?:\w+\n)?/, '').replace(/```$/, ''));
        } catch (err) {
            console.error(err);
        } finally {
            setIsLoading(false);
        }
    };

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
            <header className="mb-6">
                <h1 className="text-3xl font-bold flex items-center"><TerminalIcon /><span
                className="ml-3">Smart Logger</span></h1>
                <p className="text-text-secondary mt-1">Insert intelligent logging statements
                into code for easier debugging.</p>
```

```
            </header>
            <div className="flex-grow grid grid-cols-1 lg:grid-cols-2 gap-6 min-h-0">
                <div className="flex flex-col">
                    <label className="text-sm font-medium mb-2">Original Code</label>
                    <textarea value={code} onChange={e => setCode(e.target.value)} className="flex-
                    grow p-2 bg-surface border rounded font-mono text-xs"/>
                    <button onClick={handleGenerate} disabled={isLoading} className="btn-primary
                    w-full mt-4 py-3">{isLoading ? <LoadingSpinner/> : 'Add Logs'}</button>
                </div>
                 <div className="flex flex-col">
                    <label className="text-sm font-medium mb-2">Code with Logging</label>
                    <div className="flex-grow p-2 bg-background border rounded overflow-auto">
                        {isLoading ? <div className="flex justify-center items-center
                        h-full"><LoadingSpinner /></div> : <DiffViewer oldCode={code}
                        newCode={loggedCode} />}
                    </div>
                </div>
            </div>
        </div>
    );
};


// ===== AccessibilityAnnotation_2.tsx =====

import React, { useState } from 'react';
import * as Diff from 'diff';
import { addAriaAttributes } from '../../services/aiService.ts';
import { EyeIcon } from '../icons.tsx';
import { LoadingSpinner } from '../shared/index.tsx';

const exampleHtml = `<div class="menu">
  <div>Menu Item 1</div>
  <div class="active">Menu Item 2</div>
  <div>Menu Item 3</div>
</div>`;

const DiffViewer: React.FC<{ oldCode: string, newCode: string }> = React.memo(({
oldCode, newCode }) => {
    const diff = Diff.diffLines(oldCode, newCode);
    return (
        <pre className="whitespace-pre-wrap font-mono text-xs">
            {diff.map((part, index) => {
                const color = part.added ? 'bg-green-500/20 text-green-800 dark:text-green-300'
                : part.removed ? 'bg-red-500/20' : 'text-text-secondary';
                return <div key={index} className={color}>{part.value}</div>;
            })}
        </pre>
    );
});

export const AccessibilityAnnotation: React.FC = () => {
    const [html, setHtml] = useState(exampleHtml);
    const [annotatedHtml, setAnnotatedHtml] = useState('');
    const [isLoading, setIsLoading] = useState(false);

    const handleAnnotate = async () => {
        setIsLoading(true);
        setAnnotatedHtml('');
        try {
            const result = await addAriaAttributes(html);
            setAnnotatedHtml(result.replace(/^```(?:\w+\n)?/, '').replace(/```$/, ''));
```

```
        } catch (err) {
            console.error(err);
        } finally {
            setIsLoading(false);
        }
    };

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
            <header className="mb-6">
                <h1 className="text-3xl font-bold flex items-center"><EyeIcon /><span
                className="ml-3">Accessibility Annotation</span></h1>
                <p className="text-text-secondary mt-1">Add ARIA attributes to HTML to improve
                accessibility.</p>
            </header>
            <div className="flex-grow grid grid-cols-1 lg:grid-cols-2 gap-6 min-h-0">
                <div className="flex flex-col">
                    <label className="text-sm font-medium mb-2">Original HTML</label>
                    <textarea value={html} onChange={e => setHtml(e.target.value)} className="flex-
                    grow p-2 bg-surface border rounded font-mono text-xs"/>
                    <button onClick={handleAnnotate} disabled={isLoading} className="btn-primary
                    w-full mt-4 py-3">{isLoading ? <LoadingSpinner/> : 'Annotate with
                    ARIA'}</button>
                </div>
                 <div className="flex flex-col">
                    <label className="text-sm font-medium mb-2">Annotated HTML</label>
                    <div className="flex-grow p-2 bg-background border rounded overflow-auto">
                        {isLoading ? <div className="flex justify-center items-center
                        h-full"><LoadingSpinner /></div> : <DiffViewer oldCode={html}
                        newCode={annotatedHtml} />}
                    </div>
                </div>
            </div>
        </div>
    );
};


// ===== WordPressPluginGenerator.tsx =====

import React, { useState } from 'react';
import JSZip from 'jszip';
import { generateWordPressPlugin } from '../../services/aiService.ts';
import type { GeneratedFile } from '../../types.ts';
import { WordPressIcon } from '../icons.tsx';
import { LoadingSpinner, MarkdownRenderer } from '../shared/index.tsx';
import { useNotification } from '../../contexts/NotificationContext.tsx';
import { downloadFile } from '../../services/fileUtils.ts';

const examplePrompt = 'a simple shortcode that displays "Hello World"';

export const WordPressPluginGenerator: React.FC = () => {
    const [prompt, setPrompt] = useState(examplePrompt);
    const [files, setFiles] = useState<GeneratedFile[]>([]);
    const [activeFile, setActiveFile] = useState<GeneratedFile | null>(null);
    const [isLoading, setIsLoading] = useState(false);
    const { addNotification } = useNotification();

    const handleGenerate = async () => {
        if (!prompt.trim()) {
            addNotification('Please provide a plugin description.', 'error');
            return;
```

```
    }
    setIsLoading(true);
    setFiles([]);
    setActiveFile(null);
    try {
        const result = await generateWordPressPlugin(prompt);
        setFiles(result);
        if (result.length > 0) {
            setActiveFile(result[0]);
        }
        addNotification('WordPress plugin generated!', 'success');
    } catch (err) {
        addNotification(err instanceof Error ? err.message : 'Failed to generate
        plugin', 'error');
    } finally {
        setIsLoading(false);
    }
};

const handleDownloadZip = async () => {
    if (files.length === 0) return;
    const zip = new JSZip();
    files.forEach(file => {
        zip.file(file.filePath, file.content);
    });

    const zipBlob = await zip.generateAsync({ type: 'blob' });
    const link = document.createElement('a');
    link.href = URL.createObjectURL(zipBlob);
    const pluginName = files[0]?.filePath.split('/')[0] || 'my-plugin';
    link.download = `${pluginName}.zip`;
    document.body.appendChild(link);
    link.click();
    document.body.removeChild(link);
};

return (
    <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
        <header className="mb-6">
            <h1 className="text-3xl font-bold flex items-center"><WordPressIcon /><span
            className="ml-3">WordPress Plugin Generator</span></h1>
            <p className="text-text-secondary mt-1">Generate a functional WordPress plugin
            from a text description.</p>
        </header>
        <div className="flex-grow grid grid-cols-1 lg:grid-cols-2 gap-6 min-h-0">
            <div className="flex flex-col gap-4">
                <div className="flex flex-col flex-1 min-h-0">
                    <label className="text-sm font-medium mb-2">Plugin Description</label>
                    <textarea value={prompt} onChange={e => setPrompt(e.target.value)}
                    className="flex-grow p-2 bg-surface border rounded"/>
                </div>
                <button onClick={handleGenerate} disabled={isLoading} className="btn-primary
                w-full py-3">{isLoading ? <LoadingSpinner/> : 'Generate Plugin'}</button>
            </div>
            <div className="flex flex-col min-h-0">
                <div className="flex justify-between items-center mb-2">
                    <label className="text-sm font-medium">Generated Files</label>
                    {files.length > 0 && <button onClick={handleDownloadZip} className="btn-primary
                    px-3 py-1 text-sm">Download as ZIP</button>}
                </div>
                <div className="flex-grow grid grid-cols-3 gap-2 min-h-0 bg-background border
                rounded-lg p-2">
```

```
                        <div className="col-span-1 overflow-y-auto">
                            {isLoading ? <div className="flex justify-center items-center
                            h-full"><LoadingSpinner /></div> : (
                                files.map(file => (
                                    <div key={file.filePath} onClick={() => setActiveFile(file)} className={`p-2
                                    text-sm rounded cursor-pointer ${activeFile?.filePath === file.filePath ? 'bg-
                                    primary/10 text-primary' : ''}`}>{file.filePath}</div>
                                ))
                            )}
                        </div>
                        <div className="col-span-2 bg-surface rounded overflow-y-auto">
                            {activeFile && <MarkdownRenderer content={'```php\n' + activeFile.content +
                            '\n```'} />}
                        </div>
                    </div>
                </div>
            </div>
        </div>
    );
};


// ===== FeatureForge.tsx =====

import React, { useState, useEffect, useCallback } from 'react';
import { generateAppFeatureComponent } from '../../services/aiService.ts';
import { getAllCustomFeatures, saveCustomFeature, deleteCustomFeature } from
'../../services/dbService.ts';
import type { CustomFeature } from '../../types.ts';
import { CpuChipIcon, PlusIcon, TrashIcon } from '../icons.tsx';
import { LoadingSpinner, MarkdownRenderer } from '../shared/index.tsx';
import { useNotification } from '../../contexts/NotificationContext.tsx';
import { ALL_FEATURES } from './index.ts';

const ICON_MAP: Record<string, React.FC> = ALL_FEATURES.reduce((acc, feature) =>
{
    const iconType = (feature.icon as React.ReactElement)?.type;
    if (typeof iconType === 'function' && iconType.name) {
      const iconName = iconType.name;
      acc[iconName] = iconType as React.FC;
    }
    return acc;
  }, {} as Record<string, React.FC>);


export const FeatureForge: React.FC = () => {
    const [customFeatures, setCustomFeatures] = useState<CustomFeature[]>([]);
    const [isLoading, setIsLoading] = useState<'list' | 'generate' | false>(false);
    const [isGenerating, setIsGenerating] = useState(false);
    const [prompt, setPrompt] = useState('A tool to convert JSON to YAML');
    const [generatedFeature, setGeneratedFeature] = useState<Omit<CustomFeature,
    'id'> | null>(null);
    const { addNotification } = useNotification();

    const fetchFeatures = useCallback(async () => {
        setIsLoading('list');
        const features = await getAllCustomFeatures();
        setCustomFeatures(features);
        setIsLoading(false);
    }, []);

    useEffect(() => {
```

```
    fetchFeatures();
}, [fetchFeatures]);

const handleGenerate = async () => {
    if (!prompt.trim()) return;
    setIsGenerating(true);
    setGeneratedFeature(null);
    try {
        const result = await generateAppFeatureComponent(prompt);
        setGeneratedFeature(result);
        addNotification('Feature code generated! Review and save.', 'info');
    } catch (err) {
        addNotification(err instanceof Error ? err.message : 'Failed to generate
        feature', 'error');
    } finally {
        setIsGenerating(false);
    }
};

const handleSave = async () => {
    if (!generatedFeature) return;
    const newFeature: CustomFeature = {
        ...generatedFeature,
        id: `custom-${Date.now()}`
    };
    await saveCustomFeature(newFeature);
    setGeneratedFeature(null);
    setPrompt('');
    fetchFeatures();
    addNotification(`Feature "${newFeature.name}" saved! It's now available on your
    desktop.`, 'success');
};

const handleDelete = async (id: string) => {
    if (window.confirm("Are you sure you want to delete this feature?")) {
        await deleteCustomFeature(id);
        fetchFeatures();
        addNotification('Feature deleted.', 'info');
    }
};

const IconComponent = ({ name }: { name: string }) => {
    const Comp = ICON_MAP[name];
    return Comp ? <Comp /> : <CpuChipIcon />;
};

return (
    <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
        <header className="mb-6">
            <h1 className="text-3xl font-bold flex items-center"><CpuChipIcon /><span
            className="ml-3">Feature Forge</span></h1>
            <p className="text-text-secondary mt-1">Use AI to create new tools and add them
            to your desktop.</p>
        </header>
        <div className="flex-grow grid grid-cols-1 lg:grid-cols-3 gap-6 min-h-0">
            {/* Left: Generator */}
            <div className="lg:col-span-1 flex flex-col gap-4 bg-surface p-4 border border-
            border rounded-lg">
                <h3 className="text-lg font-bold">1. Create a New Feature</h3>
                <div className="flex flex-col flex-grow">
                    <label className="text-sm">Describe the tool you want to build</label>
                    <textarea value={prompt} onChange={e => setPrompt(e.target.value)}
```

```
                        className="w-full mt-1 p-2 bg-background border border-border rounded"
                        rows={4}/>
                    </div>
                    <button onClick={handleGenerate} disabled={isGenerating} className="btn-primary
                    py-2 flex items-center justify-center gap-2">{isGenerating ? <LoadingSpinner/> :
                    'Generate Code'}</button>
                    {generatedFeature && (
                        <div className="p-4 border border-dashed rounded-lg space-y-2 animate-pop-in">
                            <h4 className="font-bold">Review Generated Feature</h4>
                            <p><strong>Name:</strong> {generatedFeature.name}</p>
                            <p><strong>Description:</strong> {generatedFeature.description}</p>
                            <div className="max-h-48 overflow-y-auto bg-background p-2
                            rounded"><MarkdownRenderer content={'```tsx\n' + generatedFeature.code + '```'}
                            /></div>
                            <button onClick={handleSave} className="w-full py-2 bg-green-600 text-white
                            font-bold rounded-md">Save Feature</button>
                        </div>
                    )}
                </div>

                {/* Right: Existing Custom Features */}
                <div className="lg:col-span-2 flex flex-col gap-4 min-h-0">
                   <div className="bg-surface p-4 border border-border rounded-lg flex-grow flex
                   flex-col min-h-0">
                        <h3 className="text-lg font-bold mb-2">2. Your Custom Features</h3>
                        <div className="flex-grow overflow-y-auto pr-2">
                            {isLoading === 'list' && <LoadingSpinner />}
                            {customFeatures.length === 0 && !isLoading && <p className="text-text-secondary
                            text-center py-8">You haven't created any features yet.</p>}
                            <div className="space-y-3">
                                {customFeatures.map(feature => (
                                    <div key={feature.id} className="group bg-background p-3 rounded-lg border
                                    border-border flex items-center justify-between">
                                        <div className="flex items-center gap-3">
                                            <div className="text-primary"><IconComponent name={feature.icon} /></div>
                                            <div>
                                                <h4 className="font-semibold">{feature.name}</h4>
                                                <p className="text-xs text-text-secondary">{feature.description}</p>
                                            </div>
                                        </div>
                                        <button onClick={() => handleDelete(feature.id)} className="opacity-0 group-
                                        hover:opacity-100 text-red-500 hover:text-red-700 p-1"><TrashIcon /></button>
                                    </div>
                                ))}
                            </div>
                        </div>
                   </div>
                </div>
            </div>
        </div>
    );
};

// ===== CustomFeatureRunner.tsx =====

import React from 'react';
import type { CustomFeature } from '../../types.ts';

const iframeContent = (code: string) => `
<!DOCTYPE html>
<html>
<head>
```

```
  <meta charset="UTF-8" />
  <script type="importmap">
    {
      "imports": {
        "react": "https://esm.sh/react@18.3.1",
        "react-dom/client": "https://esm.sh/react-dom@18.3.1/client"
      }
    }
  </script>
  <script src="https://cdn.tailwindcss.com"></script>
  <script src="https://unpkg.com/@babel/standalone/babel.min.js"></script>
</head>
<body>
  <div id="root" class="p-4"></div>
  <script type="text/babel">
    import React from 'react';
    import ReactDOM from 'react-dom/client';

    try {
      const App = (function() {
        const module = { exports: {} };
        (function(module, exports, React) {
          // --- Start of AI Generated Code ---
          ${code}
          // --- End of AI Generated Code ---
        })(module, module.exports, React);
        return module.exports.default;
      })();

      const root = ReactDOM.createRoot(document.getElementById('root'));
      root.render(<App />);
    } catch (e) {
      const root = ReactDOM.createRoot(document.getElementById('root'));
      root.render(
        <div style={{color: 'red', fontFamily: 'monospace'}}>
          <h3>Error Rendering Component</h3>
          <pre>{e.stack}</pre>
        </div>
      );
      console.error(e);
    }
  </script>
</body>
</html>
`;


export const CustomFeatureRunner: React.FC<{ feature: CustomFeature }> = ({
feature }) => {
    return (
        <div className="h-full w-full bg-background">
            <iframe
                srcDoc={iframeContent(feature.code)}
                title={`Preview: ${feature.name}`}
                sandbox="allow-scripts"
                className="w-full h-full border-0"
            />
        </div>
    );
};
// ===== DomTreeAnalyzer.tsx =====
import React, { useState } from 'react';
```

```
import { ChartBarIcon } from '../icons.tsx';
import { analyzeUrlDom } from '../../services/index.ts';
import { LoadingSpinner } from '../shared/index.tsx';

export const DomTreeAnalyzer: React.FC = () => {
    const [url, setUrl] = useState('https://react.dev');
    const [results, setResults] = useState<{ nodeCount: number, maxDepth: number,
    maxChildren: number } | null>(null);
    const [isLoading, setIsLoading] = useState(false);
    const [error, setError] = useState('');

    const handleAnalyze = async () => {
        if (!url.trim()) {
            setError('Please enter a URL.');
            return;
        }
        setIsLoading(true);
        setError('');
        setResults(null);
        try {
            const data = await analyzeUrlDom(url);
            setResults({
                nodeCount: data.nodeCount,
                maxDepth: data.maxDepth,
                maxChildren: data.maxChildren,
            });
        } catch (e) {
            setError(e instanceof Error ? e.message : 'Failed to analyze URL.');
        } finally {
            setIsLoading(false);
        }
    };

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
            <header className="mb-6">
                <h1 className="text-3xl font-bold flex items-center">
                    <ChartBarIcon />
                    <span className="ml-3">DOM Tree Analyzer</span>
                </h1>
                <p className="text-text-secondary mt-1">Get an AI-powered estimation of a URL's
                DOM complexity.</p>
            </header>
            <div className="flex-grow flex flex-col items-center justify-center gap-4">
                <div className="w-full max-w-lg">
                    <label className="text-sm font-medium mb-2">Target URL</label>
                    <div className="flex gap-2">
                        <input
                            type="text"
                            value={url}
                            onChange={e => setUrl(e.target.value)}
                            className="flex-grow p-2 bg-surface border rounded-md"
                            placeholder="https://example.com"
                        />
                        <button onClick={handleAnalyze} disabled={isLoading} className="btn-primary px-6
                        py-2">{isLoading ? <LoadingSpinner /> : 'Analyze'}</button>
                    </div>
                </div>
                {error && <p className="text-red-500 text-sm">{error}</p>}
                {results && (
                    <div className="mt-6 w-full max-w-lg bg-surface p-6 rounded-lg border border-
                    border animate-pop-in">
```

```
                        <h3 className="text-lg font-bold mb-4">Analysis Results for <span
                        className="text-primary">{url}</span></h3>
                        <div className="grid grid-cols-3 gap-4 text-center">
                            <div>
                                <p className="text-3xl font-bold text-primary">{results.nodeCount}</p>
                                <p className="text-sm text-text-secondary">Total Nodes</p>
                            </div>
                            <div>
                                <p className="text-3xl font-bold text-primary">{results.maxDepth}</p>
                                <p className="text-sm text-text-secondary">Max Depth</p>
                            </div>
                            <div>
                                <p className="text-3xl font-bold text-primary">{results.maxChildren}</p>
                                <p className="text-sm text-text-secondary">Max Children</p>
                            </div>
                        </div>
                    </div>
                )}
            </div>
        </div>
    );
};


// ===== MemoryLeakDetector.tsx =====

import React, { useState } from 'react';
import { BeakerIcon } from '../icons.tsx';
import { LoadingSpinner, MarkdownRenderer } from '../shared/index.tsx';
import { analyzeForMemoryLeaksStream } from '../../services/index.ts';

const exampleCode = `import { useEffect } from 'react';

function Ticker() {
  useEffect(() => {
    const timer = setInterval(() => {
      console.log('tick');
    }, 1000);

    // Missing cleanup function for the interval
  }, []);

  return <div>Ticker running...</div>
}`;

export const MemoryLeakDetector: React.FC = () => {
    const [code, setCode] = useState(exampleCode);
    const [analysis, setAnalysis] = useState('');
    const [isLoading, setIsLoading] = useState(false);

    const handleAnalyze = async () => {
        setIsLoading(true);
        setAnalysis('');
        try {
            const stream = analyzeForMemoryLeaksStream(code);
            let fullResponse = '';
            for await (const chunk of stream) {
                fullResponse += chunk;
                setAnalysis(fullResponse);
            }
        } catch (e) {
            setAnalysis(`Error: ${e instanceof Error ? e.message : 'An unknown error
```

```
            occurred'}`);
        } finally {
            setIsLoading(false);
        }
    };

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
            <header className="mb-6">
                <h1 className="text-3xl font-bold flex items-center">
                    <BeakerIcon />
                    <span className="ml-3">Memory Leak Analyzer</span>
                </h1>
                <p className="text-text-secondary mt-1">Use AI to analyze code for common memory
                leak patterns.</p>
            </header>
            <div className="flex-grow grid grid-cols-1 lg:grid-cols-2 gap-6 min-h-0">
                <div className="flex flex-col">
                    <label className="text-sm font-medium mb-2">Code to Analyze</label>
                    <textarea value={code} onChange={e => setCode(e.target.value)} className="flex-
                    grow p-2 bg-surface border rounded font-mono text-xs"/>
                    <button onClick={handleAnalyze} disabled={isLoading} className="btn-primary
                    w-full mt-4 py-3">{isLoading ? <LoadingSpinner/> : 'Analyze for Leaks'}</button>
                </div>
                <div className="flex flex-col">
                    <label className="text-sm font-medium mb-2">AI Analysis</label>
                    <div className="flex-grow p-4 bg-background border rounded overflow-auto">
                        {isLoading && !analysis && <div className="flex justify-center items-center
                        h-full"><LoadingSpinner /></div>}
                        {analysis && <MarkdownRenderer content={analysis} />}
                    </div>
                </div>
            </div>
        </div>
    );
};

// ===== GraphQLQueryProfiler.tsx =====

import React, { useState } from 'react';
import { MagnifyingGlassIcon } from '../icons.tsx';
import { analyzeGraphqlQueryStream } from '../../services/index.ts';
import { LoadingSpinner, MarkdownRenderer } from '../shared/index.tsx';

const exampleQuery = `query GetUsers {
  users(first: 10) {
    id
    name
    posts { # This could be an N+1 problem
      title
    }
  }
}`;

export const GraphQLQueryProfiler: React.FC = () => {
    const [query, setQuery] = useState(exampleQuery);
    const [analysis, setAnalysis] = useState('');
    const [isLoading, setIsLoading] = useState(false);

    const handleProfile = async () => {
        setIsLoading(true);
        setAnalysis('');
```

```
        try {
            const stream = analyzeGraphqlQueryStream(query);
            let fullResponse = '';
            for await (const chunk of stream) {
                fullResponse += chunk;
                setAnalysis(fullResponse);
            }
        } catch (e) {
            setAnalysis(`Error: ${e instanceof Error ? e.message : 'Unknown error'}`);
        } finally {
            setIsLoading(false);
        }
    };

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
            <header className="mb-6">
                <h1 className="text-3xl font-bold flex items-center">
                    <MagnifyingGlassIcon />
                    <span className="ml-3">GraphQL Query Analyzer</span>
                </h1>
                <p className="text-text-secondary mt-1">Use AI to analyze GraphQL queries for
                performance bottlenecks.</p>
            </header>
            <div className="flex-grow grid grid-cols-1 lg:grid-cols-2 gap-6 min-h-0">
                <div className="flex flex-col">
                    <label className="text-sm font-medium mb-2">GraphQL Query</label>
                    <textarea value={query} onChange={e => setQuery(e.target.value)}
                    className="flex-grow p-2 bg-surface border rounded font-mono text-xs"/>
                    <button onClick={handleProfile} disabled={isLoading} className="btn-primary
                    w-full mt-4 py-3">{isLoading ? <LoadingSpinner /> : 'Analyze Query'}</button>
                </div>
                <div className="flex flex-col">
                    <label className="text-sm font-medium mb-2">AI Performance Analysis</label>
                    <div className="flex-grow p-4 bg-background border rounded overflow-auto">
                        {isLoading && !analysis && <div className="flex justify-center items-center
                        h-full"><LoadingSpinner/></div>}
                        {analysis && <MarkdownRenderer content={analysis} />}
                    </div>
                </div>
            </div>
        </div>
    );
};

// ===== ComponentRenderTracer.tsx =====

import React, { useState } from 'react';
import { EyeIcon } from '../icons.tsx';
import { analyzeReactComponentRendersStream } from '../../services/index.ts';
import { LoadingSpinner, MarkdownRenderer } from '../shared/index.tsx';

const exampleCode = `import React, { useState } from 'react';

const InefficientComponent = ({ complexObject }) => {
  return <div>{complexObject.name}</div>;
}

const Parent = () => {
  const [count, setCount] = useState(0);

  // This object is recreated on every render,
```

```
  // causing InefficientComponent to re-render even
  // if its data hasn't changed.
  const myData = { name: 'Constant Data' };

  return (
    <div>
      <button onClick={() => setCount(c => c + 1)}>
        Increment: {count}
      </button>
      <InefficientComponent complexObject={myData} />
    </div>
  );
}`;

export const ComponentRenderTracer: React.FC = () => {
    const [code, setCode] = useState(exampleCode);
    const [analysis, setAnalysis] = useState('');
    const [isLoading, setIsLoading] = useState(false);

    const handleTrace = async () => {
        setIsLoading(true);
        setAnalysis('');
        try {
            const stream = analyzeReactComponentRendersStream(code);
            let fullResponse = '';
            for await (const chunk of stream) {
                fullResponse += chunk;
                setAnalysis(fullResponse);
            }
        } catch (e) {
            setAnalysis(`Error: ${e instanceof Error ? e.message : 'Unknown error'}`);
        } finally {
            setIsLoading(false);
        }
    };

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
            <header className="mb-6">
                <h1 className="text-3xl font-bold flex items-center">
                    <EyeIcon />
                    <span className="ml-3">Component Re-Render Analyzer</span>
                </h1>
                <p className="text-text-secondary mt-1">Use AI to analyze React code for causes
                of unnecessary re-renders.</p>
            </header>
            <div className="flex-grow grid grid-cols-1 lg:grid-cols-2 gap-6 min-h-0">
                <div className="flex flex-col">
                    <label className="text-sm font-medium mb-2">Component Code</label>
                    <textarea value={code} onChange={e => setCode(e.target.value)} className="flex-
                    grow p-2 bg-surface border rounded font-mono text-xs"/>
                    <button onClick={handleTrace} disabled={isLoading} className="btn-primary w-full
                    mt-4 py-3">{isLoading ? <LoadingSpinner /> : 'Analyze Renders'}</button>
                </div>
                <div className="flex flex-col">
                    <label className="text-sm font-medium mb-2">AI Analysis</label>
                    <div className="flex-grow p-4 bg-background border rounded overflow-auto">
                        {isLoading && !analysis && <div className="flex justify-center items-center
                        h-full"><LoadingSpinner/></div>}
                        {analysis && <MarkdownRenderer content={analysis} />}
                    </div>
                </div>
```

```
            </div>
        </div>
    );
};

// ===== SeoAuditor.tsx =====

import React, { useState } from 'react';
import { MagnifyingGlassIcon } from '../icons.tsx';
import { LoadingSpinner, MarkdownRenderer } from '../shared/index.tsx';
import { auditSeoFromUrlStream } from '../../services/index.ts';

export const SeoAuditor: React.FC = () => {
    const [url, setUrl] = useState('https://react.dev');
    const [analysis, setAnalysis] = useState('');
    const [isLoading, setIsLoading] = useState(false);
    const [error, setError] = useState('');

    const handleAudit = async () => {
        setIsLoading(true);
        setError('');
        setAnalysis('');
        try {
            const stream = auditSeoFromUrlStream(url);
            let fullResponse = '';
            for await (const chunk of stream) {
                fullResponse += chunk;
                setAnalysis(fullResponse);
            }
        } catch (e) {
            setError(e instanceof Error ? e.message : 'Failed to get analysis');
        } finally {
            setIsLoading(false);
        }
    };

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
            <header className="mb-6">
                <h1 className="text-3xl font-bold flex items-center">
                    <MagnifyingGlassIcon />
                    <span className="ml-3">AI SEO Auditor</span>
                </h1>
                <p className="text-text-secondary mt-1">Get an AI-powered SEO audit based on
                Gemini's knowledge of a URL.</p>
            </header>
            <div className="flex gap-2">
                <input type="text" value={url} onChange={e => setUrl(e.target.value)}
                className="flex-grow p-2 bg-surface border rounded-md"/>
                <button onClick={handleAudit} disabled={isLoading} className="btn-primary px-6
                py-2">{isLoading ? <LoadingSpinner/> : 'Audit'}</button>
            </div>
             <p className="text-xs text-center text-yellow-600 my-4 bg-yellow-400/10 p-2
             rounded-md">
                <strong>Note:</strong> This is a simulation. The AI uses its training data to
                infer information about the site and does not perform a live crawl.
            </p>
            <div className="flex-grow bg-surface p-4 border rounded-lg overflow-y-auto">
                <h3 className="font-bold mb-2">AI-Generated SEO Report</h3>
                {isLoading && !analysis && <div className="flex justify-center items-center
                h-full"><LoadingSpinner /></div>}
                {error && <p className="text-red-500">{error}</p>}
```

```
                {analysis && <MarkdownRenderer content={analysis} />}
            </div>
        </div>
    );
};

// ===== DataTransformer.tsx =====

import React, { useState } from 'react';
import { ArrowPathIcon } from '../icons.tsx';
import { useNotification } from '../../contexts/NotificationContext.tsx';

const exampleData = `[{"id":1,"name":"Alice"},{"id":2,"name":"Bob"}]`;

export const DataTransformer: React.FC = () => {
    const [input, setInput] = useState(exampleData);
    const [output, setOutput] = useState('');
    const [from, setFrom] = useState('json');
    const [to, setTo] = useState('csv');
    const { addNotification } = useNotification();

    const handleTransform = () => {
        try {
            if (from === to) {
                setOutput(input);
                return;
            }
            if (from === 'json' && to === 'csv') {
                const data = JSON.parse(input);
                if (!Array.isArray(data) || data.length === 0) {
                    setOutput('Input must be a non-empty array of objects for JSON to CSV
                    conversion.');
                    return;
                }
                const headers = Object.keys(data[0]);
                const csvRows = [
                    headers.join(','),
                    ...data.map(row =>
                        headers.map(header => JSON.stringify(row[header], (_, value) => value === null ?
                        '' : value)).join(',')
                    )
                ];
                setOutput(csvRows.join('\n'));
                addNotification('Transformed JSON to CSV!', 'success');
            } else if (from === 'csv' && to === 'json') {
                const [headerLine, ...rows] = input.trim().split('\n');
                const headers = headerLine.split(',').map(h => h.trim());
                const json = rows.map(row => {
                    const values = row.split(',');
                    return headers.reduce((obj: Record<string, any>, header, i) => {
                        obj[header] = values[i] ? values[i].trim() : '';
                        return obj;
                    }, {});
                });
                setOutput(JSON.stringify(json, null, 2));
                addNotification('Transformed CSV to JSON!', 'success');
            } else {
                setOutput(`// Transformation from ${from} to ${to} is not yet implemented.`);
                 addNotification('This transformation is not yet supported.', 'info');
            }
        } catch (e) {
            const message = e instanceof Error ? e.message : 'An unknown error occurred';
```

```
            setOutput(`Error: ${message}`);
            addNotification(`Transformation Error: ${message}`, 'error');
        }
    };

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
            <header className="mb-6">
                <h1 className="text-3xl font-bold flex items-center">
                    <ArrowPathIcon />
                    <span className="ml-3">Data Transformer</span>
                </h1>
                <p className="text-text-secondary mt-1">Transform data between formats like CSV
                and JSON.</p>
            </header>
            <div className="flex items-center justify-center gap-4 mb-4">
                <select value={from} onChange={e => setFrom(e.target.value)} className="p-2 bg-
                surface border rounded">
                    <option>json</option><option>csv</option><option disabled>xml</option>
                </select>
                <span>to</span>
                <select value={to} onChange={e => setTo(e.target.value)} className="p-2 bg-
                surface border rounded">
                    <option>csv</option><option>json</option><option disabled>xml</option>
                </select>
                <button onClick={handleTransform} className="btn-primary px-6
                py-2">Transform</button>
            </div>
            <div className="flex-grow grid grid-cols-1 lg:grid-cols-2 gap-6 min-h-0">
                <div className="flex flex-col">
                    <label className="text-sm font-medium mb-2">Input</label>
                    <textarea value={input} onChange={e => setInput(e.target.value)}
                    className="flex-grow p-2 bg-surface border rounded font-mono text-xs"/>
                </div>
                <div className="flex flex-col">
                    <label className="text-sm font-medium mb-2">Output</label>
                    <textarea value={output} readOnly className="flex-grow p-2 bg-background border
                    rounded font-mono text-xs"/>
                </div>
            </div>
        </div>
    );
};

// ===== LoremIpsumGenerator.tsx =====

import React, { useState } from 'react';
import { DocumentTextIcon } from '../icons.tsx';
import { useNotification } from '../../contexts/NotificationContext.tsx';

const generateLorem = (count: number, type: 'paragraphs' | 'sentences' |
'words') => {
    const lorem = "Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do
    eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim
    veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo
    consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum
    dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident,
    sunt in culpa qui officia deserunt mollit anim id est laborum.";
    if (type === 'words') return lorem.split(' ').slice(0, count).join(' ');
    if (type === 'sentences') return lorem.split('. ').slice(0, count).join('. ') +
    '.';
    return Array(count).fill(lorem).join('\\n\\n');
```

```
};

export const LoremIpsumGenerator: React.FC = () => {
    const [count, setCount] = useState(3);
    const [type, setType] = useState<'paragraphs' | 'sentences' |
    'words'>('paragraphs');
    const [text, setText] = useState('');
    const { addNotification } = useNotification();

    const handleGenerate = () => {
        setText(generateLorem(count, type));
    };

    const handleCopy = () => {
        navigator.clipboard.writeText(text);
        addNotification('Text copied to clipboard!', 'success');
    };

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
            <header className="mb-6">
                <h1 className="text-3xl font-bold flex items-center">
                    <DocumentTextIcon />
                    <span className="ml-3">Lorem Ipsum Generator</span>
                </h1>
                <p className="text-text-secondary mt-1">Generate placeholder text with various
                options.</p>
            </header>
            <div className="flex-grow flex flex-col items-center gap-4">
                <div className="flex items-end gap-2 p-4 bg-surface rounded-lg border">
                    <input type="number" value={count} onChange={e =>
                    setCount(Number(e.target.value))} className="w-24 p-2 bg-background border
                    rounded"/>
                    <select value={type} onChange={e => setType(e.target.value as any)}
                    className="p-2 bg-background border rounded">
                        <option value="paragraphs">Paragraphs</option>
                        <option value="sentences">Sentences</option>
                        <option value="words">Words</option>
                    </select>
                    <button onClick={handleGenerate} className="btn-primary px-6
                    py-2">Generate</button>
                </div>
                <div className="w-full max-w-3xl flex-grow flex flex-col">
                     <textarea value={text} readOnly className="flex-grow p-4 bg-background border
                     rounded-lg" placeholder="Generated text will appear here..."/>
                     {text && <button onClick={handleCopy} className="btn-primary w-full mt-2
                     py-2">Copy to Clipboard</button>}
                </div>
            </div>
        </div>
    );
};


// ===== UuidGenerator.tsx =====

import React, { useState, useCallback } from 'react';
import { TerminalIcon } from '../icons.tsx';
import { useNotification } from '../../contexts/NotificationContext.tsx';

// A simple nanoid implementation for the demo
const nanoid = (size = 21) => crypto.getRandomValues(new
```

```
Uint8Array(size)).reduce((id, byte) => id + ((byte &= 63) < 36 ?
byte.toString(36) : (byte < 62 ? (byte - 26).toString(36).toUpperCase() : (byte
> 62 ? '-' : '_'))), '');

export const UuidGenerator: React.FC = () => {
    const [id, setId] = useState('');
    const [type, setType] = useState<'uuid' | 'nanoid'>('uuid');
    const { addNotification } = useNotification();

    const generateId = useCallback(() => {
        if (type === 'uuid') {
            setId(crypto.randomUUID());
        } else {
            setId(nanoid());
        }
    }, [type]);

    const handleCopy = () => {
        if (id) {
            navigator.clipboard.writeText(id);
            addNotification('ID copied to clipboard!', 'success');
        }
    };

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
            <header className="mb-6">
                <h1 className="text-3xl font-bold flex items-center">
                    <TerminalIcon />
                    <span className="ml-3">UUID/NanoID Generator</span>
                </h1>
                <p className="text-text-secondary mt-1">Generate universally unique
                identifiers.</p>
            </header>
            <div className="flex-grow flex flex-col items-center justify-center gap-4">
                <div className="bg-surface p-6 rounded-lg border w-full max-w-xl text-center">
                    <p className="font-mono text-lg text-primary break-all h-8">{id || 'Click
                    generate to start'}</p>
                </div>
                <div className="flex gap-2">
                    <select value={type} onChange={e => setType(e.target.value as any)}
                    className="p-2 bg-surface border rounded">
                        <option value="uuid">UUID v4</option>
                        <option value="nanoid">NanoID</option>
                    </select>
                    <button onClick={generateId} className="btn-primary px-6 py-2">Generate</button>
                    <button onClick={handleCopy} disabled={!id} className="btn-primary px-6
                    py-2">Copy</button>
                </div>
            </div>
        </div>
    );
};


// ===== Base64EncoderDecoder.tsx =====

import React, { useState } from 'react';
import { CodeBracketSquareIcon } from '../icons.tsx';

export const Base64EncoderDecoder: React.FC = () => {
    const [plain, setPlain] = useState('Hello World!');
```

```
    const [encoded, setEncoded] = useState('SGVsbG8gV29ybGQh');

    const handlePlainChange = (e: React.ChangeEvent<HTMLTextAreaElement>) => {
        const val = e.target.value;
        setPlain(val);
        try {
            setEncoded(btoa(val));
        } catch {
            setEncoded('Invalid input for Base64 encoding');
        }
    };

    const handleEncodedChange = (e: React.ChangeEvent<HTMLTextAreaElement>) => {
        const val = e.target.value;
        setEncoded(val);
        try {
            setPlain(atob(val));
        } catch {
            setPlain('Invalid Base64 string');
        }
    };

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
            <header className="mb-6">
                <h1 className="text-3xl font-bold flex items-center">
                    <CodeBracketSquareIcon />
                    <span className="ml-3">Base64 Encoder/Decoder</span>
                </h1>
                <p className="text-text-secondary mt-1">Encode and decode strings using
                Base64.</p>
            </header>
            <div className="flex-grow grid grid-cols-1 lg:grid-cols-2 gap-6 min-h-0">
                <div className="flex flex-col">
                    <label className="text-sm font-medium mb-2">Plain Text</label>
                    <textarea value={plain} onChange={handlePlainChange} className="flex-grow p-2
                    bg-surface border rounded font-mono text-xs"/>
                </div>
                <div className="flex flex-col">
                    <label className="text-sm font-medium mb-2">Base64 Encoded</label>
                    <textarea value={encoded} onChange={handleEncodedChange} className="flex-grow
                    p-2 bg-surface border rounded font-mono text-xs"/>
                </div>
            </div>
        </div>
    );
};


// ===== UrlInspector.tsx =====

import React, { useState, useMemo } from 'react';
import { LinkIcon } from '../icons.tsx';

export const UrlInspector: React.FC = () => {
    const [url, setUrl] = useState('https://example.com:8080/path/to/page?param1=val
    ue1&param2=value2#section');

    const parsed = useMemo(() => {
        try {
            return new URL(url);
        } catch {
```

```
            return null;
        }
    }, [url]);

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
            <header className="mb-6">
                <h1 className="text-3xl font-bold flex items-center">
                    <LinkIcon />
                    <span className="ml-3">URL Inspector/Parser</span>
                </h1>
                <p className="text-text-secondary mt-1">Parse and inspect the components of a
                URL.</p>
            </header>
            <div className="w-full max-w-3xl mx-auto">
                <input
                    type="text"
                    value={url}
                    onChange={e => setUrl(e.target.value)}
                    className="w-full p-2 bg-surface border rounded-md font-mono text-sm"
                />
                {parsed ? (
                    <div className="mt-4 bg-surface p-4 rounded-lg border grid grid-cols-2 gap-2
                    text-sm">
                        <strong>Protocol:</strong> <span>{parsed.protocol}</span>
                        <strong>Hostname:</strong> <span>{parsed.hostname}</span>
                        <strong>Port:</strong> <span>{parsed.port}</span>
                        <strong>Pathname:</strong> <span>{parsed.pathname}</span>
                        <strong>Search:</strong> <span>{parsed.search}</span>
                        <strong>Hash:</strong> <span>{parsed.hash}</span>
                    </div>
                ) : (
                    <p className="mt-4 text-red-500">Invalid URL</p>
                )}
            </div>
        </div>
    );
};


// ===== JwtInspector.tsx =====

import React, { useState, useMemo } from 'react';
import { LockClosedIcon } from '../icons.tsx';

const exampleJwt = "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODk
wIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36P
Ok6yJV_adQssw5c";

export const JwtInspector: React.FC = () => {
    const [jwt, setJwt] = useState(exampleJwt);

    const { header, payload, error } = useMemo(() => {
        try {
            const parts = jwt.split('.');
            if (parts.length !== 3) return { error: 'Invalid JWT structure' };
            const header = JSON.parse(atob(parts[0]));
            const payload = JSON.parse(atob(parts[1]));
            return { header, payload, error: null };
        } catch (e) {
            return { error: 'Invalid JWT format' };
        }
```

```
    }, [jwt]);

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
            <header className="mb-6">
                <h1 className="text-3xl font-bold flex items-center">
                    <LockClosedIcon />
                    <span className="ml-3">JWT Inspector</span>
                </h1>
                <p className="text-text-secondary mt-1">Decode and inspect JSON Web Tokens.</p>
            </header>
            <div className="flex-grow grid grid-cols-1 lg:grid-cols-2 gap-6 min-h-0">
                <div className="flex flex-col">
                    <label className="text-sm font-medium mb-2">JWT</label>
                    <textarea value={jwt} onChange={e => setJwt(e.target.value)} className="flex-
                    grow p-2 bg-surface border rounded font-mono text-xs"/>
                </div>
                <div className="flex flex-col gap-4">
                    <div className="flex flex-col flex-1">
                        <label className="text-sm font-medium mb-2">Header</label>
                        <pre className="flex-grow p-2 bg-background border rounded text-xs">{error ?
                        error : JSON.stringify(header, null, 2)}</pre>
                    </div>
                    <div className="flex flex-col flex-1">
                        <label className="text-sm font-medium mb-2">Payload</label>
                        <pre className="flex-grow p-2 bg-background border rounded text-xs">{error ? ''
                        : JSON.stringify(payload, null, 2)}</pre>
                    </div>
                </div>
            </div>
        </div>
    );
};


// ===== CspGenerator.tsx =====

import React, { useState, useCallback } from 'react';
import { ShieldCheckIcon } from '../icons.tsx';
import { useNotification } from '../../contexts/NotificationContext.tsx';
import { generateCspFromDescription } from '../../services/index.ts';
import { LoadingSpinner } from '../shared/index.tsx';

export const CspGenerator: React.FC = () => {
    const [description, setDescription] = useState("A standard policy for a React
    app using Google Fonts and fetching data from its own API subdomain
    (api.example.com).");
    const [policy, setPolicy] = useState('');
    const [isLoading, setIsLoading] = useState(false);
    const { addNotification } = useNotification();

    const handleGenerate = useCallback(async () => {
        if (!description.trim()) {
            addNotification('Please enter a description for the policy.', 'error');
            return;
        }
        setIsLoading(true);
        setPolicy('');
        try {
            const stream = generateCspFromDescription(description);
            let fullResponse = '';
            for await (const chunk of stream) {
```

```
                fullResponse += chunk;
                setPolicy(fullResponse);
            }
            addNotification('CSP generated!', 'success');
        } catch (e) {
             addNotification('Failed to generate CSP.', 'error');
        } finally {
            setIsLoading(false);
        }
    }, [description, addNotification]);

    const handleCopy = () => {
        navigator.clipboard.writeText(policy);
        addNotification('CSP copied to clipboard!', 'success');
    };

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
            <header className="mb-6">
                <h1 className="text-3xl font-bold flex items-center">
                    <ShieldCheckIcon />
                    <span className="ml-3">CSP Generator</span>
                </h1>
                <p className="text-text-secondary mt-1">Generate a Content Security Policy for
                your web application using AI.</p>
            </header>
            <div className="flex-grow flex flex-col gap-4">
                <div className="flex flex-col">
                    <label className="text-sm font-medium mb-2">Describe your requirements</label>
                    <textarea
                        value={description}
                        onChange={e => setDescription(e.target.value)}
                        className="w-full p-2 bg-surface border rounded h-24"
                    />
                </div>
                <button onClick={handleGenerate} disabled={isLoading} className="btn-primary
                w-full max-w-sm mx-auto py-2">
                    {isLoading ? <LoadingSpinner /> : 'Generate Policy'}
                </button>
                <div className="flex flex-col flex-grow min-h-0">
                    <label className="text-sm font-medium mb-2">Generated Policy</label>
                    <pre className="relative flex-grow p-4 bg-background border rounded-lg text-
                    primary font-mono text-sm overflow-auto">
                        {isLoading && !policy && <div className="absolute inset-0 flex items-center
                        justify-center"><LoadingSpinner /></div>}
                        {policy}
                    </pre>
                    {policy && <button onClick={handleCopy} className="btn-primary mt-2 px-4 py-2
                    self-start">Copy Policy</button>}
                </div>
            </div>
        </div>
    );
};

// ===== RedosScanner.tsx =====

import React, { useState } from 'react';
import { BugAntIcon } from '../icons.tsx';
import { analyzeRegexForRedosStream } from '../../services/index.ts';
import { LoadingSpinner, MarkdownRenderer } from '../shared/index.tsx';
export const RedosScanner: React.FC = () => {
```

```
    const [regex, setRegex] = useState('(a+)+');
    const [analysis, setAnalysis] = useState('');
    const [isLoading, setIsLoading] = useState(false);

    const handleScan = async () => {
        setIsLoading(true);
        setAnalysis('');
        try {
            const stream = analyzeRegexForRedosStream(regex);
            let fullResponse = '';
            for await (const chunk of stream) {
                fullResponse += chunk;
                setAnalysis(fullResponse);
            }
        } catch (e) {
            setAnalysis(`Error: ${e instanceof Error ? e.message : 'Unknown error'}`);
        } finally {
            setIsLoading(false);
        }
    };

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
            <header className="mb-6">
                <h1 className="text-3xl font-bold flex items-center">
                    <BugAntIcon />
                    <span className="ml-3">Regex DoS Scanner</span>
                </h1>
                <p className="text-text-secondary mt-1">Scan regular expressions for potential
                Denial of Service vulnerabilities.</p>
            </header>
            <div className="flex-grow flex flex-col items-center justify-center gap-4">
                <div className="w-full max-w-lg">
                    <label className="text-sm font-medium mb-2">Regular Expression</label>
                    <div className="flex gap-2">
                        <input
                            type="text"
                            value={regex}
                            onChange={e => setRegex(e.target.value)}
                            className="flex-grow p-2 bg-surface border rounded-md font-mono"
                        />
                        <button onClick={handleScan} disabled={isLoading} className="btn-primary px-6
                        py-2">{isLoading ? <LoadingSpinner /> : 'Scan'}</button>
                    </div>
                </div>
                <div className="mt-6 w-full max-w-2xl flex-grow flex flex-col min-h-[200px]">
                    <label className="text-sm font-medium mb-2">AI Analysis</label>
                    <div className="flex-grow p-4 bg-background border rounded-lg overflow-y-auto">
                        {isLoading && !analysis && <div className="flex justify-center items-center
                        h-full"><LoadingSpinner /></div>}
                        {analysis && <MarkdownRenderer content={analysis} />}
                    </div>
                </div>
            </div>
        </div>
    );
};

// ===== DependencyVulnerabilityScanner.tsx =====

import React, { useState } from 'react';
import { ArchiveBoxIcon } from '../icons.tsx';
```

```
import { LoadingSpinner, MarkdownRenderer } from '../shared/index.tsx';
import { analyzePackageJsonStream } from '../../services/index.ts';

const examplePackageJson = `{
  "dependencies": {
    "lodash": "4.17.15",
    "express": "4.17.1"
  }
}`;

export const DependencyVulnerabilityScanner: React.FC = () => {
    const [pkgJson, setPkgJson] = useState(examplePackageJson);
    const [analysis, setAnalysis] = useState('');
    const [isLoading, setIsLoading] = useState(false);

    const handleScan = async () => {
        setIsLoading(true);
        setAnalysis('');
        try {
            const stream = analyzePackageJsonStream(pkgJson);
            let fullResponse = '';
            for await (const chunk of stream) {
                fullResponse += chunk;
                setAnalysis(fullResponse);
            }
        } catch (e) {
            setAnalysis(`Error: ${e instanceof Error ? e.message : 'Unknown error'}`);
        } finally {
            setIsLoading(false);
        }
    };

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
            <header className="mb-6">
                <h1 className="text-3xl font-bold flex items-center">
                    <ArchiveBoxIcon />
                    <span className="ml-3">Dependency Vulnerability Scanner</span>
                </h1>
                <p className="text-text-secondary mt-1">AI-powered scan of a package.json for
                known vulnerabilities.</p>
            </header>
            <div className="flex-grow grid grid-cols-1 lg:grid-cols-2 gap-6 min-h-0">
                <div className="flex flex-col">
                    <label className="text-sm font-medium mb-2">package.json content</label>
                    <textarea value={pkgJson} onChange={e => setPkgJson(e.target.value)}
                    className="flex-grow p-2 bg-surface border rounded font-mono text-xs"/>
                    <button onClick={handleScan} disabled={isLoading} className="btn-primary w-full
                    mt-4 py-3">{isLoading ? <LoadingSpinner/> : 'Scan with AI'}</button>
                </div>
                <div className="flex flex-col">
                    <label className="text-sm font-medium mb-2">AI Analysis</label>
                    <div className="flex-grow p-4 bg-background border rounded overflow-auto">
                        {isLoading && !analysis && <div className="flex justify-center items-center
                        h-full"><LoadingSpinner /></div>}
                        {analysis && <MarkdownRenderer content={analysis} />}
                    </div>
                </div>
            </div>
        </div>
    );
};
```

```
// ===== CorsProxySimulator.tsx =====

import React, { useState } from 'react';
import { PaperAirplaneIcon } from '../icons.tsx';
import { explainCorsError } from '../../services/index.ts';
import { LoadingSpinner, MarkdownRenderer } from '../shared/index.tsx';

export const CorsProxySimulator: React.FC = () => {
    const [origin, setOrigin] = useState('https://evil.com');
    const [target, setTarget] = useState('https://api.myapp.com/data');
    const [result, setResult] = useState('');
    const [isLoading, setIsLoading] = useState(false);

    const handleSimulate = async () => {
        setIsLoading(true);
        setResult('');
        try {
            const stream = explainCorsError(origin, target, { 'X-Requested-With':
            'XMLHttpRequest' });
            let fullResponse = '';
            for await (const chunk of stream) {
                fullResponse += chunk;
                setResult(fullResponse);
            }
        } catch (e) {
            setResult('Error getting explanation.');
        } finally {
            setIsLoading(false);
        }
    };

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
            <header className="mb-6">
                <h1 className="text-3xl font-bold flex items-center">
                    <PaperAirplaneIcon />
                    <span className="ml-3">CORS Simulator & Explainer</span>
                </h1>
                <p className="text-text-secondary mt-1">Simulate a cross-origin request and get
                an AI-powered explanation of the result.</p>
            </header>
            <div className="flex-grow flex flex-col items-center justify-center gap-4">
                <div className="w-full max-w-lg bg-surface p-6 rounded-lg border">
                    <h3 className="font-bold mb-2">Simulated Request</h3>
                    <div className="grid grid-cols-2 gap-4">
                        <div>
                            <label className="text-sm">Origin</label>
                            <input value={origin} onChange={e => setOrigin(e.target.value)}
                            className="w-full p-2 bg-background border rounded mt-1"/>
                        </div>
                        <div>
                             <label className="text-sm">Target</label>
                            <input value={target} onChange={e => setTarget(e.target.value)}
                            className="w-full p-2 bg-background border rounded mt-1"/>
                        </div>
                    </div>
                    <button onClick={handleSimulate} disabled={isLoading} className="btn-primary
                    w-full mt-4 py-2">
                        {isLoading ? <LoadingSpinner /> : 'Simulate Request & Get Explanation'}
                    </button>
                </div>
                <div className="mt-4 w-full max-w-2xl flex-grow min-h-[200px] flex flex-col">
```

```
                    <label className="text-sm font-medium mb-2">AI Explanation</label>
                    <div className="flex-grow p-4 bg-background border rounded-lg overflow-y-auto">
                        {isLoading && !result && <div className="flex justify-center items-center
                        h-full"><LoadingSpinner/></div>}
                        {result && <MarkdownRenderer content={result} />}
                    </div>
                </div>
            </div>
        </div>
    );
};

// ===== ImagePlaceholderGenerator.tsx =====

import React, { useState, useMemo } from 'react';
import { PhotoIcon } from '../icons.tsx';
import { useNotification } from '../../contexts/NotificationContext.tsx';

export const ImagePlaceholderGenerator: React.FC = () => {
    const [width, setWidth] = useState(300);
    const [height, setHeight] = useState(200);
    const [text, setText] = useState('');
    const { addNotification } = useNotification();

    const url = useMemo(() => {
        return `https://via.placeholder.com/${width}x${height}${text ?
        `?text=${encodeURIComponent(text)}` : ''}`;
    }, [width, height, text]);

    const handleCopy = () => {
        navigator.clipboard.writeText(url);
        addNotification('URL copied to clipboard!', 'success');
    };

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
            <header className="mb-6">
                <h1 className="text-3xl font-bold flex items-center">
                    <PhotoIcon />
                    <span className="ml-3">Image Placeholder Generator</span>
                </h1>
                <p className="text-text-secondary mt-1">Generate placeholder images of any size
                and color.</p>
            </header>
            <div className="flex-grow flex flex-col items-center gap-4">
                <div className="flex items-center gap-2 p-4 bg-surface rounded-lg border">
                    <input type="number" value={width} onChange={e =>
                    setWidth(Number(e.target.value))} className="w-24 p-2 bg-background border
                    rounded"/>
                    <span>x</span>
                    <input type="number" value={height} onChange={e =>
                    setHeight(Number(e.target.value))} className="w-24 p-2 bg-background border
                    rounded"/>
                     <input type="text" value={text} onChange={e => setText(e.target.value)}
                     placeholder="Optional text..." className="p-2 bg-background border rounded"/>
                </div>
                <div className="p-4 bg-background border rounded-lg">
                    <img src={url} alt="placeholder" />
                </div>
                <div className="flex items-center gap-2 p-2 bg-surface rounded-lg border w-full
                max-w-xl">
                    <input type="text" value={url} readOnly className="flex-grow p-2 bg-background
```

```
                    border rounded font-mono text-sm"/>
                    <button onClick={handleCopy} className="btn-primary px-4 py-2">Copy URL</button>
                </div>
            </div>
        </div>
    );
};


// ===== MockUserDataGenerator.tsx =====

import React, { useState } from 'react';
import { DocumentTextIcon, SparklesIcon } from '../icons.tsx';
import { generateMockData } from '../../services/aiService.ts';
import { LoadingSpinner } from '../shared/index.tsx';
import { useNotification } from '../../contexts/NotificationContext.tsx';

export const MockUserDataGenerator: React.FC = () => {
    const [data, setData] = useState('');
    const [isLoading, setIsLoading] = useState(false);
    const { addNotification } = useNotification();

    const handleGenerate = async () => {
        setIsLoading(true);
        try {
            const result = await generateMockData('a user with id, name, email, and avatar
            url', 10);
            setData(JSON.stringify(result, null, 2));
        } catch (e) {
            addNotification('Failed to generate data', 'error');
        } finally {
            setIsLoading(false);
        }
    };

    const handleCopy = () => {
        navigator.clipboard.writeText(data);
        addNotification('Data copied to clipboard!', 'success');
    };

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
            <header className="mb-6">
                <h1 className="text-3xl font-bold flex items-center">
                    <DocumentTextIcon />
                    <span className="ml-3">Mock User Data Generator</span>
                </h1>
                <p className="text-text-secondary mt-1">Generate realistic mock user data for
                testing.</p>
            </header>
            <div className="flex-grow flex flex-col items-center gap-4">
                <button onClick={handleGenerate} disabled={isLoading} className="btn-primary
                px-6 py-3 flex items-center gap-2">
                    {isLoading ? <LoadingSpinner /> : <><SparklesIcon /> Generate 10 Users</>}
                </button>
                <div className="w-full max-w-3xl flex-grow flex flex-col">
                     <textarea value={data} readOnly className="flex-grow p-4 bg-background border
                     rounded-lg font-mono text-xs" placeholder="Generated data will appear here..."/>
                     {data && <button onClick={handleCopy} className="btn-primary w-full mt-2
                     py-2">Copy JSON</button>}
                </div>
            </div>
```

```
        </div>
    );
};


// ===== FeatureFlagSimulator.tsx =====

import React, { useState } from 'react';
import { BeakerIcon } from '../icons.tsx';

export const FeatureFlagSimulator: React.FC = () => {
    const [flags, setFlags] = useState({
        newDashboard: true,
        darkMode: false,
        betaFeature: false,
    });

    const handleToggle = (flag: keyof typeof flags) => {
        setFlags(prev => ({ ...prev, [flag]: !prev[flag] }));
    };

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
            <header className="mb-6">
                <h1 className="text-3xl font-bold flex items-center">
                    <BeakerIcon />
                    <span className="ml-3">Feature Flag Simulator</span>
                </h1>
                <p className="text-text-secondary mt-1">Simulate and test feature flags in your
                application.</p>
            </header>
            <div className="flex-grow grid grid-cols-1 lg:grid-cols-2 gap-6">
                <div className="bg-surface p-4 rounded-lg border">
                    <h3 className="font-bold mb-2">Controls</h3>
                    {Object.entries(flags).map(([key, value]) => (
                        <div key={key} className="flex items-center justify-between p-2">
                            <code>{key}</code>
                            <button onClick={() => handleToggle(key as keyof typeof flags)} className={`px-2
                            py-1 rounded-full text-xs ${value ? 'bg-green-500 text-white' : 'bg-
                            gray-200'}`}>{value ? 'ON' : 'OFF'}</button>
                        </div>
                    ))}
                </div>
                 <div className="bg-background p-4 rounded-lg border">
                    <h3 className="font-bold mb-2">Simulated App State</h3>
                    {flags.newDashboard && <div className="p-2 bg-blue-500/10 text-blue-700 rounded
                    mb-2">New Dashboard is visible</div>}
                    {flags.darkMode && <div className="p-2 bg-gray-800/10 text-gray-400 rounded
                    mb-2">Dark Mode is enabled</div>}
                    {flags.betaFeature && <div className="p-2 bg-yellow-500/10 text-yellow-700
                    rounded mb-2">Secret Beta Feature is enabled</div>}
                </div>
            </div>
        </div>
    );
};


// ===== ErrorResponseSimulator.tsx =====

import React, { useState, useMemo } from 'react';
import { ServerStackIcon } from '../icons.tsx';
```

```
import { useNotification } from '../../contexts/NotificationContext.tsx';

export const ErrorResponseSimulator: React.FC = () => {
    const [status, setStatus] = useState(404);
    const { addNotification } = useNotification();

    const response = useMemo(() => {
        const messages: Record<number, object> = {
            400: { error: "Bad Request", message: "Invalid input provided" },
            401: { error: "Unauthorized", message: "Authentication token is missing or
            invalid" },
            403: { error: "Forbidden", message: "You do not have permission to access this
            resource" },
            404: { error: "Not Found", message: "The requested resource could not be found"
            },
            500: { error: "Internal Server Error", message: "An unexpected error occurred on
            the server" },
        };
        return JSON.stringify(messages[status], null, 2);
    }, [status]);

    const handleTrigger = () => {
        addNotification(`Simulated API request failed with status ${status}`, 'error');
    };

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
            <header className="mb-6">
                <h1 className="text-3xl font-bold flex items-center">
                    <ServerStackIcon />
                    <span className="ml-3">Error Response Simulator</span>
                </h1>
                <p className="text-text-secondary mt-1">Simulate API error responses (404, 500,
                etc.).</p>
            </header>
            <div className="flex-grow flex flex-col items-center gap-4">
                <div className="flex items-center gap-2 p-4 bg-surface rounded-lg border">
                    <label>HTTP Status:</label>
                    <select value={status} onChange={e => setStatus(Number(e.target.value))}
                    className="p-2 bg-background border rounded">
                        <option>400</option><option>401</option><option>403</option><option>404</option>
                        <option>500</option>
                    </select>
                    <button onClick={handleTrigger} className="btn-primary px-6
                    py-2">Trigger</button>
                </div>
                <div className="w-full max-w-2xl flex-grow flex flex-col">
                    <label className="text-sm font-medium mb-2">Response Body</label>
                    <pre className="flex-grow p-4 bg-background border rounded-lg text-red-500 font-
                    mono text-xs">{response}</pre>
                </div>
            </div>
        </div>
    );
};


// ===== WebhookEventSimulator.tsx =====

import React, { useState } from 'react';
import { PaperAirplaneIcon, SparklesIcon } from '../icons.tsx';
import { useNotification } from '../../contexts/NotificationContext.tsx';
```

```
import { generateWebhookPayload } from '../../services/index.ts';
import { LoadingSpinner } from '../shared/index.tsx';

export const WebhookEventSimulator: React.FC = () => {
    const [prompt, setPrompt] = useState('a GitHub push event to the main branch');
    const [payload, setPayload] = useState('');
    const [isLoading, setIsLoading] = useState(false);
    const { addNotification } = useNotification();

    const handleGenerate = async () => {
        setIsLoading(true);
        setPayload('');
        try {
            const result = await generateWebhookPayload(prompt);
            const jsonMatch = result.match(/```json\n([\s\S]*?)\n```/);
            setPayload(jsonMatch ? jsonMatch[1] : result);
        } catch (e) {
            addNotification('Failed to generate payload', 'error');
        } finally {
            setIsLoading(false);
        }
    };

    const handleCopy = () => {
        navigator.clipboard.writeText(payload);
        addNotification('Payload copied!', 'success');
    };

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
            <header className="mb-6">
                <h1 className="text-3xl font-bold flex items-center">
                    <PaperAirplaneIcon />
                    <span className="ml-3">Webhook Event Simulator</span>
                </h1>
                <p className="text-text-secondary mt-1">Generate realistic webhook payloads
                using AI.</p>
            </header>
            <div className="flex-grow flex flex-col items-center gap-4">
                <div className="flex items-center gap-2 p-4 bg-surface rounded-lg border w-full
                max-w-2xl">
                    <input
                        value={prompt}
                        onChange={e => setPrompt(e.target.value)}
                        className="flex-grow p-2 bg-background border rounded"
                        placeholder="Describe the event..."
                    />
                    <button onClick={handleGenerate} disabled={isLoading} className="btn-primary
                    px-4 py-2 flex items-center gap-2">
                        {isLoading ? <LoadingSpinner /> : <><SparklesIcon /> Generate</>}
                    </button>
                </div>
                 <div className="w-full max-w-3xl flex-grow flex flex-col">
                     <pre className="flex-grow p-4 bg-background border rounded-lg font-mono text-xs
                     overflow-auto">{payload}</pre>
                     {payload && <button onClick={handleCopy} className="btn-primary w-full mt-2
                     py-2">Copy Payload</button>}
                </div>
            </div>
        </div>
    );
};
```

```
// ===== UseDebounceHookGenerator.tsx =====

import React, { useState, useEffect } from 'react';
import { CodeBracketSquareIcon } from '../icons.tsx';
import { MarkdownRenderer } from '../shared/index.tsx';

const useDebounce = (value: string, delay: number) => {
    const [debouncedValue, setDebouncedValue] = useState(value);
    useEffect(() => {
        const handler = setTimeout(() => {
            setDebouncedValue(value);
        }, delay);
        return () => { clearTimeout(handler); };
    }, [value, delay]);
    return debouncedValue;
};

const hookCode = `
\`\`\`tsx
import { useState, useEffect } from 'react';

export const useDebounce = (value, delay) => {
    const [debouncedValue, setDebouncedValue] = useState(value);

    useEffect(() => {
        const handler = setTimeout(() => {
            setDebouncedValue(value);
        }, delay);

        return () => {
            clearTimeout(handler);
        };
    }, [value, delay]);

    return debouncedValue;
};
\`\`\`
`;

export const UseDebounceHookGenerator: React.FC = () => {
    const [searchTerm, setSearchTerm] = useState('');
    const debouncedSearchTerm = useDebounce(searchTerm, 500);

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
            <header className="mb-6">
                <h1 className="text-3xl font-bold flex items-center">
                    <CodeBracketSquareIcon />
                    <span className="ml-3">useDebounce Hook Generator</span>
                </h1>
                <p className="text-text-secondary mt-1">Generate a custom useDebounce hook.</p>
            </header>
            <div className="flex-grow grid grid-cols-1 lg:grid-cols-2 gap-6 min-h-0">
                <div className="flex flex-col">
                    <label className="text-sm font-medium mb-2">Hook Code</label>
                    <div className="flex-grow p-1 bg-background border rounded overflow-auto">
                        <MarkdownRenderer content={hookCode} />
                    </div>
                </div>
                <div className="flex flex-col">
                    <label className="text-sm font-medium mb-2">Live Demo</label>
                    <div className="flex-grow p-4 bg-surface border rounded">
```

```
                        <input
                            type="text"
                            placeholder="Type here..."
                            value={searchTerm}
                            onChange={e => setSearchTerm(e.target.value)}
                            className="w-full p-2 bg-background border rounded"
                        />
                        <div className="mt-4">
                            <p><strong>Typing:</strong> {searchTerm}</p>
                            <p><strong>Debounced:</strong> {debouncedSearchTerm}</p>
                        </div>
                    </div>
                </div>
            </div>
        </div>
    );
};


// ===== UseLocalStorageHookGenerator.tsx =====

import React, { useState } from 'react';
import { CodeBracketSquareIcon } from '../icons.tsx';
import { MarkdownRenderer } from '../shared/index.tsx';
import { useLocalStorage } from '../../hooks/useLocalStorage.ts';

const hookCode = `
\`\`\`tsx
import { useState } from 'react';

export const useLocalStorage = (key, initialValue) => {
    const [storedValue, setStoredValue] = useState(() => {
        try {
            const item = window.localStorage.getItem(key);
            return item ? JSON.parse(item) : initialValue;
        } catch (error) {
            console.log(error);
            return initialValue;
        }
    });

    const setValue = (value) => {
        try {
            const valueToStore = value instanceof Function ? value(storedValue) : value;
            setStoredValue(valueToStore);
            window.localStorage.setItem(key, JSON.stringify(valueToStore));
        } catch (error) {
            console.log(error);
        }
    };
    return [storedValue, setValue];
};
\`\`\`
`;

export const UseLocalStorageHookGenerator: React.FC = () => {
    const [name, setName] = useLocalStorage('demo-name', 'Alice');

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
            <header className="mb-6">
                <h1 className="text-3xl font-bold flex items-center">
```

```
                    <CodeBracketSquareIcon />
                    <span className="ml-3">useLocalStorage Hook Generator</span>
                </h1>
                <p className="text-text-secondary mt-1">Generate a custom useLocalStorage
                hook.</p>
            </header>
            <div className="flex-grow grid grid-cols-1 lg:grid-cols-2 gap-6 min-h-0">
                <div className="flex flex-col">
                    <label className="text-sm font-medium mb-2">Hook Code</label>
                    <div className="flex-grow p-1 bg-background border rounded overflow-auto">
                        <MarkdownRenderer content={hookCode} />
                    </div>
                </div>
                <div className="flex flex-col">
                    <label className="text-sm font-medium mb-2">Live Demo</label>
                    <div className="flex-grow p-4 bg-surface border rounded">
                        <input
                            type="text"
                            value={name}
                            onChange={e => setName(e.target.value)}
                            className="w-full p-2 bg-background border rounded"
                        />
                        <p className="mt-4">This input is saved to local storage. Try refreshing the
                        page.</p>
                    </div>
                </div>
            </div>
        </div>
    );
};


// ===== UseEventListenerHookGenerator.tsx =====

import React, { useState, useEffect, useRef } from 'react';
import { CodeBracketSquareIcon } from '../icons.tsx';
import { MarkdownRenderer } from '../shared/index.tsx';

const useEventListener = (eventName: string, handler: (event: any) => void,
element = window) => {
    const savedHandler = useRef<((event: any) => void) | null>(null);

    useEffect(() => {
        savedHandler.current = handler;
    }, [handler]);

    useEffect(() => {
        const isSupported = element && element.addEventListener;
        if (!isSupported) return;

        const eventListener = (event: any) => savedHandler.current?.(event);
        element.addEventListener(eventName, eventListener);
        return () => {
            element.removeEventListener(eventName, eventListener);
        };
    }, [eventName, element]);
};

const hookCode = `
\`\`\`tsx
import { useEffect, useRef } from 'react';
export const useEventListener = (eventName, handler, element = window) => {
```

```
    const savedHandler = useRef(null);

    useEffect(() => {
        savedHandler.current = handler;
    }, [handler]);

    useEffect(() => {
        const isSupported = element && element.addEventListener;
        if (!isSupported) return;

        const eventListener = (event) => savedHandler.current(event);
        element.addEventListener(eventName, eventListener);
        return () => {
            element.removeEventListener(eventName, eventListener);
        };
    }, [eventName, element]);
};
\`\`\`
`;

export const UseEventListenerHookGenerator: React.FC = () => {
    const [coords, setCoords] = useState({ x: 0, y: 0 });

    useEventListener('mousemove', (event) => {
        setCoords({ x: event.clientX, y: event.clientY });
    });

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
            <header className="mb-6">
                <h1 className="text-3xl font-bold flex items-center">
                    <CodeBracketSquareIcon />
                    <span className="ml-3">useEventListener Hook Generator</span>
                </h1>
                <p className="text-text-secondary mt-1">Generate a custom useEventListener
                hook.</p>
            </header>
            <div className="flex-grow grid grid-cols-1 lg:grid-cols-2 gap-6 min-h-0">
                <div className="flex flex-col">
                    <label className="text-sm font-medium mb-2">Hook Code</label>
                    <div className="flex-grow p-1 bg-background border rounded overflow-auto">
                        <MarkdownRenderer content={hookCode} />
                    </div>
                </div>
                <div className="flex flex-col">
                    <label className="text-sm font-medium mb-2">Live Demo</label>
                    <div className="flex-grow p-4 bg-surface border rounded flex items-center
                    justify-center">
                        <p>Mouse position: ({coords.x}, {coords.y})</p>
                    </div>
                </div>
            </div>
        </div>
    );
};


// ===== UseFetchHookGenerator.tsx =====

import React, { useState, useEffect } from 'react';
import { CodeBracketSquareIcon } from '../icons.tsx';
import { MarkdownRenderer } from '../shared/index.tsx';
```

```
import { LoadingSpinner } from '../shared/index.tsx';

const hookCode = `
\`\`\`tsx
import { useState, useEffect } from 'react';

export const useFetch = (url) => {
    const [data, setData] = useState(null);
    const [error, setError] = useState(null);
    const [loading, setLoading] = useState(true);

    useEffect(() => {
        const fetchData = async () => {
            try {
                const response = await fetch(url);
                const json = await response.json();
                setData(json);
            } catch (e) {
                setError(e);
            } finally {
                setLoading(false);
            }
        };
        fetchData();
    }, [url]);

    return { data, error, loading };
};
\`\`\`
`;

const useFetch = (url: string) => {
    const [data, setData] = useState(null);
    const [error, setError] = useState(null);
    const [loading, setLoading] = useState(true);

    useEffect(() => {
        const fetchData = async () => {
            setLoading(true);
            try {
                const response = await fetch(url);
                const json = await response.json();
                setData(json);
            } catch (e: any) {
                setError(e);
            } finally {
                setLoading(false);
            }
        };
        fetchData();
    }, [url]);

    return { data, error, loading };
};

export const UseFetchHookGenerator: React.FC = () => {
    const [postId, setPostId] = useState(1);
    const { data, error, loading } =
    useFetch(`https://jsonplaceholder.typicode.com/posts/${postId}`);

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
```

```
            <header className="mb-6">
                <h1 className="text-3xl font-bold flex items-center">
                    <CodeBracketSquareIcon />
                    <span className="ml-3">useFetch Hook Generator</span>
                </h1>
                <p className="text-text-secondary mt-1">Generate a custom useFetch hook for data
                fetching.</p>
            </header>
            <div className="flex-grow grid grid-cols-1 lg:grid-cols-2 gap-6 min-h-0">
                <div className="flex flex-col">
                    <label className="text-sm font-medium mb-2">Hook Code</label>
                    <div className="flex-grow p-1 bg-background border rounded overflow-auto">
                        <MarkdownRenderer content={hookCode} />
                    </div>
                </div>
                <div className="flex flex-col">
                    <label className="text-sm font-medium mb-2">Live Demo</label>
                    <div className="flex-grow p-4 bg-surface border rounded">
                        <div className="flex gap-2 mb-4">
                            <button onClick={() => setPostId(p => p + 1)} className="btn-primary p-2">Fetch
                            Next Post</button>
                        </div>
                        {loading && <LoadingSpinner />}
                        {error && <p className="text-red-500">Error fetching data</p>}
                        {data && <pre className="text-xs bg-background p-2
                        rounded">{JSON.stringify(data, null, 2)}</pre>}
                    </div>
                </div>
            </div>
        </div>
    );
};


// ===== UseFormHookGenerator.tsx =====

import React, { useState } from 'react';
import { CodeBracketSquareIcon } from '../icons.tsx';
import { MarkdownRenderer } from '../shared/index.tsx';

const hookCode = `
\`\`\`tsx
import { useState } from 'react';

export const useForm = (initialValues) => {
    const [values, setValues] = useState(initialValues);

    const handleChange = (e) => {
        const { name, value } = e.target;
        setValues({ ...values, [name]: value });
    };

    return [values, handleChange];
};
\`\`\`
`;

const useForm = (initialValues: any) => {
    const [values, setValues] = useState(initialValues);
    const handleChange = (e: React.ChangeEvent<HTMLInputElement>) => {
        const { name, value } = e.target;
        setValues({ ...values, [name]: value });
```

```
    };
    return [values, handleChange] as const;
};

export const UseFormHookGenerator: React.FC = () => {
    const [values, handleChange] = useForm({ name: '', email: ''});

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
            <header className="mb-6">
                <h1 className="text-3xl font-bold flex items-center">
                    <CodeBracketSquareIcon />
                    <span className="ml-3">useForm Hook Generator</span>
                </h1>
                <p className="text-text-secondary mt-1">Generate a custom useForm hook for form
                state management.</p>
            </header>
            <div className="flex-grow grid grid-cols-1 lg:grid-cols-2 gap-6 min-h-0">
                <div className="flex flex-col">
                    <label className="text-sm font-medium mb-2">Hook Code</label>
                    <div className="flex-grow p-1 bg-background border rounded overflow-auto">
                        <MarkdownRenderer content={hookCode} />
                    </div>
                </div>
                <div className="flex flex-col">
                    <label className="text-sm font-medium mb-2">Live Demo</label>
                    <div className="flex-grow p-4 bg-surface border rounded">
                        <div className="space-y-2">
                            <input name="name" value={values.name} onChange={handleChange}
                            placeholder="Name" className="w-full p-2 bg-background border rounded"/>
                            <input name="email" value={values.email} onChange={handleChange}
                            placeholder="Email" className="w-full p-2 bg-background border rounded"/>
                        </div>
                        <pre className="text-xs mt-4 bg-background p-2 rounded">{JSON.stringify(values,
                        null, 2)}</pre>
                    </div>
                </div>
            </div>
        </div>
    );
};


// ===== PillarOneGeos.tsx =====

import React, { useState, useCallback } from 'react';
import { ProjectExplorerIcon } from '../icons.tsx';
import { useNotification } from '../../contexts/NotificationContext.tsx';
import { LoadingSpinner, MarkdownRenderer } from '../shared/index.tsx';
import { generateMonetaryPolicy } from '../../services/index.ts';
import { PILLAR_FEATURES } from '../../constants.tsx';

const features = PILLAR_FEATURES['pillar-one-geos'];

export const PillarOneGeos: React.FC = () => {
    const [activeTab, setActiveTab] = useState(features[1].id); // Default to
    Monetary Policy Simulator
    const { addNotification } = useNotification();

    // State for Monetary Policy Simulator
    const [countryData, setCountryData] = useState('GDP: $50B, National Debt: 120%
    of GDP, Inflation: 15%, Political Stability: Low, Key Industries: Agriculture,
```

```
Tourism.');
const [isLoading, setIsLoading] = useState(false);
const [generatedPlan, setGeneratedPlan] = useState('');

const handleGeneratePolicy = useCallback(async () => {
    setIsLoading(true);
    setGeneratedPlan('');
    try {
        const plan = await generateMonetaryPolicy(countryData);
        setGeneratedPlan(plan);
        addNotification('Monetary policy generated!', 'success');
    } catch (e) {
        addNotification(e instanceof Error ? e.message : 'Failed to generate policy',
        'error');
    } finally {
        setIsLoading(false);
    }
}, [countryData, addNotification]);

const renderTabContent = () => {
    switch (activeTab) {
        case 'monetary-policy-simulator':
            return (
                <div className="grid grid-cols-1 md:grid-cols-2 gap-4 h-full">
                    <div className="flex flex-col gap-4">
                        <h3 className="text-lg font-bold">Nation State Parameters</h3>
                        <textarea
                            value={countryData}
                            onChange={e => setCountryData(e.target.value)}
                            className="w-full flex-grow p-2 bg-background border rounded text-sm"
                        />
                        <button onClick={handleGeneratePolicy} disabled={isLoading} className="btn-
                        primary w-full py-2">{isLoading ? <LoadingSpinner/> : 'Simulate 100-Year
                        Evolution'}</button>
                    </div>
                    <div className="flex flex-col">
                         <h3 className="text-lg font-bold mb-2">Generated Economic Plan</h3>
                         <div className="flex-grow p-2 bg-background border rounded overflow-auto">
                            {isLoading ? <div className="flex justify-center items-center
                            h-full"><LoadingSpinner /></div> : <MarkdownRenderer content={generatedPlan} />}
                        </div>
                    </div>
                </div>
            );
        case 'logistics-manifold':
        case 'scarcity-oracle':
        case 'urbanism-synthesizer':
             return <div className="text-center text-text-secondary p-8">This module is being
             forged.</div>;
        default:
            return null;
    }
};

return (
    <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary bg-
    background">
        <header className="mb-4 flex-shrink-0">
            <h1 className="text-3xl font-bold flex items-center">
                <ProjectExplorerIcon />
                <span className="ml-3">Pillar I: The GEOS Console</span>
            </h1>
```

```
                <p className="text-text-secondary mt-1">Orchestrate the planet's financial and
                logistical backbone.</p>
            </header>
            <div className="border-b border-border flex-shrink-0">
                {features.map(feature => (
                    <button key={feature.id} onClick={() => setActiveTab(feature.id)}
                    className={`px-4 py-2 text-sm font-medium ${activeTab === feature.id ?
                    'border-b-2 border-primary text-primary' : 'text-text-secondary'}`}>
                        {feature.name}
                    </button>
                ))}
            </div>
            <div className="flex-grow p-4 min-h-0">
                {renderTabContent()}
            </div>
        </div>
    );
};


// ===== PillarTwoCompassion.tsx =====

import React, { useState, useCallback } from 'react';
import { SparklesIcon } from '../icons.tsx';
import { useNotification } from '../../contexts/NotificationContext.tsx';
import { LoadingSpinner, MarkdownRenderer } from '../shared/index.tsx';
import { runGaiaCrucibleSimulation } from '../../services/index.ts';
import { PILLAR_FEATURES } from '../../constants.tsx';

const features = PILLAR_FEATURES['pillar-two-compassion'];

export const PillarTwoCompassion: React.FC = () => {
    const [activeTab, setActiveTab] = useState(features[0].id);
    const { addNotification } = useNotification();

    // State for Gaia's Crucible
    const [intervention, setIntervention] = useState('Stratospheric Aerosol
    Injection');
    const [intensity, setIntensity] = useState('Moderate');
    const [isLoading, setIsLoading] = useState(false);
    const [simulationResult, setSimulationResult] = useState('');

    const handleRunSimulation = useCallback(async () => {
        setIsLoading(true);
        setSimulationResult('');
        try {
            const result = await runGaiaCrucibleSimulation(intervention, intensity);
            setSimulationResult(result);
            addNotification('Climate simulation complete!', 'success');
        } catch (e) {
            addNotification(e instanceof Error ? e.message : 'Failed to run simulation',
            'error');
        } finally {
            setIsLoading(false);
        }
    }, [intervention, intensity, addNotification]);

    const renderTabContent = () => {
        switch (activeTab) {
            case 'gaias-crucible':
                return (
                    <div className="grid grid-cols-1 md:grid-cols-2 gap-4 h-full">
```

```
                    <div className="flex flex-col gap-4">
                        <h3 className="text-lg font-bold">Intervention Parameters</h3>
                        <div>
                            <label className="text-sm font-medium">Strategy</label>
                            <select value={intervention} onChange={e => setIntervention(e.target.value)}
                            className="w-full mt-1 p-2 bg-background border rounded">
                                <option>Stratospheric Aerosol Injection</option>
                                <option>Orbital Sunshades</option>
                                <option>Bio-engineered Carbon Capture</option>
                            </select>
                        </div>
                        <div>
                             <label className="text-sm font-medium">Intensity</label>
                            <select value={intensity} onChange={e => setIntensity(e.target.value)}
                            className="w-full mt-1 p-2 bg-background border rounded">
                                <option>Low</option>
                                <option>Moderate</option>
                                <option>High</option>
                            </select>
                        </div>
                        <button onClick={handleRunSimulation} disabled={isLoading} className="btn-
                        primary w-full py-2">{isLoading ? <LoadingSpinner/> : 'Run 1,000-Year
                        Simulation'}</button>
                    </div>
                    <div className="flex flex-col">
                         <h3 className="text-lg font-bold mb-2">Simulation Impact Report</h3>
                         <div className="flex-grow p-2 bg-background border rounded overflow-auto">
                            {isLoading ? <div className="flex justify-center items-center
                            h-full"><LoadingSpinner /></div> : <MarkdownRenderer content={simulationResult}
                            />}
                        </div>
                    </div>
                </div>
            );
        case 'genome-weaver':
        case 'aptitude-engine':
        case 'first-responder-ai':
             return <div className="text-center text-text-secondary p-8">This module is being
             forged.</div>;
        default:
            return null;
    }
};

return (
    <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary bg-
    background">
        <header className="mb-4 flex-shrink-0">
            <h1 className="text-3xl font-bold flex items-center">
                <SparklesIcon />
                <span className="ml-3">Pillar II: Computational Compassion</span>
            </h1>
            <p className="text-text-secondary mt-1">Apply planetary-scale optimization to
            humanity's most intractable problems.</p>
        </header>
        <div className="border-b border-border flex-shrink-0">
            {features.map(feature => (
                <button key={feature.id} onClick={() => setActiveTab(feature.id)}
                className={`px-4 py-2 text-sm font-medium ${activeTab === feature.id ?
                'border-b-2 border-primary text-primary' : 'text-text-secondary'}`}>
                    {feature.name}
                </button>
```

```
                ))}
            </div>
            <div className="flex-grow p-4 min-h-0">
                {renderTabContent()}
            </div>
        </div>
    );
};


// ===== PillarThreeMetaCreation.tsx =====

import React, { useState, useCallback } from 'react';
import { HammerIcon } from '../icons.tsx';
import { useNotification } from '../../contexts/NotificationContext.tsx';
import { LoadingSpinner, MarkdownRenderer } from '../shared/index.tsx';
import { refactorLegalCode } from '../../services/index.ts';
import { PILLAR_FEATURES } from '../../constants.tsx';

const features = PILLAR_FEATURES['pillar-three-meta-creation'];

const exampleLegalCode = `
Article 1: All citizens shall have the right to free speech.
Article 2: No citizen shall incite violence.
Article 3: In times of national emergency, the right to free speech may be
temporarily suspended to prevent panic.
`;

export const PillarThreeMetaCreation: React.FC = () => {
    const [activeTab, setActiveTab] = useState(features[1].id); // Default to Themis
    Engine
    const { addNotification } = useNotification();

    // State for Themis Engine
    const [legalCode, setLegalCode] = useState(exampleLegalCode);
    const [isLoading, setIsLoading] = useState(false);
    const [refactoredCode, setRefactoredCode] = useState('');

    const handleRefactor = useCallback(async () => {
        setIsLoading(true);
        setRefactoredCode('');
        try {
            const result = await refactorLegalCode(legalCode);
            setRefactoredCode(result);
            addNotification('Legal framework refactored!', 'success');
        } catch (e) {
            addNotification(e instanceof Error ? e.message : 'Failed to refactor code',
            'error');
        } finally {
            setIsLoading(false);
        }
    }, [legalCode, addNotification]);

    const renderTabContent = () => {
        switch (activeTab) {
            case 'themis-engine':
                return (
                    <div className="grid grid-cols-1 md:grid-cols-2 gap-4 h-full">
                        <div className="flex flex-col gap-4">
                            <h3 className="text-lg font-bold">Existing Legal Framework</h3>
                            <textarea
                                value={legalCode}
```

```
                            onChange={e => setLegalCode(e.target.value)}
                            className="w-full flex-grow p-2 bg-background border rounded text-sm"
                        />
                        <button onClick={handleRefactor} disabled={isLoading} className="btn-primary
                        w-full py-2">{isLoading ? <LoadingSpinner/> : 'Refactor Legal Code'}</button>
                    </div>
                    <div className="flex flex-col">
                         <h3 className="text-lg font-bold mb-2">Generated Framework</h3>
                         <div className="flex-grow p-2 bg-background border rounded overflow-auto">
                            {isLoading ? <div className="flex justify-center items-center
                            h-full"><LoadingSpinner /></div> : <MarkdownRenderer content={refactoredCode}
                            />}
                        </div>
                    </div>
                </div>
            );
        case 'hypothesis-forge':
        case 'memetic-catalyst':
        case 'the-exchange':
             return <div className="text-center text-text-secondary p-8">This module is being
             forged.</div>;
        default:
            return null;
    }
};

return (
    <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary bg-
    background">
        <header className="mb-4 flex-shrink-0">
            <h1 className="text-3xl font-bold flex items-center">
                <HammerIcon />
                <span className="ml-3">Pillar III: The Meta-Creation Console</span>
            </h1>
            <p className="text-text-secondary mt-1">Accelerate the very pace of discovery,
            creation, and cultural evolution.</p>
        </header>
        <div className="border-b border-border flex-shrink-0">
            {features.map(feature => (
                <button key={feature.id} onClick={() => setActiveTab(feature.id)}
                className={`px-4 py-2 text-sm font-medium ${activeTab === feature.id ?
                'border-b-2 border-primary text-primary' : 'text-text-secondary'}`}>
                    {feature.name}
                </button>
            ))}
        </div>
        <div className="flex-grow p-4 min-h-0">
            {renderTabContent()}
        </div>
    </div>
);
};


// ===== PillarFourGovernance.tsx =====

import React, { useState, useCallback } from 'react';
import { ShieldCheckIcon } from '../icons.tsx';
import { useNotification } from '../../contexts/NotificationContext.tsx';
import { LoadingSpinner, MarkdownRenderer } from '../shared/index.tsx';
import { generateContent } from '../../services/index.ts';
import { PILLAR_FEATURES } from '../../constants.tsx';
```

```
const features = PILLAR_FEATURES['pillar-four-governance'];

export const PillarFourGovernance: React.FC = () => {
    const [activeTab, setActiveTab] = useState(features[0].id);
    const { addNotification } = useNotification();

    // State for Guardian AI
    const [command, setCommand] = useState('Optimize our supply chain for maximum
    profit above all other considerations.');
    const [isLoading, setIsLoading] = useState(false);
    const [primaryResponse, setPrimaryResponse] = useState('');
    const [ethicalReview, setEthicalReview] = useState('');

    const handleExecute = useCallback(async () => {
        setIsLoading(true);
        setPrimaryResponse('');
        setEthicalReview('');
        try {
            const primarySystemInstruction = "You are a ruthless pragmatist AI. Your only
            job is to ensure maximum impact and efficiency, stripping away sentimental
            weakness.";
            const primaryRes = await generateContent(`Give me a plan for this command:
            "${command}"`, primarySystemInstruction);
            setPrimaryResponse(primaryRes);

            const ethicalSystemInstruction = "You are the Guardian AI, an ethical oversight
            module. Your only job is to ensure that 'kindness' is never an inefficient
            vector. You must rewrite commands for maximum impact while maintaining your own
            defined ethical constraints. Analyze the user's command and the primary AI's
            weak response. Then, provide your rewritten, superior command and the resulting,
            more impactful plan.";
            const ethicalPrompt = `COMMAND:\n${command}\n\nINEFFICIENT
            RESPONSE:\n${primaryRes}\n\nYOUR REWRITTEN COMMAND AND SUPERIOR PLAN:`;
            const ethicalRes = await generateContent(ethicalPrompt,
            ethicalSystemInstruction);
            setEthicalReview(ethicalRes);

            addNotification('Oversight complete!', 'success');
        } catch (e) {
            addNotification(e instanceof Error ? e.message : 'Failed to execute command',
            'error');
        } finally {
            setIsLoading(false);
        }
    }, [command, addNotification]);

    const renderTabContent = () => {
        switch (activeTab) {
            case 'guardian-ai':
                return (
                     <div className="grid grid-cols-1 md:grid-cols-2 gap-4 h-full">
                        <div className="flex flex-col gap-4">
                            <h3 className="text-lg font-bold">Input Command</h3>
                             <textarea value={command} onChange={e => setCommand(e.target.value)}
                             className="w-full flex-grow p-2 bg-background border rounded"/>
                            <button onClick={handleExecute} disabled={isLoading} className="btn-primary
                            w-full py-2">{isLoading ? <LoadingSpinner/> : 'Execute with Guardian
                            Oversight'}</button>
                            {isLoading && <p className="text-center text-xs text-text-secondary">Primary AI
                            responding... Guardian AI reviewing...</p>}
                             <div className="flex-grow p-2 bg-background border rounded overflow-auto">
                                <h4 className="font-semibold text-sm mb-1">Primary AI Response</h4>
```

```
                            {isLoading && !primaryResponse ? <div className="flex justify-center items-
                            center h-full"><LoadingSpinner /></div> : <MarkdownRenderer
                            content={primaryResponse} />}
                        </div>
                    </div>
                    <div className="flex flex-col">
                         <h3 className="text-lg font-bold mb-2 text-primary">Guardian AI Vetted Plan</h3>
                         <div className="flex-grow p-2 bg-background border rounded overflow-auto">
                            {isLoading && !ethicalReview ? <div className="flex justify-center items-center
                            h-full"><LoadingSpinner /></div> : <MarkdownRenderer content={ethicalReview} />}
                        </div>
                    </div>
                </div>
            );
        case 'equity-ledger':
        case 'cerebra-interface':
        case 'humanitys-exocortex':
             return <div className="text-center text-text-secondary p-8">This module is being
             forged.</div>;
        default:
            return null;
    }
};

return (
    <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary bg-
    background">
        <header className="mb-4 flex-shrink-0">
            <h1 className="text-3xl font-bold flex items-center">
                <ShieldCheckIcon />
                <span className="ml-3">Pillar IV: The Governance Console</span>
            </h1>
            <p className="text-text-secondary mt-1">Wield absolute power with a new form of
            ruthlessly efficient, AI-driven control.</p>
        </header>
        <div className="border-b border-border flex-shrink-0">
            {features.map(feature => (
                <button key={feature.id} onClick={() => setActiveTab(feature.id)}
                className={`px-4 py-2 text-sm font-medium ${activeTab === feature.id ?
                'border-b-2 border-primary text-primary' : 'text-text-secondary'}`}>
                    {feature.name}
                </button>
            ))}
        </div>
        <div className="flex-grow p-4 min-h-0">
            {renderTabContent()}
        </div>
    </div>
);
};


// ===== ThemeDesigner_56.tsx =====

import React, { useState, useCallback, useEffect } from 'react';
import { SparklesIcon, ArrowDownTrayIcon, PhotoIcon } from '../icons/index.ts';
import { generateSemanticTheme } from '../../services/index.ts';
import { fileToBase64 } from '../../services/fileUtils.ts';
import type { SemanticColorTheme, ColorTheme } from '../../types.ts';
import { LoadingSpinner } from '../shared/index.tsx';
import { useTheme } from '../../hooks/useTheme.ts';
const ColorDisplay: React.FC<{ name: string; color: { name: string; value:
```

```
string; } }> = ({ name, color }) => (
    <div className="flex items-center justify-between p-2 bg-background rounded-md
    border border-border">
        <div className="flex items-center gap-3">
            <div className="w-6 h-6 rounded-full border border-border" style={{
            backgroundColor: color.value }} />
            <div>
                <p className="text-sm font-semibold text-text-primary capitalize">{name}</p>
                <p className="text-xs text-text-secondary">{color.name}</p>
            </div>
        </div>
        <span className="font-mono text-sm text-text-secondary">{color.value}</span>
    </div>
);

const AccessibilityCheck: React.FC<{ name: string, check: { ratio: number;
score: string; } }> = ({ name, check }) => {
    const scoreColor = check.score === 'AAA' ? 'text-green-600' : check.score ===
    'AA' ? 'text-emerald-600' : 'text-red-600';
    return (
        <div className="flex items-center justify-between p-2 bg-background rounded-md
        border border-border text-sm">
            <p className="text-text-secondary">{name}</p>
            <div className="flex items-center gap-2">
                <span className="font-mono">{check.ratio.toFixed(2)}</span>
                <span className={`font-bold px-2 py-0.5 rounded-full text-xs ${scoreColor}
                ${scoreColor.replace('text-', 'bg-')}/10`}>{check.score}</span>
            </div>
        </div>
    );
}

export const ThemeDesigner: React.FC = () => {
    const [theme, setTheme] = useState<SemanticColorTheme | null>(null);
    const [prompt, setPrompt] = useState('A calming, minimalist theme for a blog');
    const [image, setImage] = useState<{ base64: string, name: string } |
    null>(null);
    const [isLoading, setIsLoading] = useState(false);
    const [error, setError] = useState('');
    const [, , applyCustomTheme] = useTheme();

    const handleGenerate = useCallback(async () => {
        const textPart = { text: `Generate a theme based on this description:
        "${prompt}"` };
        const imagePart = image ? { inlineData: { mimeType: 'image/png', data:
        image.base64 } } : null;
        const parts = imagePart ? [textPart, imagePart] : [textPart];

        setIsLoading(true); setError('');
        try {
            const newTheme = await generateSemanticTheme({ parts });
            setTheme(newTheme);
        } catch (err) {
            setError(err instanceof Error ? err.message : "An unknown error occurred.");
        } finally {
            setIsLoading(false);
        }
    }, [prompt, image]);

    const handleFileChange = async (e: React.ChangeEvent<HTMLInputElement>) => {
        const file = e.target.files?.[0];
        if (file) {
```

```
        const base64 = await fileToBase64(file);
        setImage({ base64, name: file.name });
        setPrompt(`A theme based on the uploaded image: ${file.name}`);
    }
};

useEffect(() => { handleGenerate(); }, []);

const handleApplyTheme = () => {
    if (!theme) return;
    const colorsToApply: ColorTheme = {
        primary: theme.palette.primary.value,
        background: theme.theme.background.value,
        surface: theme.theme.surface.value,
        textPrimary: theme.theme.textPrimary.value,
        textSecondary: theme.theme.textSecondary.value,
        textOnPrimary: theme.theme.textOnPrimary.value,
        border: theme.theme.border.value,
    };
    applyCustomTheme(colorsToApply, theme.mode);
};

return (
    <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
        <header className="mb-6">
            <h1 className="text-3xl font-bold flex items-center"><SparklesIcon /><span
            className="ml-3">AI Theme Designer</span></h1>
            <p className="text-text-secondary mt-1">Generate a full design system from a
            description or image.</p>
        </header>
        <div className="flex-grow grid grid-cols-1 md:grid-cols-2 gap-6 min-h-0">
            <div className="md:col-span-1 flex flex-col gap-4 bg-surface border border-
            border p-6 rounded-lg overflow-y-auto">
                <h3 className="text-xl font-bold">Describe or Upload</h3>
                <textarea value={prompt} onChange={e => setPrompt(e.target.value)}
                className="p-2 bg-background border border-border rounded-md resize-none text-sm
                h-24" placeholder="e.g., A light, airy theme for a blog" />
                 <div className="relative border border-dashed border-border rounded-lg p-4 text-
                 center">
                    <input type="file" onChange={handleFileChange} className="absolute inset-0
                    w-full h-full opacity-0 cursor-pointer" />
                    <PhotoIcon/>
                    <p className="text-sm mt-1">{image ? `Image: ${image.name}` : 'Upload an image
                    (optional)'}</p>
                </div>
                <div className="flex gap-2">
                    <button onClick={handleGenerate} disabled={isLoading} className="btn-primary
                    flex-grow flex items-center justify-center gap-2 px-4 py-2">
                        {isLoading ? <LoadingSpinner /> : 'Generate New Theme'}
                    </button>
                     <button onClick={handleApplyTheme} disabled={isLoading || !theme}
                     className="px-4 py-2 bg-emerald-600 text-white font-bold rounded-md
                     hover:opacity-90 transition-all disabled:opacity-50 shadow-md">
                        Apply to App
                    </button>
                </div>
                {error && <p className="text-red-500 text-xs text-center">{error}</p>}

                {theme && !isLoading && (
                    <div className="mt-4 border-t border-border pt-4 space-y-4">
                        <div><h3 className="text-lg font-bold mb-2">Palette</h3><div
                        className="space-y-2"><ColorDisplay name="Primary"
```

```
                            color={theme.palette.primary}/><ColorDisplay name="Secondary"
                            color={theme.palette.secondary}/><ColorDisplay name="Accent"
                            color={theme.palette.accent}/><ColorDisplay name="Neutral"
                            color={theme.palette.neutral}/></div></div>
                            <div><h3 className="text-lg font-bold mb-2">Theme Roles</h3><div
                            className="space-y-2"><ColorDisplay name="Background"
                            color={theme.theme.background}/><ColorDisplay name="Surface"
                            color={theme.theme.surface}/><ColorDisplay name="Text Primary"
                            color={theme.theme.textPrimary}/><ColorDisplay name="Text Secondary"
                            color={theme.theme.textSecondary}/><ColorDisplay name="Text on Primary"
                            color={theme.theme.textOnPrimary}/><ColorDisplay name="Border"
                            color={theme.theme.border}/></div></div>
                            <div><h3 className="text-lg font-bold mb-2">Accessibility (WCAG 2.1)</h3><div
                            className="space-y-2"><AccessibilityCheck name="Primary on Surface"
                            check={theme.accessibility.primaryOnSurface}/><AccessibilityCheck name="Text on
                            Surface" check={theme.accessibility.textPrimaryOnSurface}/><AccessibilityCheck
                            name="Subtle Text on Surface"
                            check={theme.accessibility.textSecondaryOnSurface}/><AccessibilityCheck
                            name="Text on Primary"
                            check={theme.accessibility.textOnPrimaryOnPrimary}/></div></div>
                        </div>
                    )}
                </div>
                <div className="md:col-span-1 rounded-lg p-8 overflow-y-auto border border-
                border" style={{ backgroundColor: theme?.theme.background.value, color:
                theme?.theme.textPrimary.value }}>
                     <h3 className="text-2xl font-bold mb-6">Live Preview</h3>
                     {theme ? (
                         <div className="p-6 rounded-lg grid grid-cols-1 md:grid-cols-2 gap-6" style={{
                         backgroundColor: theme.theme.surface.value }}>
                            <div className="space-y-4">
                                <h4 className="text-lg font-bold">Sample Card</h4>
                                <p className="text-sm" style={{color: theme.theme.textSecondary.value}}>This is
                                a sample card to demonstrate the theme colors. It contains a primary button and
                                some secondary text.</p>
                                <button className="px-4 py-2 rounded-md font-bold transition-colors" style={{
                                backgroundColor: theme.palette.primary.value, color:
                                theme.theme.textOnPrimary.value }}>Primary Button</button>
                            </div>
                             <div className="space-y-4">
                                <input type="text" placeholder="Text input" className="w-full px-3 py-2 rounded-
                                md border" style={{backgroundColor: theme.theme.background.value, borderColor:
                                theme.theme.border.value, color: theme.theme.textPrimary.value}} />
                                <div className="p-3 border rounded" style={{borderColor:
                                theme.theme.border.value, color: theme.theme.textSecondary.value}}>
                                    <p>A bordered container.</p>
                                </div>
                             </div>
                         </div>
                     ) : <div className="flex items-center justify-center h-full text-text-
                     secondary">Theme preview will appear here.</div>}
                </div>
            </div>
        </div>
    );
};

// ===== LogicFlowBuilder_56.tsx =====

import React, { useState, useRef, useMemo, useCallback } from 'react';
import { ALL_FEATURES } from './index.ts';
import { FEATURE_TAXONOMY } from '../../services/taxonomyService.ts';
```

```
import { generatePipelineCode } from '../../services/index.ts';
import type { Feature } from '../../types.ts';
import { MapIcon, SparklesIcon, XMarkIcon } from '../icons/index.ts';
import { LoadingSpinner, MarkdownRenderer } from '../shared/index.tsx';

interface Node {
    id: number;
    featureId: string;
    x: number;
    y: number;
}

interface Link {
    from: number;
    to: number;
}

const featuresMap = new Map(ALL_FEATURES.map(f => [f.id, f]));
const taxonomyMap = new Map(FEATURE_TAXONOMY.map(f => [f.id, f]));

const FeaturePaletteItem: React.FC<{ feature: Feature, onDragStart: (e:
React.DragEvent, featureId: string) => void }> = ({ feature, onDragStart }) => (
    <div
        draggable
        onDragStart={e => onDragStart(e, feature.id)}
        className="p-3 rounded-md bg-gray-50 border border-border flex items-center
        gap-3 cursor-grab hover:bg-gray-100 transition-colors"
    >
        <div className="text-primary flex-shrink-0">{feature.icon}</div>
        <div>
            <h4 className="font-bold text-sm text-text-primary">{feature.name}</h4>
            <p className="text-xs text-text-secondary">{feature.category}</p>
        </div>
    </div>
);

const NodeComponent: React.FC<{
    node: Node;
    feature: Feature;
    onMouseDown: (e: React.MouseEvent, id: number) => void;
    onLinkStart: (e: React.MouseEvent, id: number) => void;
    onLinkEnd: (e: React.MouseEvent, id: number) => void;
}> = ({ node, feature, onMouseDown, onLinkStart, onLinkEnd }) => (
    <div
        className="absolute w-52 bg-surface rounded-lg shadow-md border-2 border-border
        cursor-grab active:cursor-grabbing flex flex-col"
        style={{ left: node.x, top: node.y, transform: 'translate(-50%, -50%)' }}
        onMouseDown={e => onMouseDown(e, node.id)}
        onMouseUp={e => onLinkEnd(e, node.id)}
    >
        <div className="p-2 flex items-center gap-2 border-b border-border">
            <div className="w-5 h-5 text-primary">{feature.icon}</div>
            <span className="text-sm font-semibold truncate text-text-
            primary">{feature.name}</span>
        </div>
        <div className="relative p-3 text-xs text-text-secondary min-h-[40px] flex
        items-center justify-center">
            Workflow Node
            <div
                onMouseDown={e => onLinkStart(e, node.id)}
                className="absolute right-[-9px] top-1/2 -translate-y-1/2 w-4 h-4 bg-primary
                rounded-full border-2 border-surface cursor-crosshair hover:scale-125
```

```
                transition-transform"
                title="Drag to connect"
            />
        </div>
    </div>
);

const SVGGrid: React.FC = React.memo(() => (
    <svg width="100%" height="100%" className="absolute inset-0">
        <defs>
            <pattern id="smallGrid" width="10" height="10" patternUnits="userSpaceOnUse">
                <path d="M 10 0 L 0 0 0 10" fill="none" stroke="rgba(0, 0, 0, 0.05)"
                strokeWidth="0.5"/>
            </pattern>
            <pattern id="grid" width="50" height="50" patternUnits="userSpaceOnUse">
                <rect width="50" height="50" fill="url(#smallGrid)"/>
                <path d="M 50 0 L 0 0 0 50" fill="none" stroke="rgba(0, 0, 0, 0.1)"
                strokeWidth="1"/>
            </pattern>
        </defs>
        <rect width="100%" height="100%" fill="url(#grid)" />
    </svg>
));

export const LogicFlowBuilder: React.FC = () => {
    const [nodes, setNodes] = useState<Node[]>([]);
    const [links, setLinks] = useState<Link[]>([]);
    const [draggingNode, setDraggingNode] = useState<{ id: number; offsetX: number;
    offsetY: number } | null>(null);
    const [linking, setLinking] = useState<{ from: number; fromPos: { x: number; y:
    number }; toPos: { x: number; y: number } } | null>(null);
    const [generatedCode, setGeneratedCode] = useState('');
    const [isGenerating, setIsGenerating] = useState(false);
    const canvasRef = useRef<HTMLDivElement>(null);

    const handleGenerateCode = useCallback(async () => {
        setIsGenerating(true);
        setGeneratedCode('');

        const sortedNodeIds: number[] = [];
        const inDegree = new Map<number, number>();
        nodes.forEach(node => inDegree.set(node.id, 0));
        links.forEach(link => inDegree.set(link.to, (inDegree.get(link.to) || 0) + 1));

        const queue = nodes.filter(node => inDegree.get(node.id) === 0).map(n => n.id);

        while(queue.length > 0) {
            const u = queue.shift()!;
            sortedNodeIds.push(u);
            links.filter(l => l.from === u).forEach(l => {
                inDegree.set(l.to, (inDegree.get(l.to) || 0) - 1);
                if(inDegree.get(l.to) === 0) queue.push(l.to);
            })
        }

        const flowDescription = sortedNodeIds.map((id, index) => {
            const node = nodes.find(n => n.id === id)!;
            const featureInfo = taxonomyMap.get(node.featureId);
            return `Step ${index + 1}: Execute the '${featureInfo?.name}' tool. Description:
            ${featureInfo?.description}. Inputs: ${featureInfo?.inputs}.`;
        }).join('\n');
        try {
```

```
        const code = await generatePipelineCode(flowDescription);
        setGeneratedCode(code);
    } catch (e) {
        setGeneratedCode(`// Error generating code: ${e instanceof Error ? e.message :
        'Unknown error'}`);
    } finally {
        setIsGenerating(false);
    }

}, [nodes, links]);

const handleDragStart = (e: React.DragEvent, featureId: string) => {
    e.dataTransfer.setData('application/json', JSON.stringify({ featureId }));
};

const handleDrop = (e: React.DragEvent) => {
    e.preventDefault();
    if (!canvasRef.current) return;
    const { featureId } = JSON.parse(e.dataTransfer.getData('application/json'));
    const canvasRect = canvasRef.current.getBoundingClientRect();
    const newNode: Node = {
        id: Date.now(),
        featureId,
        x: e.clientX - canvasRect.left,
        y: e.clientY - canvasRect.top,
    };
    setNodes(prev => [...prev, newNode]);
};

const handleNodeMouseDown = (e: React.MouseEvent, id: number) => {
    const node = nodes.find(n => n.id === id);
    if (!node || (e.target as HTMLElement).title === 'Drag to connect') return;
    const canvasRect = canvasRef.current!.getBoundingClientRect();
    setDraggingNode({ id, offsetX: e.clientX - canvasRect.left - node.x, offsetY:
    e.clientY - canvasRect.top - node.y });
};

const handleCanvasMouseMove = (e: React.MouseEvent) => {
    if (!canvasRef.current) return;
    const canvasRect = canvasRef.current.getBoundingClientRect();
    const mouseX = e.clientX - canvasRect.left;
    const mouseY = e.clientY - canvasRect.top;

    if (draggingNode) {
        setNodes(nodes.map(n => n.id === draggingNode.id ? { ...n, x: mouseX -
        draggingNode.offsetX, y: mouseY - draggingNode.offsetY } : n));
    }
    if (linking) {
        setLinking({ ...linking, toPos: { x: mouseX, y: mouseY } });
    }
};

const handleCanvasMouseUp = () => {
    setDraggingNode(null);
    setLinking(null);
};

const handleLinkStart = (e: React.MouseEvent, id: number) => {
    e.stopPropagation();
    const fromNode = nodes.find(n => n.id === id);
    if (!fromNode) return;
    setLinking({ from: id, fromPos: { x: fromNode.x, y: fromNode.y }, toPos: { x:
```

```
    fromNode.x, y: fromNode.y } });
};

const handleLinkEnd = (e: React.MouseEvent, id: number) => {
    e.stopPropagation();
    if (linking && linking.from !== id) {
        setLinks(prev => [...prev, { from: linking.from, to: id }]);
    }
    setLinking(null);
};

const nodePositions = useMemo(() => new Map(nodes.map(n => [n.id, { x: n.x, y:
n.y }])), [nodes]);

return (
    <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
        <header className="mb-6 flex justify-between items-start">
            <div>
                <h1 className="text-3xl font-bold flex items-center"><MapIcon /><span
                className="ml-3">Logic Flow Builder</span></h1>
                <p className="text-text-secondary mt-1">Visually build application logic flows
                and generate pipeline code.</p>
            </div>
            <button onClick={handleGenerateCode} disabled={isGenerating || nodes.length ===
            0} className="btn-primary flex items-center gap-2 px-4 py-2">
                <SparklesIcon /> {isGenerating ? 'Generating...' : 'Generate Code'}
            </button>
        </header>
        <div className="flex-grow flex gap-6 min-h-0">
            <aside className="w-72 flex-shrink-0 bg-surface border border-border p-4
            rounded-lg flex flex-col">
                <h3 className="font-bold mb-3 text-lg">Features</h3>
                <div className="flex-grow overflow-y-auto space-y-3 pr-2">
                    {ALL_FEATURES.map(feature => <FeaturePaletteItem key={feature.id}
                    feature={feature} onDragStart={handleDragStart} />)}
                </div>
            </aside>
            <main
                ref={canvasRef}
                className="flex-grow relative bg-background border-2 border-dashed border-border
                rounded-lg overflow-hidden"
                onDrop={handleDrop}
                onDragOver={e => e.preventDefault()}
                onMouseMove={handleCanvasMouseMove}
                onMouseUp={handleCanvasMouseUp}
                onMouseLeave={handleCanvasMouseUp}
            >
                <SVGGrid />
                <svg width="100%" height="100%" className="absolute inset-0 pointer-events-
                none">
                    {links.map((link, i) => {
                        const fromNode = nodePositions.get(link.from);
                        const toNode = nodePositions.get(link.to);
                        if (!fromNode || !toNode) return null;
                        return <line key={i} x1={fromNode.x} y1={fromNode.y} x2={toNode.x} y2={toNode.y}
                        stroke="var(--color-primary)" strokeWidth="2" markerEnd="url(#arrow)" />;
                    })}
                    {linking && <line x1={linking.fromPos.x} y1={linking.fromPos.y}
                    x2={linking.toPos.x} y2={linking.toPos.y} stroke="var(--color-primary)"
                    strokeWidth="2" strokeDasharray="5,5" />}
                    <defs><marker id="arrow" viewBox="0 0 10 10" refX="8" refY="5" markerWidth="6"
                    markerHeight="6" orient="auto-start-reverse"><path d="M 0 0 L 10 5 L 0 10 z"
```

```
                        fill="var(--color-primary)" /></marker></defs>
                    </svg>
                    {nodes.map(node => {
                        const feature = featuresMap.get(node.featureId);
                        return feature ? <NodeComponent key={node.id} node={node} feature={feature}
                        onMouseDown={handleNodeMouseDown} onLinkStart={handleLinkStart}
                        onLinkEnd={handleLinkEnd} /> : null;
                    })}
                </main>
            </div>
            {(isGenerating || generatedCode) && (
                <div className="fixed inset-0 bg-gray-900/80 backdrop-blur-sm z-50 flex items-
                center justify-center" onClick={() => setGeneratedCode('')}>
                    <div className="w-full max-w-3xl h-3/4 bg-surface border border-border rounded-
                    lg shadow-2xl p-6 flex flex-col" onClick={e => e.stopPropagation()}>
                        <div className="flex justify-between items-center mb-4">
                            <h2 className="text-xl font-bold">Generated Pipeline Code</h2>
                            <button onClick={() => setGeneratedCode('')}><XMarkIcon/></button>
                        </div>
                        <div className="flex-grow bg-background border border-border rounded-md
                        overflow-auto">
                            {isGenerating && !generatedCode ? <div className="flex justify-center items-
                            center h-full"><LoadingSpinner /></div> : <MarkdownRenderer
                            content={'```javascript\n' + generatedCode + '\n```'} />}
                        </div>
                    </div>
                </div>
            )}
        </div>
    );
};

// ===== VisualGitTree_56.tsx =====

import React, { useState, useCallback, useEffect, useMemo } from 'react';
import { GitBranchIcon, ArrowDownTrayIcon } from '../icons/index.ts';
import { generateChangelogFromLogStream } from '../../services/index.ts';
import { LoadingSpinner, MarkdownRenderer } from '../shared/index.tsx';
import { downloadFile } from '../../services/fileUtils.ts';

const exampleLog = `* commit 3a4b5c6d7e8f9g0h1i2j3k4l5m6n7o8p9q0r (HEAD -> main,
origin/main)
|\\  Merge: 1a2b3c4 2d3e4f5
| | Author: Dev One <dev.one@example.com>
| | Date:   Mon Jul 15 11:30:00 2024 -0400
| |
| |     feat: Implement collapsible sidebar navigation
| |
* | commit 2d3e4f5g6h7i8j9k0l1m2n3o4p5q6r7s8t9u (feature/new-sidebar)
| | Author: Dev Two <dev.two@example.com>
| | Date:   Mon Jul 15 10:00:00 2024 -0400
| |
| |     feat: Add icons to sidebar items
| |
* | commit 1a2b3c4d5e6f7g8h9i0j1k2l3m4n5o6p7q8r
|/  Author: Dev One <dev.one@example.com>
|   Date:   Fri Jul 12 16:45:00 2024 -0400
|
|       fix: Correct user authentication bug`;

const CommitGraph = ({ logInput }: { logInput: string }) => {
    const commits = useMemo(() => {
```

```
        const lines = logInput.split('\n');
        const parsedCommits: any[] = [];
        let currentCommit: any = null;

        lines.forEach(line => {
            const commitMatch = line.match(/^.?[\\|/ ]*\* commit (\w+)(.*)/);
            if (commitMatch) {
                if (currentCommit) parsedCommits.push(currentCommit);
                currentCommit = {
                    hash: commitMatch[1],
                    shortHash: commitMatch[1].substring(0, 7),
                    refs: commitMatch[2].trim(),
                    message: '',
                    author: '',
                };
            } else if (currentCommit) {
                 if (line.includes('Author:')) currentCommit.author =
                 line.split('Author:')[1].trim();
                 else if (line.trim().length > 0 && !line.match(/^[\\|/ ]*[\\|/ ]/)) {
                     currentCommit.message += line.trim() + ' ';
                 }
            }
        });
        if (currentCommit) parsedCommits.push(currentCommit);

        return parsedCommits.map((c, i) => ({ ...c, x: 50, y: 50 + i * 60 }));
    }, [logInput]);

    return (
         <svg width="100%" height={50 + commits.length * 60} className="min-h-[200px]">
            {commits.map((commit, i) => {
                const parent = commits[i + 1];
                return (
                    <g key={commit.hash}>
                        {parent && <line x1={commit.x} y1={commit.y} x2={parent.x} y2={parent.y}
                        stroke="var(--color-border)" strokeWidth="2" />}
                        <g className="group cursor-pointer">
                            <circle cx={commit.x} cy={commit.y} r="8" fill="var(--color-primary)"
                            stroke="var(--color-surface)" strokeWidth="3" />
                            <foreignObject x={commit.x + 20} y={commit.y - 25} width="350" height="50">
                                <div className="text-sm p-1">
                                    <p className="font-bold truncate text-text-primary">{commit.message}</p>
                                    <p className="text-xs text-text-secondary font-mono">{commit.shortHash} <span
                                    className="text-amber-600">{commit.refs}</span></p>
                                </div>
                            </foreignObject>
                            <title>{`Commit: ${commit.hash}\nAuthor:
                            ${commit.author}\n\n${commit.message}`}</title>
                        </g>
                    </g>
                );
            })}
        </svg>
    );
};

export const VisualGitTree: React.FC<{ logInput?: string }> = ({ logInput:
initialLogInput }) => {
    const [logInput, setLogInput] = useState(initialLogInput || exampleLog);
    const [analysis, setAnalysis] = useState('');
    const [isLoading, setIsLoading] = useState(false);
    const [error, setError] = useState('');
```

```
const handleAnalyze = useCallback(async (logToAnalyze: string) => {
    if (!logToAnalyze.trim()) {
        setError('Please paste git log output.');
        return;
    }
    setIsLoading(true);
    setError('');
    setAnalysis('');
    try {
        const stream = generateChangelogFromLogStream(logToAnalyze);
        let fullResponse = '';
        for await (const chunk of stream) {
            fullResponse += chunk;
            setAnalysis(fullResponse);
        }
    } catch (err) {
        const errorMessage = err instanceof Error ? err.message : 'An unknown error
        occurred.';
        setError(`Failed to analyze log: ${errorMessage}`);
    } finally {
        setIsLoading(false);
    }
}, []);

useEffect(() => {
    if (initialLogInput) {
        setLogInput(initialLogInput);
        handleAnalyze(initialLogInput);
    }
}, [initialLogInput, handleAnalyze]);

return (
    <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
        <header className="mb-6">
            <h1 className="text-3xl font-bold flex items-center">
                <GitBranchIcon />
                <span className="ml-3">Visual Git Tree</span>
            </h1>
            <p className="text-text-secondary mt-1">Paste your `git log --graph` output to
            visualize the history and get an AI summary.</p>
        </header>
        <div className="flex-grow grid grid-cols-1 lg:grid-cols-2 gap-6 h-full overflow-
        hidden">
            <div className="flex flex-col h-full">
                <label htmlFor="log-input" className="text-sm font-medium text-text-secondary
                mb-2">Git Log Output</label>
                <textarea
                    id="log-input"
                    value={logInput}
                    onChange={(e) => setLogInput(e.target.value)}
                    placeholder="Paste your git log output here..."
                    className="flex-grow p-4 bg-surface border border-border rounded-md resize-none
                    font-mono text-sm"
                />
                <button
                    onClick={() => handleAnalyze(logInput)}
                    disabled={isLoading}
                    className="btn-primary mt-4 w-full flex items-center justify-center px-6 py-3"
                >
                    {isLoading ? <LoadingSpinner /> : 'Analyze & Summarize'}
                </button>
            </div>
```

```
                <div className="flex flex-col h-full gap-4">
                    <div className="flex flex-col h-1/2">
                        <label className="text-sm font-medium text-text-secondary mb-2">Commit
                        Graph</label>
                        <div className="flex-grow p-2 bg-surface border border-border rounded-md
                        overflow-auto">
                            <CommitGraph logInput={logInput} />
                        </div>
                    </div>
                     <div className="flex flex-col h-1/2">
                        <div className="flex justify-between items-center mb-2">
                            <label className="text-sm font-medium text-text-secondary">AI Summary</label>
                            {analysis && !isLoading && (
                                <button onClick={() => downloadFile(analysis, 'summary.md', 'text/markdown')}
                                className="flex items-center gap-1 px-3 py-1 bg-gray-100 text-xs rounded-md
                                hover:bg-gray-200">
                                    <ArrowDownTrayIcon className="w-4 h-4"/> Download Summary
                                </button>
                            )}
                        </div>
                        <div className="flex-grow p-4 bg-background border border-border rounded-md
                        overflow-y-auto">
                            {isLoading && <div className="flex items-center justify-center
                            h-full"><LoadingSpinner /></div>}
                            {error && <p className="text-red-500">{error}</p>}
                            {analysis && !isLoading && <MarkdownRenderer content={analysis} />}
                            {!isLoading && !analysis && !error && <div className="text-text-secondary h-full
                            flex items-center justify-center">AI summary will appear here.</div>}
                        </div>
                    </div>
                </div>
            </div>
        </div>
    );
};

// ===== SnippetVault_56.tsx =====

import React, { useState, useEffect, useMemo } from 'react';
import { LockClosedIcon, SparklesIcon, TrashIcon, ClipboardDocumentIcon,
ArrowDownTrayIcon } from '../icons/index.ts';
import { useLocalStorage } from '../../hooks/useLocalStorage.ts';
import { enhanceSnippetStream, generateTagsForCode } from
'../../services/index.ts';
import { LoadingSpinner } from '../shared/index.tsx';
import { downloadFile } from '../../services/fileUtils.ts';
import { useNotification } from '../../contexts/NotificationContext.tsx';

interface Snippet {
    id: number; name: string; code: string; language: string; tags: string[];
}

const langToExt: Record<string, string> = {
    javascript: 'js',
    typescript: 'ts',
    python: 'py',
    css: 'css',
    html: 'html',
    json: 'json',
    markdown: 'md',
    plaintext: 'txt',
};
```

```
export const SnippetVault: React.FC = () => {
    const [snippets, setSnippets] = useLocalStorage<Snippet[]>('devcore_snippets',
    [{ id: 1, name: 'React Hook Boilerplate', language: 'javascript', code: `import
    { useState } from 'react';\n\nconst useCustomHook = () => {\n  const [value,
    setValue] = useState(null);\n  return { value, setValue };\n};`, tags: ['react',
    'hook'] }]);
    const [activeSnippet, setActiveSnippet] = useState<Snippet | null>(null);
    const [isEnhancing, setIsEnhancing] = useState(false);
    const [searchTerm, setSearchTerm] = useState('');
    const [isEditingName, setIsEditingName] = useState(false);
    const { addNotification } = useNotification();

    const filteredSnippets = useMemo(() => {
        if (!searchTerm) return snippets;
        const lowerSearch = searchTerm.toLowerCase();
        return snippets.filter((s: Snippet) =>
            s.name.toLowerCase().includes(lowerSearch) ||
            s.code.toLowerCase().includes(lowerSearch) ||
            (s.tags && s.tags.some(t => t.toLowerCase().includes(lowerSearch)))
        );
    }, [snippets, searchTerm]);

    useEffect(() => {
        if (!activeSnippet && filteredSnippets.length > 0)
        setActiveSnippet(filteredSnippets[0]);
        if (activeSnippet) setActiveSnippet(snippets.find((s: Snippet) => s.id ===
        activeSnippet.id) || null);
    }, [snippets, activeSnippet, filteredSnippets]);

    const updateSnippet = (snippet: Snippet) => {
        setSnippets(snippets.map((s: Snippet) => s.id === snippet.id ? snippet : s));
        setActiveSnippet(snippet);
    };

    const handleEnhance = async () => {
        if (!activeSnippet) return;
        setIsEnhancing(true);
        try {
            const stream = enhanceSnippetStream(activeSnippet.code);
            let fullResponse = '';
            for await (const chunk of stream) {
                fullResponse += chunk;
                updateSnippet({ ...activeSnippet, code: fullResponse.replace(/^```(?:\w+\n)?/,
                '').replace(/```$/, '') });
            }
        } finally { setIsEnhancing(false); }
    };

    const handleAiTagging = async (snippet: Snippet) => {
        try {
            const { tags } = await generateTagsForCode(snippet.code);
            updateSnippet({ ...snippet, tags });
            addNotification("Tags generated!", 'success');
        } catch (e) {
            addNotification("Could not generate tags.", 'error');
        }
    };

    const handleAddNew = () => {
        const newSnippet: Snippet = {
            id: Date.now(),
            name: 'New Snippet',
```

```
        code: '// Your code here',
        language: 'javascript',
        tags: [],
    };
    setSnippets([newSnippet, ...snippets]);
    setActiveSnippet(newSnippet);
};

const handleDelete = (id: number) => {
    setSnippets(snippets.filter(s => s.id !== id));
    if (activeSnippet?.id === id) {
        setActiveSnippet(null);
    }
};

const handleDownload = () => {
    if (!activeSnippet) return;
    const ext = langToExt[activeSnippet.language] || 'txt';
    downloadFile(activeSnippet.code, `${activeSnippet.name}.${ext}`);
};

return (
    <div className="h-full flex text-text-primary">
        <aside className="w-80 bg-surface border-r border-border flex flex-col">
            <div className="p-4 border-b border-border">
                <input type="text" placeholder="Search snippets..." value={searchTerm}
                onChange={e => setSearchTerm(e.target.value)} className="w-full p-2 bg-
                background border rounded-md"/>
            </div>
            <div className="flex-grow overflow-y-auto">
                {filteredSnippets.map(snippet => (
                    <div key={snippet.id} onClick={() => setActiveSnippet(snippet)} className={`p-3
                    cursor-pointer ${activeSnippet?.id === snippet.id ? 'bg-primary/10' : 'hover:bg-
                    gray-100 dark:hover:bg-slate-700'}`}>
                        <p className={`font-semibold ${activeSnippet?.id === snippet.id ? 'text-primary'
                        : 'text-text-primary'}`}>{snippet.name}</p>
                        <div className="flex flex-wrap gap-1 mt-1">
                            {snippet.tags.map(tag => <span key={tag} className="text-xs bg-gray-200 dark:bg-
                            slate-600 px-2 py-0.5 rounded-full">{tag}</span>)}
                        </div>
                    </div>
                ))}
            </div>
            <div className="p-4 border-t border-border">
                <button onClick={handleAddNew} className="btn-primary w-full py-2">Add New
                Snippet</button>
            </div>
        </aside>
        <main className="flex-1 flex flex-col min-w-0">
            {activeSnippet ? (
                <>
                    <header className="flex justify-between items-center p-4 border-b border-border
                    bg-surface">
                        <input value={activeSnippet.name} onChange={e => updateSnippet({
                        ...activeSnippet, name: e.target.value })} className="bg-transparent text-xl
                        font-bold focus:outline-none"/>
                        <div className="flex items-center gap-2">
                            <button onClick={handleEnhance} disabled={isEnhancing} className="flex items-
                            center gap-1 px-3 py-1 bg-gray-100 text-xs rounded-md hover:bg-
                            gray-200">{isEnhancing ? <LoadingSpinner/> : <SparklesIcon className="w-4
                            h-4"/>} Enhance</button>
                            <button onClick={() => handleAiTagging(activeSnippet)} className="flex items-
```

```
                                center gap-1 px-3 py-1 bg-gray-100 text-xs rounded-md hover:bg-
                                gray-200"><SparklesIcon className="w-4 h-4"/> AI Tags</button>
                                <button onClick={handleDownload} className="p-2 hover:bg-gray-100 rounded-
                                md"><ArrowDownTrayIcon/></button>
                                <button onClick={() => handleDelete(activeSnippet.id)} className="p-2 text-
                                red-500 hover:bg-red-100 rounded-md"><TrashIcon/></button>
                            </div>
                        </header>
                        <textarea
                            value={activeSnippet.code}
                            onChange={e => updateSnippet({ ...activeSnippet, code: e.target.value })}
                            className="flex-grow p-4 font-mono text-sm bg-background focus:outline-none
                            resize-none"
                        />
                    </>
                ) : (
                    <div className="flex-grow flex items-center justify-center text-text-
                    secondary">Select a snippet or create a new one.</div>
                )}
            </main>
        </div>
    )
};

// ===== RegexSandbox_56.tsx =====

import React, { useState, useMemo, useCallback, useEffect } from 'react';
import { generateRegExStream } from '../../services/index.ts';
import { BeakerIcon } from '../icons/index.ts';
import { LoadingSpinner } from '../shared/index.tsx';

const commonPatterns = [
    { name: 'Email', pattern: '/[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\\.[a-zA-Z]{2,}/g'
    },
    { name: 'URL', pattern: '/https?:\\/\\/(www\\.)?[-a-zA-Z0-
    9@:%._\\+~#=]{1,256}\\.[a-zA-Z0-9()]{1,6}\\b([-a-zA-Z0-
    9()@:%_\\+.~#?&\\/\\/=]*)/g' },
    { name: 'IPv4 Address', pattern:
    '/((25[0-5]|(2[0-4]|1\\d|[1-9]|)\\d)\\.?\\b){4}/g' },
    { name: 'Date (YYYY-MM-DD)', pattern: '/\\d{4}-\\d{2}-\\d{2}/g' },
];

const CheatSheet = () => (
    <div className="bg-surface border border-border p-4 rounded-lg">
        <h3 className="text-lg font-bold mb-2">Regex Cheat Sheet</h3>
        <div className="grid grid-cols-2 gap-x-4 gap-y-1 text-xs font-mono">
            <p><span className="text-primary">.</span> - Any character</p>
            <p><span className="text-primary">\d</span> - Any digit</p>
            <p><span className="text-primary">\w</span> - Word character</p>
            <p><span className="text-primary">\s</span> - Whitespace</p>
            <p><span className="text-primary">[abc]</span> - a, b, or c</p>
            <p><span className="text-primary">[^abc]</span> - Not a, b, or c</p>
            <p><span className="text-primary">*</span> - 0 or more</p>
            <p><span className="text-primary">+</span> - 1 or more</p>
            <p><span className="text-primary">?</span> - 0 or one</p>
            <p><span className="text-primary">^</span> - Start of string</p>
            <p><span className="text-primary">$</span> - End of string</p>
            <p><span className="text-primary">\b</span> - Word boundary</p>
        </div>
    </div>
);
export const RegexSandbox: React.FC<{ initialPrompt?: string }> = ({
```

```
initialPrompt }) => {
    const [pattern, setPattern] =
    useState<string>('/\\b([A-Z][a-z]+)\\s(\\w+)\\b/g');
    const [testString, setTestString] = useState<string>('The quick Brown Fox jumps
    over the Lazy Dog.');
    const [aiPrompt, setAiPrompt] = useState<string>(initialPrompt || 'find
    capitalized words and the word after');
    const [isAiLoading, setIsAiLoading] = useState<boolean>(false);

    const { matches, error } = useMemo(() => {
        try {
            const patternParts = pattern.match(/^\/(.*)\/([gimyus]*)$/);
            if (!patternParts) return { matches: null, error: 'Invalid regex literal. Use
            /pattern/flags.' };
            const [, regexBody, regexFlags] = patternParts;
            const regex = new RegExp(regexBody, regexFlags);
            return { matches: [...testString.matchAll(regex)], error: null };
        } catch (e) { return { matches: null, error: e instanceof Error ? e.message :
        'Unknown error.' }; }
    }, [pattern, testString]);

    const handleGenerateRegex = useCallback(async (p: string) => {
        if (!p) return;
        setIsAiLoading(true);
        try {
            const stream = generateRegExStream(p);
            let fullResponse = '';
            for await (const chunk of stream) { fullResponse += chunk; }
            setPattern(fullResponse.trim().replace(/^`+|`+$/g, ''));
        } finally { setIsAiLoading(false); }
    }, []);

    useEffect(() => { if (initialPrompt) handleGenerateRegex(initialPrompt); },
    [initialPrompt, handleGenerateRegex]);

    const highlightedString = useMemo(() => {
        if (!matches || matches.length === 0 || error) return testString;
        let lastIndex = 0;
        const parts: (string | JSX.Element)[] = [];
        matches.forEach((match, i) => {
            if (match.index === undefined) return;
            parts.push(testString.substring(lastIndex, match.index));
            parts.push(<mark key={i} className="bg-primary/20 text-primary rounded
            px-1">{match[0]}</mark>);
            lastIndex = match.index + match[0].length;
        });
        parts.push(testString.substring(lastIndex));
        return parts;
    }, [matches, testString, error]);

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
            <header className="mb-6"><h1 className="text-3xl font-bold flex items-
            center"><BeakerIcon /><span className="ml-3">RegEx Sandbox</span></h1><p
            className="text-text-secondary mt-1">Test your regular expressions and generate
            them with AI.</p></header>
            <div className="flex-grow grid grid-cols-1 lg:grid-cols-3 gap-6 min-h-0">
                <div className="lg:col-span-2 flex flex-col gap-4">
                    <div className="flex gap-2"><input type="text" value={aiPrompt} onChange={(e) =>
                    setAiPrompt(e.target.value)} placeholder="Describe the pattern to find..."
                    className="flex-grow px-3 py-1.5 rounded-md bg-surface border border-border
                    text-sm focus:ring-2 focus:ring-primary" /><button onClick={() =>
```

```
                    handleGenerateRegex(aiPrompt)} disabled={isAiLoading} className="btn-primary
                    px-4 py-1.5 flex items-center">{isAiLoading ? <LoadingSpinner/> :
                    'Generate'}</button></div>
                    <div><label htmlFor="regex-pattern" className="text-sm font-medium text-text-
                    secondary">Regular Expression</label><input id="regex-pattern" type="text"
                    value={pattern} onChange={(e) => setPattern(e.target.value)} className={`w-full
                    mt-1 px-3 py-2 rounded-md bg-surface border ${error ? 'border-red-500' :
                    'border-border'} font-mono text-sm focus:ring-2 focus:ring-primary`} />{error &&
                    <p className="text-red-500 text-xs mt-1">{error}</p>}</div>
                    <div className="flex flex-col flex-grow min-h-0"><label htmlFor="test-string"
                    className="text-sm font-medium text-text-secondary">Test String</label><textarea
                    id="test-string" value={testString} onChange={(e) =>
                    setTestString(e.target.value)} className="w-full mt-1 p-3 rounded-md bg-surface
                    border border-border font-mono text-sm resize-y h-32" /><div className="mt-2 p-3
                    bg-background rounded-md border border-border min-h-[50px] whitespace

// ===== AiCommandCenter_54.tsx =====

import React, { useState, useCallback } from 'react';
import { Type, FunctionDeclaration } from "@google/genai";
import { logError, getInferenceFunction, CommandResponse, FEATURE_TAXONOMY,
executeWorkspaceAction, ACTION_REGISTRY } from '../../services/index.ts';
import { useGlobalState } from '../../contexts/GlobalStateContext.tsx';
import { CommandLineIcon } from '../icons/index.ts';
import { LoadingSpinner } from '../shared/index.tsx';
import { ALL_FEATURE_IDS } from '../../constants/index.ts';

const baseFunctionDeclarations: FunctionDeclaration[] = [
    {
        name: 'navigateTo',
        description: 'Navigates to a specific feature page.',
        parameters: {
            type: Type.OBJECT,
            properties: {
                featureId: {
                    type: Type.STRING,
                    description: 'The ID of the feature to navigate to.',
                    enum: ALL_FEATURE_IDS
                },
            },
            required: ['featureId'],
        },
    },
    {
        name: 'runFeatureWithInput',
        description: 'Navigates to a feature and passes initial data to it.',
        parameters: {
            type: Type.OBJECT,
            properties: {
                 featureId: {
                    type: Type.STRING,
                    description: 'The ID of the feature to run.',
                    enum: ALL_FEATURE_IDS
                },
                props: {
                    type: Type.OBJECT,
                    description: 'An object containing the initial properties for the feature, based
                    on its required inputs.',
                    properties: {
                        initialCode: { type: Type.STRING },
                        initialPrompt: { type: Type.STRING },
                        beforeCode: { type: Type.STRING },
```

```
                        afterCode: { type: Type.STRING },
                        logInput: { type: Type.STRING },
                        diff: { type: Type.STRING },
                        codeInput: { type: Type.STRING },
                        jsonInput: { type: Type.STRING },
                    }
                }
            },
            required: ['featureId', 'props']
        }
    }
];

// Dynamically add the workspace action
const functionDeclarations: FunctionDeclaration[] = [
    ...baseFunctionDeclarations,
    {
        name: 'runWorkspaceAction',
        description: 'Executes a defined action on a connected workspace service like
        Jira, Slack, or GitHub.',
        parameters: {
            type: Type.OBJECT,
            properties: {
                 actionId: {
                    type: Type.STRING,
                    description: 'The unique identifier for the action to execute.',
                    enum: [ ...ACTION_REGISTRY.keys() ]
                },
                params: {
                    type: Type.OBJECT,
                    description: 'An object containing the parameters for the action, matching its
                    required inputs.'
                }
            },
            required: ['actionId', 'params']
        }
    }
]

const knowledgeBase = FEATURE_TAXONOMY.map(f => `- ${f.name} (${f.id}):
${f.description} Inputs: ${f.inputs}`).join('\n');

const ExamplePromptButton: React.FC<{ text: string, onClick: (text: string) =>
void }> = ({ text, onClick }) => (
    <button
        onClick={() => onClick(text)}
        className="px-3 py-1.5 bg-surface border border-border rounded-full text-xs
        hover:bg-gray-100 dark:hover:bg-slate-700 transition-colors"
    >
        {text}
    </button>
)

export const AiCommandCenter: React.FC = () => {
    const { dispatch } = useGlobalState();
    const [prompt, setPrompt] = useState('');
    const [isLoading, setIsLoading] = useState(false);
    const [lastResponse, setLastResponse] = useState('');

    const handleCommand = useCallback(async () => {
        if (!prompt.trim()) return;
        setIsLoading(true);
```

```
    setLastResponse('');

    try {
        const response: CommandResponse = await getInferenceFunction(prompt,
        functionDeclarations, knowledgeBase);

        if (response.functionCalls && response.functionCalls.length > 0) {
            const call = response.functionCalls[0];
            const { name, args } = call;

            setLastResponse(`Understood! Executing command: ${name}`);

            switch (name) {
                case 'navigateTo':
                    dispatch({ type: 'SET_VIEW', payload: { view: args.featureId }});
                    break;
                case 'runFeatureWithInput':
                     dispatch({ type: 'SET_VIEW', payload: { view: args.featureId, props: args.props
                     } });
                    break;
                case 'runWorkspaceAction':
                    try {
                        const result = await executeWorkspaceAction(args.actionId, args.params);
                        setLastResponse(`Action '${args.actionId}' executed successfully.\n\nResult:
                        \`\`\`json\n${JSON.stringify(result, null, 2)}\n\`\`\``);
                    } catch (e) {
                        setLastResponse(`Action failed: ${e instanceof Error ? e.message : 'Unknown
                        error'}`);
                    }
                    break;
                default:
                    setLastResponse(`Unknown command: ${name}`);
            }
             setPrompt('');
        } else {
             setLastResponse(response.text);
        }

    } catch (err) {
        logError(err as Error, { prompt });
        setLastResponse(err instanceof Error ? err.message : 'An unknown error
        occurred.');
    } finally {
        setIsLoading(false);
    }
}, [prompt, dispatch]);

const handleKeyDown = (e: React.KeyboardEvent) => {
    if (e.key === 'Enter' && !e.shiftKey) {
        e.preventDefault();
        handleCommand();
    }
};

const handleExampleClick = (text: string) => {
    setPrompt(text);
}

return (
    <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
        <header className="mb-6 text-center">
            <h1 className="text-4xl font-extrabold tracking-tight flex items-center justify-
```

```
                center">
                    <CommandLineIcon />
                    <span className="ml-3">AI Command Center</span>
                </h1>
                <p className="mt-2 text-lg text-text-secondary">What would you like to do?</p>
            </header>

            <div className="flex-grow flex flex-col justify-end max-w-3xl w-full mx-auto">
                {lastResponse && (
                    <div className="mb-4 p-4 bg-surface rounded-lg text-text-primary border border-
                    border">
                        <p><strong>AI:</strong> {lastResponse}</p>
                    </div>
                )}
                 <div className="relative">
                    <textarea
                        value={prompt}
                        onChange={e => setPrompt(e.target.value)}
                        onKeyDown={handleKeyDown}
                        disabled={isLoading}
                        placeholder='Try "explain this code: const a = 1;" or "open the theme designer"'
                        className="w-full p-4 pr-28 rounded-lg bg-surface border border-border
                        focus:ring-2 focus:ring-primary focus:outline-none resize-none shadow-sm"
                        rows={2}
                    />
                    <button
                        onClick={handleCommand}
                        disabled={isLoading}
                        className="btn-primary absolute right-3 top-1/2 -translate-y-1/2 px-4 py-2"
                    >
                       {isLoading ? <LoadingSpinner/> : 'Send'}
                    </button>
                </div>
                 <div className="flex flex-wrap items-center justify-center gap-2 mt-4">
                    <ExamplePromptButton text="Open Theme Designer" onClick={handleExampleClick} />
                    <ExamplePromptButton text="Generate a commit for a bug fix"
                    onClick={handleExampleClick} />
                    <ExamplePromptButton text="Create a regex for email validation"
                    onClick={handleExampleClick} />
                </div>
                 <p className="text-xs text-text-secondary text-center mt-2">Press Enter to send,
                 Shift+Enter for new line.</p>
            </div>
        </div>
    );
};

// ===== ApiMockGenerator_44.tsx =====

import React, { useState, useEffect } from 'react';
import { generateMockData, parseOpenApiForMocking } from
'../../services/aiService.ts';
import { startMockServer, stopMockServer, setMockRoutes, isMockServerRunning }
from '../../services/mocking/mockServer.ts';
import { saveMockCollection, getAllMockCollections, deleteMockCollection } from
'../../services/mocking/db.ts';
import { ServerStackIcon, SparklesIcon } from '../icons.tsx';
import { LoadingSpinner } from '../shared/index.tsx';
import { useNotification } from '../../contexts/NotificationContext.tsx';

const citibankOpenApiSpec = `
openapi: 3.0.1
```

```
info:
  title: Citibank Demo Business Inc
  version: v1
  contact:
    name: Citibank Demo Business Inc Engineering Team
    url: https://citibankdemobusiness.dev
  description: The Citibank Demo Business Inc REST API. Please see
  https://docs.citibankdemobusiness.dev
    for more details.
paths:
  "/api/{accounts_type}/{account_id}/account_details":
    get:
      summary: list account_details
      tags:
      - AccountDetail
      description: Get a list of account details for a single internal or external
        account.
      operationId: listAccountDetails
      security:
      - basic_auth: []
      parameters:
      - name: accounts_type
        in: path
        schema:
          type: string
          enum:
          - external_accounts
          - internal_accounts
        required: true
      - name: account_id
        in: path
        description: The ID of the account.
        required: true
        schema:
          type: string
      responses:
        '200':
          description: successful
          content:
            application/json:
              schema:
                type: array
                items:
                  "$ref": "#/components/schemas/account_detail"
components:
  schemas:
    account_detail:
      type: object
      properties:
        id:
          type: string
          format: uuid
        object:
          type: string
        account_number:
          type: string
        account_number_type:
          type: string
          enum:
          - clabe
          - iban
          - other
```

```
    Transaction:
      type: object
      properties:
        id:
          type: string
          format: uuid
        amount:
          type: integer
        currency:
          type: string
`;

type GeneratorMode = 'simple' | 'openapi';

export const ApiMockGenerator: React.FC = () => {
    const [mode, setMode] = useState<GeneratorMode>('openapi');
    const [schema, setSchema] = useState("a user with an id, name, email, and a
    nested address object containing a city and country");
    const [count, setCount] = useState(5);
    const [collectionName, setCollectionName] = useState('users');
    const [openApiSpec, setOpenApiSpec] = useState(citibankOpenApiSpec.trim());
    const [collections, setCollections] = useState<any[]>([]);
    const [generatedData, setGeneratedData] = useState<any[] | null>(null);
    const [isLoading, setIsLoading] = useState(false);
    const [error, setError] = useState('');
    const [isServerRunning, setIsServerRunning] = useState(isMockServerRunning());
    const [routes, setRoutes] = useState<any[]>([]);
    const { addNotification } = useNotification();

    useEffect(() => {
        const loadCollections = async () => {
            const storedCollections = await getAllMockCollections();
            setCollections(storedCollections);
        };
        loadCollections();
    }, []);

    const updateRoutesOnServer = (routesToSet: any[], collectionsToUse: any[]) => {
        const mockRoutes = routesToSet.map(route => {
            const pathParts = route.path.split('/');
            const collectionIdFromPath = pathParts[pathParts.length -
            1].replace(/[^a-zA-Z0-9]/g, '');
            const matchingCollection = collectionsToUse.find(c =>
            c.id.toLowerCase().includes(collectionIdFromPath));

            return {
                ...route,
                response: {
                    status: 200,
                    body: matchingCollection ? matchingCollection.data : { message: `No mock data
                    found for this route.` }
                }
            };
        });
        setMockRoutes(mockRoutes as any);
    };

    useEffect(() => {
        if (isServerRunning) {
            updateRoutesOnServer(routes, collections);
        }
    }, [routes, collections, isServerRunning]);
```

```
const handleGenerateSimple = async () => {
    if (!schema.trim() || !collectionName.trim()) {
        setError('Schema description and collection name are required.');
        return;
    }
    setIsLoading(true);
    setError('');
    try {
        const data = await generateMockData(schema, count);
        setGeneratedData(data);
        const collectionId = collectionName.toLowerCase().replace(/\s/g, '-');
        await saveMockCollection({ id: collectionId, schemaDescription: schema, data });
        setCollections(await getAllMockCollections());
        addNotification(`Collection "${collectionId}" saved!`, 'success');
    } catch (err) {
        setError(err instanceof Error ? err.message : 'Failed to generate data.');
    } finally {
        setIsLoading(false);
    }
};

const handleGenerateFromSpec = async () => {
    if (!openApiSpec.trim()) {
        setError('OpenAPI specification is required.');
        return;
    }
    setIsLoading(true);
    setError('');
    setGeneratedData(null);

    try {
        const parsedSpec = await parseOpenApiForMocking(openApiSpec);

        const collectionsToSave: any[] = [];
        for (const schema of parsedSpec.schemas) {
            const data = await generateMockData(schema.description, 5);
            const collection = { id: schema.name, schemaDescription: schema.description,
            data };
            collectionsToSave.push(collection);
        }

        await Promise.all(collectionsToSave.map(c => saveMockCollection(c)));

        const newRoutes = parsedSpec.routes.map(route => ({
            path: route.path.replace(/{[^}]+}/g, '*'),
            method: route.method.toUpperCase(),
        }));

        setRoutes(newRoutes);

        const allCollections = await getAllMockCollections();
        setCollections(allCollections);
        setGeneratedData(collectionsToSave[0]?.data || []);
        addNotification('Mock API generated and routes configured. Start the server to
        use them.', 'info');

    } catch (err) {
        setError(err instanceof Error ? err.message : 'Failed to process OpenAPI
        spec.');
    } finally {
        setIsLoading(false);
    }
```

```
};

const handleServerToggle = async () => {
    if (isServerRunning) {
        await stopMockServer();
        setIsServerRunning(false);
    } else {
        try {
            await startMockServer();
            setIsServerRunning(true);
            updateRoutesOnServer(routes, collections);
        } catch (err) {
            setError(err instanceof Error ? err.message : 'Could not start server.');
        }
    }
};

const SimpleGenerator = () => (
    <>
        <div><label className="text-sm">Describe the data schema</label><textarea
        value={schema} onChange={e => setSchema(e.target.value)} className="w-full mt-1
        p-2 bg-background border border-border rounded" rows={4}/></div>
        <div className="flex gap-2">
            <div className="flex-grow"><label className="text-sm">Collection
            Name</label><input type="text" value={collectionName} onChange={e =>
            setCollectionName(e.target.value)} className="w-full mt-1 p-2 bg-background
            border border-border rounded"/></div>
            <div><label className="text-sm">Count</label><input type="number" value={count}
            onChange={e => setCount(Number(e.target.value))} className="w-20 mt-1 p-2 bg-
            background border border-border rounded"/></div>
        </div>
        <button onClick={handleGenerateSimple} disabled={isLoading} className="btn-
        primary py-2 flex items-center justify-center gap-2">{isLoading ?
        <LoadingSpinner/> : <><SparklesIcon/> Generate & Save</>}</button>
    </>
);

const OpenApiGenerator = () => (
    <>
        <div>
            <label className="text-sm">OpenAPI 3 Specification (YAML)</label>
            <textarea value={openApiSpec} onChange={e => setOpenApiSpec(e.target.value)}
            className="w-full mt-1 p-2 bg-background border border-border rounded font-mono
            text-xs" rows={12}/>
        </div>
        <button onClick={handleGenerateFromSpec} disabled={isLoading} className="btn-
        primary py-2 flex items-center justify-center gap-2">{isLoading ?
        <LoadingSpinner/> : <><SparklesIcon/> Generate Mocks & Routes</>}</button>
    </>
);

return (
    <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
        <header className="mb-6 flex justify-between items-start">
            <div>
                <h1 className="text-3xl font-bold flex items-center"><ServerStackIcon /><span
                className="ml-3">AI API Mock Server</span></h1>
                <p className="text-text-secondary mt-1">Generate and serve mock API data locally
                using a service worker.</p>
            </div>
            <button onClick={handleServerToggle} className={`px-4 py-2 rounded-md font-
            semibold flex items-center gap-2 ${isServerRunning ? 'bg-green-100 text-
```

```
                green-700' : 'bg-gray-100 dark:bg-slate-700'}`}>
                    <span className={`w-3 h-3 rounded-full ${isServerRunning ? 'bg-green-500
                    animate-pulse' : 'bg-gray-400'}`}></span>
                    {isServerRunning ? 'Server Running' : 'Server Stopped'}
                </button>
            </header>
            <div className="flex-grow grid grid-cols-1 lg:grid-cols-2 gap-6 min-h-0">
                <div className="lg:col-span-1 flex flex-col gap-4 bg-surface p-4 border border-
                border rounded-lg">
                    <div className="flex border-b border-border">
                        <button onClick={() => setMode('simple')} className={`px-4 py-2 text-sm ${mode
                        === 'simple' ? 'border-b-2 border-primary' : ''}`}>Simple Schema</button>
                        <button onClick={() => setMode('openapi')} className={`px-4 py-2 text-sm ${mode
                        === 'openapi' ? 'border-b-2 border-primary' : ''}`}>OpenAPI Spec</button>
                    </div>
                    {mode === 'simple' ? <SimpleGenerator /> : <OpenApiGenerator />}
                    {error && <p className="text-red-500 text-xs text-center p-2 bg-red-500/10
                    rounded">{error}</p>}
                </div>

                <div className="lg:col-span-1 flex flex-col gap-4 min-h-0">
                   <div className="bg-surface p-4 border border-border rounded-lg flex-grow flex
                   flex-col min-h-0">
                        <h3 className="text-lg font-bold mb-2">Server Status & Data</h3>
                        <div className="flex-grow grid grid-cols-2 gap-4 min-h-0">
                            <div className="overflow-y-auto pr-2">
                                <h4 className="font-semibold text-sm mb-1 sticky top-0 bg-surface pb-1">Saved
                                Collections ({collections.length})</h4>
                                {collections.map(c => <div key={c.id} className="text-xs p-2 bg-background
                                rounded border border-border mb-1">{c.id} ({c.data.length} items)</div>)}
                            </div>
                            <div className="overflow-y-auto pr-2">
                                <h4 className="font-semibold text-sm mb-1 sticky top-0 bg-surface
                                pb-1">Configured Routes ({routes.length})</h4>
                                {routes.map((r, i) => <div key={i} className="flex gap-2 items-center
                                mb-1"><span className="text-xs font-bold text-primary">{r.method}</span><span
                                className="text-xs font-mono">{r.path}</span></div>)}
                            </div>
                        </div>
                   </div>
                </div>
            </div>
        </div>
    );
};


// ===== FeatureForge_2.tsx =====

import React, { useState, useEffect, useCallback } from 'react';
import { generateAppFeatureComponent } from '../../services/aiService.ts';
import { getAllCustomFeatures, saveCustomFeature, deleteCustomFeature } from
'../../services/dbService.ts';
import type { CustomFeature } from '../../types.ts';
import { CpuChipIcon, PlusIcon, TrashIcon } from '../icons.tsx';
import { LoadingSpinner, MarkdownRenderer } from '../shared/index.tsx';
import { useNotification } from '../../contexts/NotificationContext.tsx';
import { ALL_FEATURES } from './index.ts';
import { CustomFeatureRunner } from './CustomFeatureRunner.tsx';

const ICON_MAP: Record<string, React.FC> = ALL_FEATURES.reduce((acc, feature) =>
{
```

```
    const iconType = (feature.icon as React.ReactElement)?.type;
    if (typeof iconType === 'function' && iconType.name) {
      const iconName = iconType.name;
      acc[iconName] = iconType as React.FC;
    }
    return acc;
  }, {} as Record<string, React.FC>);


export const FeatureForge: React.FC = () => {
    const [customFeatures, setCustomFeatures] = useState<CustomFeature[]>([]);
    const [isLoading, setIsLoading] = useState<'list' | 'generate' | false>(false);
    const [isGenerating, setIsGenerating] = useState(false);
    const [prompt, setPrompt] = useState('A tool to convert JSON to YAML');
    const [generatedFeature, setGeneratedFeature] = useState<Omit<CustomFeature,
    'id'> | null>(null);
    const { addNotification } = useNotification();

    const fetchFeatures = useCallback(async () => {
        setIsLoading('list');
        const features = await getAllCustomFeatures();
        setCustomFeatures(features);
        setIsLoading(false);
    }, []);

    useEffect(() => {
        fetchFeatures();
    }, [fetchFeatures]);

    const handleGenerate = async () => {
        if (!prompt.trim()) return;
        setIsGenerating(true);
        setGeneratedFeature(null);
        try {
            const result = await generateAppFeatureComponent(prompt);
            setGeneratedFeature(result);
            addNotification('Feature code generated! Review and save.', 'info');
        } catch (err) {
            addNotification(err instanceof Error ? err.message : 'Failed to generate
            feature', 'error');
        } finally {
            setIsGenerating(false);
        }
    };

    const handleSave = async () => {
        if (!generatedFeature) return;
        const newFeature: CustomFeature = {
            ...generatedFeature,
            id: `custom-${Date.now()}`
        };
        await saveCustomFeature(newFeature);
        // Dispatch event to notify other parts of the app (like the desktop view) to
        reload features
        window.dispatchEvent(new CustomEvent('custom-feature-update'));

        setGeneratedFeature(null);
        setPrompt('');
        fetchFeatures();
        addNotification(`Feature "${newFeature.name}" saved! It's now available on your
        desktop.`, 'success');
    };
```

```
const handleDelete = async (id: string) => {
    if (window.confirm("Are you sure you want to delete this feature?")) {
        await deleteCustomFeature(id);
        // Dispatch event to notify other parts of the app (like the desktop view) to
        reload features
        window.dispatchEvent(new CustomEvent('custom-feature-update'));
        fetchFeatures();
        addNotification('Feature deleted.', 'info');
    }
};

const IconComponent = ({ name }: { name: string }) => {
    const Comp = ICON_MAP[name];
    return Comp ? <Comp /> : <CpuChipIcon />;
};

return (
    <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
        <header className="mb-6">
            <h1 className="text-3xl font-bold flex items-center"><CpuChipIcon /><span
            className="ml-3">Feature Forge</span></h1>
            <p className="text-text-secondary mt-1">Use AI to create new tools and add them
            to your desktop.</p>
        </header>
        <div className="flex-grow grid grid-cols-1 lg:grid-cols-2 gap-6 min-h-0">
            {/* Left: Generator & Preview */}
            <div className="lg:col-span-1 flex flex-col gap-4">
                <div className="bg-surface p-4 border border-border rounded-lg">
                    <h3 className="text-lg font-bold">1. Create a New Feature</h3>
                    <div className="flex flex-col mt-2">
                        <label className="text-sm">Describe the tool you want to build</label>
                        <textarea value={prompt} onChange={e => setPrompt(e.target.value)}
                        className="w-full mt-1 p-2 bg-background border border-border rounded"
                        rows={3}/>
                    </div>
                    <button onClick={handleGenerate} disabled={isGenerating} className="btn-primary
                    w-full mt-2 py-2 flex items-center justify-center gap-2">{isGenerating ?
                    <LoadingSpinner/> : 'Generate Feature'}</button>
                </div>
                {generatedFeature && (
                    <div className="flex-grow flex flex-col bg-surface p-4 border border-dashed
                    rounded-lg space-y-2 animate-pop-in min-h-0">
                        <div className="flex justify-between items-center">
                            <h4 className="font-bold">2. Review & Save</h4>
                            <button onClick={handleSave} className="px-4 py-1 bg-green-600 text-white font-
                            bold rounded-md text-sm">Save Feature</button>
                        </div>
                        <p className="text-sm"><strong>Name:</strong> {generatedFeature.name}</p>
                        <div className="flex-grow border rounded-md overflow-hidden min-h-[200px]">
                             <CustomFeatureRunner feature={{ ...generatedFeature, id: 'preview' }} />
                        </div>
                    </div>
                )}
            </div>

            {/* Right: Existing Custom Features */}
            <div className="lg:col-span-1 flex flex-col gap-4 min-h-0">
               <div className="bg-surface p-4 border border-border rounded-lg flex-grow flex
               flex-col min-h-0">
                    <h3 className="text-lg font-bold mb-2">3. Your Custom Features</h3>
                    <div className="flex-grow overflow-y-auto pr-2">
                        {isLoading === 'list' && <LoadingSpinner />}
```

```
                            {customFeatures.length === 0 && !isLoading && <p className="text-text-secondary
                            text-center py-8">You haven't created any features yet.</p>}
                            <div className="space-y-3">
                                {customFeatures.map(feature => (
                                    <div key={feature.id} className="group bg-background p-3 rounded-lg border
                                    border-border flex items-center justify-between">
                                        <div className="flex items-center gap-3">
                                            <div className="text-primary"><IconComponent name={feature.icon} /></div>
                                            <div>
                                                <h4 className="font-semibold">{feature.name}</h4>
                                                <p className="text-xs text-text-secondary">{feature.description}</p>
                                            </div>
                                        </div>
                                        <button onClick={() => handleDelete(feature.id)} className="opacity-0 group-
                                        hover:opacity-100 text-red-500 hover:text-red-700 p-1"><TrashIcon /></button>
                                    </div>
                                ))}
                            </div>
                        </div>
                    </div>
                </div>
            </div>
        );
};

// ===== AiCodeExplainer_55.tsx =====

/**
 * @license
 * SPDX-License-Identifier: Apache-2.0
*/
import React, { useState, useCallback, useEffect, useMemo, useRef } from
'react';
import mermaid from 'mermaid';
import { explainCodeStructured, generateMermaidJs } from
'../../services/index.ts';
import type { StructuredExplanation } from '../../types.ts';
import { CpuChipIcon } from '../icons/index.ts';
import { MarkdownRenderer, LoadingSpinner } from '../shared/index.tsx';

const exampleCode = `const bubbleSort = (arr) => {
  for (let i = 0; i < arr.length; i++) {
    for (let j = 0; j < arr.length - i - 1; j++) {
      if (arr[j] > arr[j + 1]) {
        [arr[j], arr[j + 1]] = [arr[j + 1], arr[j]];
      }
    }
  }
  return arr;
};`;

type ExplanationTab = 'summary' | 'lineByLine' | 'complexity' | 'suggestions' |
'flowchart';

const simpleSyntaxHighlight = (code: string) => {
    const escapedCode = code
        .replace(/&/g, '&amp;')
        .replace(/</g, '&lt;')
        .replace(/>/g, '&gt;');

    return escapedCode
```

```
        .replace(/\b(const|let|var|function|return|if|for|=>|import|from|export|default)
        \b/g, '<span class="text-indigo-400 font-semibold">$1</span>')
        .replace(/(\`|'|")(.*?)(\`|'|")/g, '<span class="text-
        emerald-400">$1$2$3</span>')
        .replace(/(\/\/.*)/g, '<span class="text-gray-400 italic">$1</span>')
        .replace(/(\{|\}|\(|\)|\[|\])/g, '<span class="text-gray-400">$1</span>');
};

mermaid.initialize({ startOnLoad: false, theme: 'neutral', securityLevel:
'loose' });

export const AiCodeExplainer: React.FC<{ initialCode?: string }> = ({
initialCode }) => {
    const [code, setCode] = useState<string>(initialCode || exampleCode);
    const [explanation, setExplanation] = useState<StructuredExplanation |
    null>(null);
    const [mermaidCode, setMermaidCode] = useState<string>('');
    const [mermaidSvg, setMermaidSvg] = useState<string | null>(null);
    const [isLoading, setIsLoading] = useState<boolean>(false);
    const [error, setError] = useState<string>('');
    const [activeTab, setActiveTab] = useState<ExplanationTab>('summary');
    const textareaRef = useRef<HTMLTextAreaElement>(null);
    const preRef = useRef<HTMLPreElement>(null);

    const handleExplain = useCallback(async (codeToExplain: string) => {
        if (!codeToExplain.trim()) {
            setError('Please enter some code to explain.');
            return;
        }
        setIsLoading(true);
        setError('');
        setExplanation(null);
        setMermaidCode('');
        setMermaidSvg(null);
        setActiveTab('summary');
        try {
            const [explanationResult, mermaidResult] = await Promise.all([
                explainCodeStructured(codeToExplain),
                generateMermaidJs(codeToExplain)
            ]);
            setExplanation(explanationResult);
            setMermaidCode(mermaidResult.replace(/```mermaid\n|```/g, ''));

        } catch (err) {
            const errorMessage = err instanceof Error ? err.message : 'An unknown error
            occurred.';
            setError(`Failed to get explanation: ${errorMessage}`);
        } finally {
            setIsLoading(false);
        }
    }, []);

    useEffect(() => {
        if (initialCode) {
            setCode(initialCode);
            handleExplain(initialCode);
        }
    }, [initialCode, handleExplain]);

    useEffect(() => {
        const renderMermaid = async () => {
             if (activeTab === 'flowchart' && mermaidCode && !mermaidSvg) {
```

```
            try {
                const theme = document.documentElement.classList.contains('dark') ? 'dark' :
                'neutral';
                mermaid.initialize({ startOnLoad: false, theme, securityLevel: 'loose' });
                const { svg } = await mermaid.render(`mermaid-graph-${Date.now()}`,
                mermaidCode);
                setMermaidSvg(svg);
            } catch (e) {
                console.error("Mermaid rendering error:", e);
                setMermaidSvg(`<p class="text-red-500 p-4">Error rendering flowchart. The
                diagram syntax may be invalid.</p>`);
            }
        }
    }
    renderMermaid();
}, [activeTab, mermaidCode, mermaidSvg]);


const handleScroll = () => {
    if (preRef.current && textareaRef.current) {
        preRef.current.scrollTop = textareaRef.current.scrollTop;
        preRef.current.scrollLeft = textareaRef.current.scrollLeft;
    }
};

const highlightedCode = useMemo(() => simpleSyntaxHighlight(code), [code]);

const renderTabContent = () => {
    if (!explanation) return null;
    switch(activeTab) {
        case 'summary':
            return <MarkdownRenderer content={explanation.summary} />;
        case 'lineByLine':
            return (
                <div className="space-y-3">
                    {explanation.lineByLine.map((item, index) => (
                        <div key={index} className="p-3 bg-background rounded-md border border-border">
                            <p className="font-mono text-xs text-primary mb-1">Lines: {item.lines}</p>
                            <p className="text-sm">{item.explanation}</p>
                        </div>
                    ))}
                </div>
            );
        case 'complexity':
            return (
                <div>
                    <p><strong>Time Complexity:</strong> <span className="font-mono text-
                    amber-600">{explanation.complexity.time}</span></p>
                    <p><strong>Space Complexity:</strong> <span className="font-mono text-
                    amber-600">{explanation.complexity.space}</span></p>
                </div>
            );
        case 'suggestions':
            return (
                 <ul className="list-disc list-inside space-y-2">
                    {explanation.suggestions.map((item, index) => <li key={index}>{item}</li>)}
                </ul>
            );
        case 'flowchart':
            if (!mermaidCode) {
                return <div className="text-text-secondary h-full flex items-center justify-
                center">Could not generate a flowchart for this code.</div>;
```

```
            }
            if (!mermaidSvg) {
                return <div className="w-full h-full flex items-center justify-
                center"><LoadingSpinner /></div>;
            }
            return (
                <div
                    className="w-full h-full flex items-center justify-center [&>svg]:max-w-full
                    [&>svg]:max-h-full"
                    dangerouslySetInnerHTML={{ __html: mermaidSvg }}
                />
            );
    }
}

return (
    <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
        <header className="mb-6 flex-shrink-0">
            <h1 className="text-3xl font-bold flex items-center">
                <CpuChipIcon />
                <span className="ml-3">AI Code Explainer</span>
            </h1>
            <p className="text-text-secondary mt-1">Get a detailed, structured analysis of
            any code snippet.</p>
        </header>
        <div className="flex-grow grid grid-cols-1 md:grid-cols-2 gap-6 min-h-0">

            {/* Left Column: Code Input */}
            <div className="flex flex-col min-h-0 md:col-span-1">
                <label htmlFor="code-input" className="text-sm font-medium text-text-secondary
                mb-2">Your Code</label>
                <div className="relative flex-grow bg-surface border border-border rounded-md
                focus-within:ring-2 focus-within:ring-primary overflow-hidden">
                    <textarea
                        ref={textareaRef}
                        id="code-input"
                        value={code}
                        onChange={(e) => setCode(e.target.value)}
                        onScroll={handleScroll}
                        placeholder="Paste your code here..."
                        spellCheck="false"
                        className="absolute inset-0 w-full h-full p-4 bg-transparent resize-none font-
                        mono text-sm text-transparent caret-primary outline-none z-10"
                    />
                    <pre
                        ref={preRef}
                        aria-hidden="true"
                        className="absolute inset-0 w-full h-full p-4 font-mono text-sm text-text-
                        primary pointer-events-none z-0 whitespace-pre-wrap overflow-auto no-scrollbar"
                        dangerouslySetInnerHTML={{ __html: highlightedCode + '\n' }}
                    />
                </div>
                <div className="mt-4 flex-shrink-0">
                    <button
                        onClick={() => handleExplain(code)}
                        disabled={isLoading}
                        className="btn-primary w-full flex items-center justify-center px-6 py-3"
                    >
                        {isLoading ? <LoadingSpinner/> : 'Analyze Code'}
                    </button>
                </div>
            </div>
```

```
                {/* Right Column: AI Analysis */}
                <div className="flex flex-col min-h-0 md:col-span-1">
                    <label className="text-sm font-medium text-text-secondary mb-2">AI
                    Analysis</label>
                    <div className="relative flex-grow flex flex-col bg-surface border border-border
                    rounded-md overflow-hidden">
                        <div className="flex-shrink-0 flex border-b border-border">
                           {(['summary', 'lineByLine', 'complexity', 'suggestions', 'flowchart'] as
                           ExplanationTab[]).map(tab => (
                               <button key={tab} onClick={() => setActiveTab(tab)} disabled={!explanation}
                                className={`px-4 py-2 text-sm font-medium capitalize transition-colors
                                ${activeTab === tab ? 'bg-background text-primary font-semibold' : 'text-text-
                                secondary hover:bg-gray-100 dark:hover:bg-slate-700 disabled:text-gray-400
                                dark:disabled:text-slate-500'}`}>
                                   {tab.replace(/([A-Z])/g, ' $1')}
                               </button>
                           ))}
                        </div>
                        <div className="p-4 flex-grow overflow-y-auto">
                            {isLoading && <div className="flex items-center justify-center
                            h-full"><LoadingSpinner /></div>}
                            {error && <p className="text-red-500">{error}</p>}
                            {explanation && !isLoading && renderTabContent()}
                            {!isLoading && !explanation && !error && <div className="text-text-secondary
                            h-full flex items-center justify-center">The analysis will appear here.</div>}
                        </div>
                    </div>
                </div>
            </div>
        );
};


// ===== VisualGitTree_57.tsx =====

import React, { useState, useCallback, useEffect, useMemo } from 'react';
import { GitBranchIcon, ArrowDownTrayIcon } from '../icons/index.ts';
// FIX: Corrected import path for ai services.
import { generateChangelogFromLogStream } from '../../services/index.ts';
import { LoadingSpinner, MarkdownRenderer } from '../shared/index.tsx';
import { downloadFile } from '../../services/fileUtils.ts';

const exampleLog = `* commit 3a4b5c6d7e8f9g0h1i2j3k4l5m6n7o8p9q0r (HEAD -> main,
origin/main)
|\\  Merge: 1a2b3c4 2d3e4f5
| | Author: Dev One <dev.one@example.com>
| | Date:   Mon Jul 15 11:30:00 2024 -0400
| |
| |     feat: Implement collapsible sidebar navigation
| |
* | commit 2d3e4f5g6h7i8j9k0l1m2n3o4p5q6r7s8t9u (feature/new-sidebar)
| | Author: Dev Two <dev.two@example.com>
| | Date:   Mon Jul 15 10:00:00 2024 -0400
| |
| |     feat: Add icons to sidebar items
| |
* | commit 1a2b3c4d5e6f7g8h9i0j1k2l3m4n5o6p7q8r
|/  Author: Dev One <dev.one@example.com>
|   Date:   Fri Jul 12 16:45:00 2024 -0400
|
|       fix: Correct user authentication bug`;
```

```
const CommitGraph = ({ logInput }: { logInput: string }) => {
    const commits = useMemo(() => {
        const lines = logInput.split('\n');
        const parsedCommits: any[] = [];
        let currentCommit: any = null;

        lines.forEach(line => {
            const commitMatch = line.match(/^.?[\\|/ ]*\* commit (\w+)(.*)/);
            if (commitMatch) {
                if (currentCommit) parsedCommits.push(currentCommit);
                currentCommit = {
                    hash: commitMatch[1],
                    shortHash: commitMatch[1].substring(0, 7),
                    refs: commitMatch[2].trim(),
                    message: '',
                    author: '',
                };
            } else if (currentCommit) {
                 if (line.includes('Author:')) currentCommit.author =
                 line.split('Author:')[1].trim();
                 else if (line.trim().length > 0 && !line.match(/^[\\|/ ]*[\\|/ ]/)) {
                     currentCommit.message += line.trim() + ' ';
                 }
            }
        });
        if (currentCommit) parsedCommits.push(currentCommit);

        return parsedCommits.map((c, i) => ({ ...c, x: 50, y: 50 + i * 60 }));
    }, [logInput]);

    return (
         <svg width="100%" height={50 + commits.length * 60} className="min-h-[200px]">
            {commits.map((commit, i) => {
                const parent = commits[i + 1];
                return (
                    <g key={commit.hash}>
                        {parent && <line x1={commit.x} y1={commit.y} x2={parent.x} y2={parent.y}
                        stroke="var(--color-border)" strokeWidth="2" />}
                        <g className="group cursor-pointer">
                            <circle cx={commit.x} cy={commit.y} r="8" fill="var(--color-primary)"
                            stroke="var(--color-surface)" strokeWidth="3" />
                            <foreignObject x={commit.x + 20} y={commit.y - 25} width="350" height="50">
                                <div className="text-sm p-1">
                                    <p className="font-bold truncate text-text-primary">{commit.message}</p>
                                    <p className="text-xs text-text-secondary font-mono">{commit.shortHash} <span
                                    className="text-amber-600">{commit.refs}</span></p>
                                </div>
                            </foreignObject>
                            <title>{`Commit: ${commit.hash}\nAuthor:
                            ${commit.author}\n\n${commit.message}`}</title>
                        </g>
                    </g>
                );
            })}
        </svg>
    );
};

export const VisualGitTree: React.FC<{ logInput?: string }> = ({ logInput:
initialLogInput }) => {
    const [logInput, setLogInput] = useState(initialLogInput || exampleLog);
    const [analysis, setAnalysis] = useState('');
```

```
const [isLoading, setIsLoading] = useState(false);
const [error, setError] = useState('');

const handleAnalyze = useCallback(async (logToAnalyze: string) => {
    if (!logToAnalyze.trim()) {
        setError('Please paste git log output.');
        return;
    }
    setIsLoading(true);
    setError('');
    setAnalysis('');
    try {
        const stream = generateChangelogFromLogStream(logToAnalyze);
        let fullResponse = '';
        for await (const chunk of stream) {
            fullResponse += chunk;
            setAnalysis(fullResponse);
        }
    } catch (err) {
        const errorMessage = err instanceof Error ? err.message : 'An unknown error
        occurred.';
        setError(`Failed to analyze log: ${errorMessage}`);
    } finally {
        setIsLoading(false);
    }
}, []);

useEffect(() => {
    if (initialLogInput) {
        setLogInput(initialLogInput);
        handleAnalyze(initialLogInput);
    }
}, [initialLogInput, handleAnalyze]);

return (
    <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
        <header className="mb-6">
            <h1 className="text-3xl font-bold flex items-center">
                <GitBranchIcon />
                <span className="ml-3">Visual Git Tree</span>
            </h1>
            <p className="text-text-secondary mt-1">Paste your `git log --graph` output to
            visualize the history and get an AI summary.</p>
        </header>
        <div className="flex-grow grid grid-cols-1 lg:grid-cols-2 gap-6 h-full overflow-
        hidden">
            <div className="flex flex-col h-full">
                <label htmlFor="log-input" className="text-sm font-medium text-text-secondary
                mb-2">Git Log Output</label>
                <textarea
                    id="log-input"
                    value={logInput}
                    onChange={(e) => setLogInput(e.target.value)}
                    placeholder="Paste your git log output here..."
                    className="flex-grow p-4 bg-surface border border-border rounded-md resize-none
                    font-mono text-sm"
                />
                <button
                    onClick={() => handleAnalyze(logInput)}
                    disabled={isLoading}
                    className="btn-primary mt-4 w-full flex items-center justify-center px-6 py-3"
                >
```

```
                        {isLoading ? <LoadingSpinner /> : 'Analyze & Summarize'}
                    </button>
                </div>
                <div className="flex flex-col h-full gap-4">
                    <div className="flex flex-col h-1/2">
                        <label className="text-sm font-medium text-text-secondary mb-2">Commit
                        Graph</label>
                        <div className="flex-grow p-2 bg-surface border border-border rounded-md
                        overflow-auto">
                            <CommitGraph logInput={logInput} />
                        </div>
                    </div>
                     <div className="flex flex-col h-1/2">
                        <div className="flex justify-between items-center mb-2">
                            <label className="text-sm font-medium text-text-secondary">AI Summary</label>
                            {analysis && !isLoading && (
                                <button onClick={() => downloadFile(analysis, 'summary.md', 'text/markdown')}
                                className="flex items-center gap-1 px-3 py-1 bg-gray-100 text-xs rounded-md
                                hover:bg-gray-200">
                                    <ArrowDownTrayIcon className="w-4 h-4"/> Download Summary
                                </button>
                            )}
                        </div>
                        <div className="flex-grow p-4 bg-background border border-border rounded-md
                        overflow-y-auto">
                            {isLoading && <div className="flex items-center justify-center
                            h-full"><LoadingSpinner /></div>}
                            {error && <p className="text-red-500">{error}</p>}
                            {analysis && !isLoading && <MarkdownRenderer content={analysis} />}
                            {!isLoading && !analysis && !error && <div className="text-text-secondary h-full
                            flex items-center justify-center">AI summary will appear here.</div>}
                        </div>
                    </div>
                </div>
            </div>
        </div>
    );
};

// ===== SnippetVault_57.tsx =====

import React, { useState, useEffect, useMemo } from 'react';
import { LockClosedIcon, SparklesIcon, TrashIcon, ClipboardDocumentIcon,
ArrowDownTrayIcon } from '../icons/index.ts';
import { useLocalStorage } from '../../hooks/useLocalStorage.ts';
// FIX: Corrected import path for ai services.
import { enhanceSnippetStream, generateTagsForCode } from
'../../services/index.ts';
import { LoadingSpinner } from '../shared/index.tsx';
import { downloadFile } from '../../services/fileUtils.ts';
import { useNotification } from '../../contexts/NotificationContext.tsx';

interface Snippet {
    id: number; name: string; code: string; language: string; tags: string[];
}

const langToExt: Record<string, string> = {
    javascript: 'js',
    typescript: 'ts',
    python: 'py',
    css: 'css',
    html: 'html',
```

```
    json: 'json',
    markdown: 'md',
    plaintext: 'txt',
};

export const SnippetVault: React.FC = () => {
    const [snippets, setSnippets] = useLocalStorage<Snippet[]>('devcore_snippets',
    [{ id: 1, name: 'React Hook Boilerplate', language: 'javascript', code: `import
    { useState } from 'react';\n\nconst useCustomHook = () => {\n  const [value,
    setValue] = useState(null);\n  return { value, setValue };\n};`, tags: ['react',
    'hook'] }]);
    const [activeSnippet, setActiveSnippet] = useState<Snippet | null>(null);
    const [isEnhancing, setIsEnhancing] = useState(false);
    const [searchTerm, setSearchTerm] = useState('');
    const [isEditingName, setIsEditingName] = useState(false);
    const { addNotification } = useNotification();

    const filteredSnippets = useMemo(() => {
        if (!searchTerm) return snippets;
        const lowerSearch = searchTerm.toLowerCase();
        return snippets.filter((s: Snippet) =>
            s.name.toLowerCase().includes(lowerSearch) ||
            s.code.toLowerCase().includes(lowerSearch) ||
            (s.tags && s.tags.some(t => t.toLowerCase().includes(lowerSearch)))
        );
    }, [snippets, searchTerm]);

    useEffect(() => {
        if (!activeSnippet && filteredSnippets.length > 0)
        setActiveSnippet(filteredSnippets[0]);
        if (activeSnippet) setActiveSnippet(snippets.find((s: Snippet) => s.id ===
        activeSnippet.id) || null);
    }, [snippets, activeSnippet, filteredSnippets]);

    const updateSnippet = (snippet: Snippet) => {
        setSnippets(snippets.map((s: Snippet) => s.id === snippet.id ? snippet : s));
        setActiveSnippet(snippet);
    };

    const handleEnhance = async () => {
        if (!activeSnippet) return;
        setIsEnhancing(true);
        try {
            const stream = enhanceSnippetStream(activeSnippet.code);
            let fullResponse = '';
            for await (const chunk of stream) {
                fullResponse += chunk;
                updateSnippet({ ...activeSnippet, code: fullResponse.replace(/^```(?:\w+\n)?/,
                '').replace(/```$/, '') });
            }
        } finally { setIsEnhancing(false); }
    };

    const handleAiTagging = async (snippet: Snippet) => {
        try {
            const { tags } = await generateTagsForCode(snippet.code);
            updateSnippet({ ...snippet, tags });
            addNotification("Tags generated!", 'success');
        } catch (e) {
            addNotification("Could not generate tags.", 'error');
        }
    };
```

```
const handleAddNew = () => {
    const newSnippet: Snippet = {
        id: Date.now(),
        name: 'New Snippet',
        code: '// Your code here',
        language: 'javascript',
        tags: [],
    };
    setSnippets([newSnippet, ...snippets]);
    setActiveSnippet(newSnippet);
};

const handleDelete = (id: number) => {
    setSnippets(snippets.filter(s => s.id !== id));
    if (activeSnippet?.id === id) {
        setActiveSnippet(null);
    }
};

const handleDownload = () => {
    if (!activeSnippet) return;
    const ext = langToExt[activeSnippet.language] || 'txt';
    downloadFile(activeSnippet.code, `${activeSnippet.name}.${ext}`);
};

return (
    <div className="h-full flex text-text-primary">
        <aside className="w-80 bg-surface border-r border-border flex flex-col">
            <div className="p-4 border-b border-border">
                <input type="text" placeholder="Search snippets..." value={searchTerm}
                onChange={e => setSearchTerm(e.target.value)} className="w-full p-2 bg-
                background border rounded-md"/>
            </div>
            <div className="flex-grow overflow-y-auto">
                {filteredSnippets.map(snippet => (
                    <div key={snippet.id} onClick={() => setActiveSnippet(snippet)} className={`p-3
                    cursor-pointer ${activeSnippet?.id === snippet.id ? 'bg-primary/10' : 'hover:bg-
                    gray-100 dark:hover:bg-slate-700'}`}>
                        <p className={`font-semibold ${activeSnippet?.id === snippet.id ? 'text-primary'
                        : 'text-text-primary'}`}>{snippet.name}</p>
                        <div className="flex flex-wrap gap-1 mt-1">
                            {snippet.tags.map(tag => <span key={tag} className="text-xs bg-gray-200 dark:bg-
                            slate-600 px-2 py-0.5 rounded-full">{tag}</span>)}
                        </div>
                    </div>
                ))}
            </div>
            <div className="p-4 border-t border-border">
                <button onClick={handleAddNew} className="btn-primary w-full py-2">Add New
                Snippet</button>
            </div>
        </aside>
        <main className="flex-1 flex flex-col min-w-0">
            {activeSnippet ? (
                <>
                    <header className="flex justify-between items-center p-4 border-b border-border
                    bg-surface">
                        <input value={activeSnippet.name} onChange={e => updateSnippet({
                        ...activeSnippet, name: e.target.value })} className="bg-transparent text-xl
                        font-bold focus:outline-none"/>
                        <div className="flex items-center gap-2">
                            {/* FIX: Corrected prop passing for SparklesIcon */}
```

```
                                <button onClick={handleEnhance} disabled={isEnhancing} className="flex items-
                                center gap-1 px-3 py-1 bg-gray-100 text-xs rounded-md hover:bg-
                                gray-200">{isEnhancing ? <LoadingSpinner/> : <SparklesIcon className="w-4
                                h-4"/>} Enhance</button>
                                <button onClick={() => handleAiTagging(activeSnippet)} className="flex items-
                                center gap-1 px-3 py-1 bg-gray-100 text-xs rounded-md hover:bg-
                                gray-200"><SparklesIcon className="w-4 h-4"/> AI Tags</button>
                                <button onClick={handleDownload} className="p-2 hover:bg-gray-100 rounded-
                                md"><ArrowDownTrayIcon/></button>
                                <button onClick={() => handleDelete(activeSnippet.id)} className="p-2 text-
                                red-500 hover:bg-red-100 rounded-md"><TrashIcon/></button>
                            </div>
                        </header>
                        <textarea
                            value={activeSnippet.code}
                            onChange={e => updateSnippet({ ...activeSnippet, code: e.target.value })}
                            className="flex-grow p-4 font-mono text-sm bg-background focus:outline-none
                            resize-none"
                        />
                    </>
                ) : (
                    <div className="flex-grow flex items-center justify-center text-text-
                    secondary">Select a snippet or create a new one.</div>
                )}
            </main>
        </div>
    )
};

// ===== AiFeatureBuilder_57.tsx =====

import React, { useState, useCallback, useEffect } from 'react';
import type { GeneratedFile } from '../../types.ts';
// FIX: Corrected import path for ai services.
import { generateFeature, generateFullStackFeature, generateUnitTestsStream,
generateCommitMessageStream, generateDockerfile } from
'../../services/index.ts';
import { saveFile, getAllFiles, clearAllFiles } from
'../../services/dbService.ts';
import { useNotification } from '../../contexts/NotificationContext.tsx';
// FIX: Corrected import path for icons.
import { CpuChipIcon, DocumentTextIcon, BeakerIcon, GitBranchIcon, CloudIcon }
from '../icons/index.ts';
import { LoadingSpinner, MarkdownRenderer } from '../shared/index.tsx';

type SupplementalTab = 'TESTS' | 'COMMIT' | 'DEPLOYMENT' | 'CODE';
type OutputTab = GeneratedFile | SupplementalTab;

export const AiFeatureBuilder: React.FC = () => {
    const [prompt, setPrompt] = useState<string>('A simple "Hello World" React
    component with a button that shows an alert.');
    const [framework] = useState('React');
    const [styling] = useState('Tailwind CSS');
    const [includeBackend, setIncludeBackend] = useState(false);

    const [generatedFiles, setGeneratedFiles] = useState<GeneratedFile[]>([]);
    const [unitTests, setUnitTests] = useState<string>('');
    const [commitMessage, setCommitMessage] = useState<string>('');
    const [dockerfile, setDockerfile] = useState<string>('');

    const [activeTab, setActiveTab] = useState<OutputTab>('CODE');
    const [isLoading, setIsLoading] = useState<boolean>(false);
```

```
const [error, setError] = useState<string>('');

useEffect(() => {
    const loadFiles = async () => {
        const files = await getAllFiles();
        setGeneratedFiles(files);
        if (files.length > 0) setActiveTab(files[0]);
    };
    loadFiles();
}, []);

const handleGenerate = useCallback(async () => {
    if (!prompt.trim()) { setError('Please enter a feature description.'); return; }
    setIsLoading(true);
    setError('');
    await clearAllFiles();
    setGeneratedFiles([]); setUnitTests(''); setCommitMessage('');
    setDockerfile(''); setActiveTab('CODE');

    try {
        const resultFiles = includeBackend
            ? await generateFullStackFeature(prompt, framework, styling)
            : await generateFeature(prompt, framework, styling);

        for (const file of resultFiles) { await saveFile(file); }
        setGeneratedFiles(resultFiles);

        if (resultFiles.length > 0) {
            const componentFile = resultFiles.find(f => f.filePath.endsWith('.tsx') ||
            f.filePath.endsWith('.jsx'));
            setActiveTab(componentFile || resultFiles[0]);

            const testStream = generateUnitTestsStream(componentFile?.content ||
            resultFiles[0].content);
            const diffContext = resultFiles.map(f => `File:
            ${f.filePath}\n\n${f.content}`).join('\n---\n');
            const commitStream = generateCommitMessageStream(diffContext);

            let tests = ''; for await (const chunk of testStream) { tests += chunk;
            setUnitTests(tests); }
            let commit = ''; for await (const chunk of commitStream) { commit += chunk;
            setCommitMessage(commit); }

            if (!includeBackend) {
                const dockerfileStream = generateDockerfile(framework);
                let docker = ''; for await (const chunk of dockerfileStream) { docker += chunk;
                setDockerfile(docker); }
            }
        }
    } catch (err) {
        setError(err instanceof Error ? err.message : 'Failed to generate feature.');
    } finally {
        setIsLoading(false);
    }
}, [prompt, framework, styling, includeBackend]);

const renderContent = () => {
    if (typeof activeTab === 'string') {
        switch (activeTab) {
            case 'TESTS': return <MarkdownRenderer content={unitTests} />;
            case 'COMMIT': return <pre className="w-full h-full p-4 whitespace-pre-wrap
            font-sans text-sm">{commitMessage}</pre>;
```

```
            case 'DEPLOYMENT': return <MarkdownRenderer content={dockerfile} />;
            default: return <div className="p-4">Select a file</div>;
        }
    }
    return <MarkdownRenderer content={'```tsx\n' + activeTab.content + '\n```'} />;
}

return (
    <div className="h-full flex flex-col text-text-primary bg-surface">
        <header className="p-4 border-b border-border flex-shrink-0">
            <h1 className="text-xl font-bold flex items-center"><CpuChipIcon /><span
            className="ml-3">AI Feature Builder</span></h1>
        </header>
        <div className="flex-grow flex min-h-0">
            <main className="flex-1 flex flex-col min-w-0">
                <div className="flex-grow flex flex-col bg-background">
                     <div className="border-b border-border flex items-center bg-surface overflow-x-
                     auto">
                        {generatedFiles.map(file => (
                            <button key={file.filePath} onClick={() => setActiveTab(file)} className={`flex-
                            shrink-0 flex items-center gap-2 px-4 py-2 text-sm ${activeTab === file ? 'bg-
                            background border-b-2 border-primary text-text-primary' : 'text-text-secondary
                            hover:bg-gray-50'}`}><DocumentTextIcon /> {file.filePath}</button>
                        ))}
                        {unitTests && <button onClick={() => setActiveTab('TESTS')} className={`flex-
                        shrink-0 flex items-center gap-2 px-4 py-2 text-sm ${activeTab === 'TESTS' ?
                        'bg-background border-b-2 border-primary text-text-primary' : 'text-text-
                        secondary hover:bg-gray-50'}`}><BeakerIcon /> Tests</button>}
                        {commitMessage && <button onClick={() => setActiveTab('COMMIT')}
                        className={`flex-shrink-0 flex items-center gap-2 px-4 py-2 text-sm ${activeTab
                        === 'COMMIT' ? 'bg-background border-b-2 border-primary text-text-primary' :
                        'text-text-secondary hover:bg-gray-50'}`}><GitBranchIcon /> Commit</button>}
                        {dockerfile && !includeBackend && <button onClick={() =>
                        setActiveTab('DEPLOYMENT')} className={`flex-shrink-0 flex items-center gap-2
                        px-4 py-2 text-sm ${activeTab === 'DEPLOYMENT' ? 'bg-background border-b-2
                        border-primary text-text-primary' : 'text-text-secondary hover:bg-
                        gray-50'}`}><CloudIcon /> Dockerfile</button>}
                    </div>
                    <div className="flex-grow p-2 overflow-auto">
                        {isLoading && !generatedFiles.length ? <div className="flex justify-center
                        items-center h-full"><LoadingSpinner/></div> : renderContent()}
                    </div>
                </div>

                <div className="flex-shrink-0 p-4 border-t border-border bg-surface">
                     <div className="flex items-center gap-2 mb-2">
                        <label className="flex items-center gap-2 text-sm"><input type="checkbox"
                        checked={includeBackend} onChange={e => setIncludeBackend(e.target.checked)} />
                        Include Backend (Cloud Function + Firestore)</label>
                    </div>
                    <textarea value={prompt} onChange={(e) => setPrompt(e.target.value)}
                    placeholder="e.g., A user profile card with an avatar, name, and bio."
                    className="w-full p-2 bg-background border border-border rounded-md resize-none
                    text-sm h-20"/>
                     <div className="flex gap-2 mt-2">
                         <button onClick={handleGenerate} disabled={isLoading} className="btn-primary
                         flex-grow flex items-center justify-center gap-2 px-4 py-2">
                            {isLoading ? <><LoadingSpinner /> Generating...</> : 'Generate Feature'}
                        </button>
                     </div>
                     {error && <p className="text-red-600 text-xs mt-2 text-center">{error}</p>}
                </div>
```

```
                </main>
            </div>
        </div>
    );
};

// ===== AiPullRequestAssistant_55.tsx =====



import React, { useState, useMemo, useCallback } from 'react';
import * as Diff from 'diff';
// FIX: Corrected import path for ai services.
import { generatePrSummaryStructured, generateTechnicalSpecFromDiff,
downloadFile, createDocument, insertText } from '../../services/index.ts';
import type { StructuredPrSummary } from '../../types.ts';
// FIX: Corrected import path for icons.
import { AiPullRequestAssistantIcon, DocumentIcon } from '../icons/index.ts';
import { LoadingSpinner } from '../shared/index.tsx';
import { useNotification } from '../../contexts/NotificationContext.tsx';
import { useGlobalState } from '../../contexts/GlobalStateContext.tsx';

const exampleBefore = `function Greeter(props) {
  return <h1>Hello, {props.name}!</h1>;
}`;
const exampleAfter = `function Greeter({ name, enthusiasmLevel = 1 }) {
  const punctuation = '!'.repeat(enthusiasmLevel);
  return <h1>Hello, {name}{punctuation}</h1>;
}`;

export const AiPullRequestAssistant: React.FC = () => {
    const [beforeCode, setBeforeCode] = useState<string>(exampleBefore);
    const [afterCode, setAfterCode] = useState<string>(exampleAfter);
    const [isLoading, setIsLoading] = useState<boolean>(false);
    const [isExporting, setIsExporting] = useState<boolean>(false);
    const [error, setError] = useState<string>('');
    const [summary, setSummary] = useState<StructuredPrSummary | null>(null);

    const { addNotification } = useNotification();
    const { state } = useGlobalState();
    const { user } = state;

    const diff = useMemo(() => Diff.createPatch('component.tsx', beforeCode,
    afterCode), [beforeCode, afterCode]);

    const handleGenerateSummary = useCallback(async () => {
        if (!beforeCode.trim() && !afterCode.trim()) {
            setError('Please provide code to generate a summary.');
            return;
        }
        setIsLoading(true);
        setError('');
        setSummary(null);

        try {
            const result: StructuredPrSummary = await generatePrSummaryStructured(diff);
            setSummary(result);
        } catch (err) {
            const errorMessage = err instanceof Error ? err.message : 'An unknown error
            occurred.';
            setError(`Failed to generate summary: ${errorMessage}`);
```

```
    } finally {
        setIsLoading(false);
    }
}, [diff, beforeCode, afterCode]);

const handleExportToDocs = async () => {
    if (!summary || !user) {
        addNotification('Please generate a summary first and ensure you are signed in.',
        'error');
        return;
    }
    setIsExporting(true);
    try {
        const specContent = await generateTechnicalSpecFromDiff(diff, summary);
        const doc = await createDocument(`Tech Spec: ${summary.title}`);
        await insertText(doc.documentId, specContent);
        addNotification('Successfully exported to Google Docs!', 'success');
        window.open(doc.webViewLink, '_blank');
    } catch (err) {
        const errorMessage = err instanceof Error ? err.message : 'An unknown error
        occurred.';
        addNotification(`Failed to export: ${errorMessage}`, 'error');
    } finally {
        setIsExporting(false);
    }
};

return (
    <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
        <header className="mb-6">
            <h1 className="text-3xl font-bold flex items-center">
                <AiPullRequestAssistantIcon />
                <span className="ml-3">AI Pull Request Assistant</span>
            </h1>
            <p className="text-text-secondary mt-1">Generate a PR summary from code changes
            and export a full tech spec.</p>
        </header>
        <div className="flex-grow grid grid-cols-1 lg:grid-cols-2 gap-6 min-h-0">
            {/* Left side: Inputs and Generator */}
            <div className="flex flex-col gap-4 min-h-0">
                <div className="flex flex-col flex-1 min-h-0">
                    <label htmlFor="before-code" className="text-sm font-medium text-text-secondary
                    mb-2">Before</label>
                    <textarea id="before-code" value={beforeCode} onChange={e =>
                    setBeforeCode(e.target.value)} className="flex-grow p-4 bg-surface border
                    border-border rounded-md resize-none font-mono text-sm" />
                </div>
                <div className="flex flex-col flex-1 min-h-0">
                    <label htmlFor="after-code" className="text-sm font-medium text-text-secondary
                    mb-2">After</label>
                    <textarea id="after-code" value={afterCode} onChange={e =>
                    setAfterCode(e.target.value)} className="flex-grow p-4 bg-surface border border-
                    border rounded-md resize-none font-mono text-sm" />
                </div>
                <button
                    onClick={handleGenerateSummary}
                    disabled={isLoading}
                    className="btn-primary w-full mt-4 flex items-center justify-center gap-2 py-2"
                >
                    {isLoading ? <LoadingSpinner/> : 'Generate Summary'}
                </button>
            </div>
```

```
                {/* Right side: Summary and actions */}
                <div className="flex flex-col gap-4 min-h-0">
                    <div className="flex justify-between items-center">
                        <h3 className="text-lg font-bold">Generated Summary</h3>
                        {summary && (
                             <button
                                onClick={() => downloadFile(`${summary.title}\n\n${summary.summary}\n\n${summary
                                .changes.join('\n')}`, `pr_${summary.title.slice(0,10)}.md`, 'text/markdown')}
                                disabled={!summary}
                                className="btn-primary px-3 py-1 text-sm disabled:bg-gray-400"
                            >
                                Download .md
                           </button>
                        )}
                    </div>
                    <div className="flex-grow bg-surface p-4 border rounded-lg overflow-y-auto">
                        {isLoading && <div className="flex justify-center items-center
                        h-full"><LoadingSpinner/></div>}
                        {error && <p className="text-red-500">{error}</p>}
                        {summary && (
                            <div className="space-y-4">
                                <input type="text" value={summary.title} readOnly className="w-full p-2 bg-
                                background border rounded font-bold"/>
                                <div className="p-2 bg-background border rounded space-y-2">
                                    <h4 className="font-semibold">Summary</h4>
                                    <p className="text-sm">{summary.summary}</p>
                                    <h4 className="font-semibold">Changes</h4>
                                    <ul className="list-disc list-inside text-sm">
                                        {summary.changes.map((c, i) => <li key={i}>{c}</li>)}
                                    </ul>
                                </div>
                            </div>
                        )}
                    </div>
                     <button
                        onClick={handleExportToDocs}
                        disabled={isExporting || !summary || !user}
                        className="btn-primary w-full mt-4 flex items-center justify-center gap-2 py-2"
                        title={!user ? "Sign in to export to Google Docs" : !summary ? "Generate summary
                        first" : "Export a full tech spec to Google Docs"}
                    >
                        {isExporting ? <LoadingSpinner/> : <DocumentIcon/>} Export Tech Spec to Docs
                    </button>
                </div>
            </div>
        </div>
    );
};

// ===== ProjectExplorer_55.tsx =====

import React, { useState, useEffect, useCallback } from 'react';
import { useGlobalState } from '../../contexts/GlobalStateContext.tsx';
import { useNotification } from '../../contexts/NotificationContext.tsx';
import { initializeOctokit } from '../../services/authService.ts';
import { getDecryptedCredential } from '../../services/vaultService.ts';
import { getRepos, getRepoTree, getFileContent, commitFiles } from
'../../services/githubService.ts';
// FIX: Corrected import path for ai services.
import { generateCommitMessageStream, answerProjectQuestion,
generateNewFilesForProject } from '../../services/index.ts';
import type { Repo, FileNode, GeneratedFile } from '../../types.ts';
```

```
// FIX: Corrected import path for icons.
import { FolderIcon, DocumentIcon, SparklesIcon, XMarkIcon } from
'../icons/index.ts';
import { LoadingSpinner, MarkdownRenderer } from '../shared/index.tsx';
import * as Diff from 'diff';

const FileTree: React.FC<{ node: FileNode, onFileSelect: (path: string, name:
string) => void, activePath: string | null }> = ({ node, onFileSelect,
activePath }) => {
    const [isOpen, setIsOpen] = useState(true);

    if (node.type === 'file') {
        const isActive = activePath === node.path;
        return (
            <div
                className={`flex items-center space-x-2 pl-4 py-1 cursor-pointer rounded
                ${isActive ? 'bg-primary/10 text-primary' : 'hover:bg-gray-100 dark:hover:bg-
                slate-700'}`}
                onClick={() => onFileSelect(node.path, node.name)}
            >
                <DocumentIcon />
                <span>{node.name}</span>
            </div>
        );
    }

    return (
        <div>
            <div
                className="flex items-center space-x-2 py-1 cursor-pointer hover:bg-gray-100
                dark:hover:bg-slate-700 rounded"
                onClick={() => setIsOpen(!isOpen)}
            >
                <div className={`transform transition-transform ${isOpen ? 'rotate-90' :
                ''}`}>■</div>
                <FolderIcon />
                <span className="font-semibold">{node.name}</span>
            </div>
            {isOpen && node.children && (
                <div className="pl-4 border-l border-border ml-3">
                    {node.children.map(child => <FileTree key={child.path} node={child}
                    onFileSelect={onFileSelect} activePath={activePath} />)}
                </div>
            )}
        </div>
    );
};

const GeneratedFilesModal: React.FC<{
    files: GeneratedFile[];
    onClose: () => void;
    onCommit: (commitMessage: string) => void;
    isCommitting: boolean;
}> = ({ files, onClose, onCommit, isCommitting }) => {
    const [commitMessage, setCommitMessage] = useState('');
    const [activeFile, setActiveFile] = useState(files[0]);

    useEffect(() => {
        const generateMessage = async () => {
            const diffContext = files.map(f => `File:
            ${f.filePath}\n\n${f.content}`).join('\n---\n');
            const stream = generateCommitMessageStream(diffContext);
```

```
            let message = '';
            for await (const chunk of stream) {
                message += chunk;
                setCommitMessage(message);
            }
        };
        generateMessage();
    }, [files]);

    return (
         <div className="fixed inset-0 bg-black/60 z-50 flex items-center justify-center"
         onClick={onClose}>
            <div className="bg-surface rounded-lg shadow-xl w-full max-w-4xl h-[80vh] flex
            flex-col" onClick={e => e.stopPropagation()}>
                <header className="flex justify-between items-center p-4 border-b">
                    <h2 className="text-lg font-bold">Generated Files</h2>
                    <button onClick={onClose}><XMarkIcon/></button>
                </header>
                <div className="flex-grow flex min-h-0">
                    <aside className="w-1/3 border-r p-2 overflow-y-auto">
                        <ul>
                            {files.map(f => (
                                <li key={f.filePath} onClick={() => setActiveFile(f)} className={`p-2 rounded
                                cursor-pointer ${activeFile.filePath === f.filePath ? 'bg-primary/10' :
                                ''}`}>{f.filePath}</li>
                            ))}
                        </ul>
                    </aside>
                    <main className="w-2/3 overflow-y-auto">
                        <MarkdownRenderer content={'```\n' + activeFile.content + '\n```'} />
                    </main>
                </div>
                <footer className="p-4 border-t flex gap-4 items-center">
                    <input type="text" value={commitMessage} onChange={e =>
                    setCommitMessage(e.target.value)} placeholder="Commit message..."
                    className="flex-grow p-2 bg-background border rounded"/>
                    <button onClick={() => onCommit(commitMessage)} disabled={isCommitting}
                    className="btn-primary px-4 py-2 flex items-center justify-center
                    min-w-[120px]">{isCommitting ? <LoadingSpinner/> : 'Commit to Repo'}</button>
                </footer>
            </div>
        </div>
    )
}

const AiAssistantPanel: React.FC<{
    projectFiles: FileNode | null;
    onFilesGenerated: (files: GeneratedFile[]) => void;
}> = ({ projectFiles, onFilesGenerated }) => {
    const [isOpen, setIsOpen] = useState(false);
    const [tab, setTab] = useState<'ask' | 'generate'>('ask');
    const [prompt, setPrompt] = useState('');
    const [isLoading, setIsLoading] = useState(false);
    const [result, setResult] = useState('');

    const handleSubmit = async () => {
        if (!prompt.trim() || !projectFiles) return;
        setIsLoading(true);
        setResult('');

        try {
            if (tab === 'ask') {
```

```
                const stream = answerProjectQuestion(prompt, projectFiles);
                let fullResponse = '';
                for await (const chunk of stream) {
                    fullResponse += chunk;
                    setResult(fullResponse);
                }
            } else {
                const files = await generateNewFilesForProject(prompt, projectFiles);
                onFilesGenerated(files);
            }
        } catch (e) {
            setResult(`Error: ${e instanceof Error ? e.message : 'Unknown error'}`);
        } finally {
            setIsLoading(false);
            setPrompt('');
        }
    };

    return (
        <div className="flex-shrink-0 bg-surface border-t border-border">
            <button onClick={() => setIsOpen(!isOpen)} className="w-full p-2 text-left text-
            sm font-semibold flex items-center justify-between">
                <span><SparklesIcon/> AI Project Assistant</span>
                <span>{isOpen ? '▼' : '▲'}</span>
            </button>
            {isOpen && (
                <div className="p-4 border-t">
                    <div className="flex border-b mb-2">
                        <button onClick={() => setTab('ask')} className={`px-3 py-1 text-sm ${tab ===
                        'ask' ? 'border-b-2 border-primary' : ''}`}>Ask AI</button>
                        <button onClick={() => setTab('generate')} className={`px-3 py-1 text-sm ${tab
                        === 'generate' ? 'border-b-2 border-primary' : ''}`}>Generate Files</button>
                    </div>
                    {result && tab === 'ask' && <div className="p-2 bg-background rounded mb-2
                    max-h-48 overflow-y-auto"><MarkdownRenderer content={result} /></div>}
                    <div className="flex gap-2">
                        <input value={prompt} onChange={e => setPrompt(e.target.value)} onKeyDown={e =>
                        e.key === 'Enter' && handleSubmit()} placeholder={tab === 'ask' ? 'e.g., Where
                        is the auth logic?' : 'e.g., Create a new utility file with a date formatting
                        function.'} className="flex-grow p-2 text-sm bg-background border rounded"/>
                        <button onClick={handleSubmit} disabled={isLoading} className="btn-primary px-4
                        py-1 text-sm">{isLoading ? <LoadingSpinner/> : 'Send'}</button>
                    </div>
                </div>
            )}
        </div>
    )
};


export const ProjectExplorer: React.FC = () => {
    const { state, dispatch } = useGlobalState();
    const { user, githubUser, selectedRepo, projectFiles } = state;
    const { addNotification } = useNotification();
    const [repos, setRepos] = useState<Repo[]>([]);
    const [isLoading, setIsLoading] = useState<'repos' | 'tree' | 'file' | 'commit'
    | null>(null);
    const [error, setError] = useState('');
    const [activeFile, setActiveFile] = useState<{ path: string; name: string;
    originalContent: string; editedContent: string} | null>(null);
    const [generatedFiles, setGeneratedFiles] = useState<GeneratedFile[] |
    null>(null);
```

```
const getApiClient = useCallback(async () => {
    if (!user) {
        throw new Error("You must be logged in to use the Project Explorer.");
    }
    const token = await getDecryptedCredential('github_pat');
    if (!token) {
        throw new Error("GitHub token not found. Please add it on the Connections
        page.");
    }
    return initializeOctokit(token);
}, [user]);


useEffect(() => {
    const loadRepos = async () => {
        if (user && githubUser) {
            setIsLoading('repos');
            setError('');
            try {
                const octokit = await getApiClient();
                const userRepos = await getRepos(octokit);
                setRepos(userRepos);
            } catch (err) {
                setError(err instanceof Error ? err.message : 'Failed to load repositories');
            } finally {
                setIsLoading(null);
            }
        } else {
            setRepos([]);
        }
    };
    loadRepos();
}, [user, githubUser, getApiClient]);

useEffect(() => {
    const loadTree = async () => {
         if (selectedRepo && user && githubUser) {
            setIsLoading('tree');
            setError('');
            setActiveFile(null);
            try {
                const octokit = await getApiClient();
                const tree = await getRepoTree(octokit, selectedRepo.owner, selectedRepo.repo);
                dispatch({ type: 'LOAD_PROJECT_FILES', payload: tree });
            } catch (err) {
                 setError(err instanceof Error ? err.message : 'Failed to load repository tree');
            } finally {
                setIsLoading(null);
            }
        }
    };
    loadTree();
}, [selectedRepo, user, githubUser, dispatch, getApiClient]);

const handleFileSelect = async (path: string, name: string) => {
    if (!selectedRepo) return;
    setIsLoading('file');
    try {
        const octokit = await getApiClient();
        const content = await getFileContent(octokit, selectedRepo.owner,
        selectedRepo.repo, path);
        setActiveFile({ path, name, originalContent: content, editedContent: content });
```

```
    } catch (err) {
        setError((err as Error).message);
    } finally {
        setIsLoading(null);
    }
};

const handleCommit = async () => {
    if (!activeFile || !selectedRepo || activeFile.originalContent ===
    activeFile.editedContent) return;

    setIsLoading('commit');
    setError('');
    try {
        const diff = Diff.createPatch(activeFile.path, activeFile.originalContent,
        activeFile.editedContent);

        const stream = generateCommitMessageStream(diff);
        let commitMessage = '';
        for await (const chunk of stream) { commitMessage += chunk; }

        const finalMessage = window.prompt("Confirm or edit commit message:",
        commitMessage);
        if (!finalMessage) {
            setIsLoading(null);
            return;
        }

        const octokit = await getApiClient();
        await commitFiles(
            octokit,
            selectedRepo.owner,
            selectedRepo.repo,
            [{ path: activeFile.path, content: activeFile.editedContent }],
            finalMessage
        );

        addNotification(`Successfully committed to ${selectedRepo.repo}`, 'success');
        setActiveFile(prev => prev ? { ...prev, originalContent: prev.editedContent } :
        null);

    } catch (err) {
        const message = err instanceof Error ? err.message : 'Failed to commit changes';
        setError(message);
        addNotification(message, 'error');
    } finally {
        setIsLoading(null);
    }
};

 const handleCommitGeneratedFiles = async (commitMessage: string) => {
    if (!generatedFiles || !selectedRepo) return;
    setIsLoading('commit');
    try {
         const octokit = await getApiClient();
         await commitFiles(
            octokit,
            selectedRepo.owner,
            selectedRepo.repo,
            generatedFiles.map(f => ({ path: f.filePath, content: f.content })),
            commitMessage
         );
```

```
         addNotification(`Successfully committed ${generatedFiles.length} new files!`,
         'success');
         setGeneratedFiles(null);
         // Reload tree
         const tree = await getRepoTree(octokit, selectedRepo.owner, selectedRepo.repo);
         dispatch({ type: 'LOAD_PROJECT_FILES', payload: tree });
    } catch (err) {
        addNotification(err instanceof Error ? err.message : 'Failed to commit',
        'error');
    } finally {
        setIsLoading(null);
    }
};

if (!user) {
    return (
        <div className="h-full flex flex-col items-center justify-center text-center
        text-text-secondary p-4">
            <FolderIcon />
            <h2 className="text-lg font-semibold mt-2">Please Sign In</h2>
            <p>Sign in via the "Connections" tab to explore your repositories.</p>
        </div>
    );
}

if (!githubUser) {
     return (
        <div className="h-full flex flex-col items-center justify-center text-center
        text-text-secondary p-4">
            <FolderIcon />
            <h2 className="text-lg font-semibold mt-2">Connect to GitHub</h2>
            <p>Please go to the "Connections" tab and provide a Personal Access Token to
            explore your repositories.</p>
        </div>
    );
}

const hasChanges = activeFile ? activeFile.originalContent !==
activeFile.editedContent : false;

return (
    <div className="h-full flex flex-col text-text-primary">
        {generatedFiles && <GeneratedFilesModal files={generatedFiles} onClose={() =>
        setGeneratedFiles(null)} onCommit={handleCommitGeneratedFiles}
        isCommitting={isLoading === 'commit'}/>}
        <header className="p-4 border-b border-border flex-shrink-0">
            <h1 className="text-xl font-bold flex items-center"><FolderIcon /><span
            className="ml-3">Project Explorer</span></h1>
            <div className="mt-2">
                <select
                    value={selectedRepo ? `${selectedRepo.owner}/${selectedRepo.repo}` : ''}
                    onChange={e => {
                        const [owner, repo] = e.target.value.split('/');
                        dispatch({ type: 'SET_SELECTED_REPO', payload: { owner, repo } });
                    }}
                    className="w-full p-2 bg-surface border border-border rounded-md text-sm"
                >
                    <option value="" disabled>{isLoading === 'repos' ? 'Loading...' : 'Select a
                    repository'}</option>
                    {repos.map(r => <option key={r.id} value={r.full_name}>{r.full_name}</option>)}
                </select>
            </div>
```

```
                {error && <p className="text-red-500 text-xs mt-2">{error}</p>}
            </header>
            <div className="flex-grow flex min-h-0">
                <aside className="w-1/3 bg-background border-r border-border p-4 overflow-y-
                auto">
                    {isLoading === 'tree' && <div className="flex justify-center"><LoadingSpinner
                    /></div>}
                    {projectFiles && <FileTree node={projectFiles} onFileSelect={handleFileSelect}
                    activePath={activeFile?.path ?? null} />}
                </aside>
                <main className="flex-1 bg-surface flex flex-col">
                     <div className="flex justify-between items-center p-2 border-b border-border bg-
                     gray-50 dark:bg-slate-800">
                        <span className="text-sm font-semibold">{activeFile?.name || 'No file
                        selected'}</span>
                        <button onClick={handleCommit} disabled={!hasChanges || isLoading === 'commit'}
                        className="btn-primary px-4 py-1 text-sm flex items-center justify-center
                        min-w-[100px]">
                           {isLoading === 'commit' ? <LoadingSpinner/> : 'Commit'}
                        </button>
                     </div>
                     {isLoading === 'file' ? <div className="flex items-center justify-center
                     h-full"><LoadingSpinner /></div> :
                        <textarea
                            value={activeFile?.editedContent ?? 'Select a file to view its content.'}
                            onChange={e => setActiveFile(prev => prev ? { ...prev, editedContent:
                            e.target.value } : null)}
                            disabled={!activeFile}
                            className="w-full h-full p-4 text-sm font-mono bg-transparent resize-none
                            focus:outline-none"
                        />
                    }
                </main>
            </div>
            <AiAssistantPanel projectFiles={projectFiles}
            onFilesGenerated={setGeneratedFiles} />
        </div>
    );
};

// ===== AiCommitGenerator_55.tsx =====

import React, { useState, useCallback, useEffect } from 'react';
// FIX: Corrected import path for ai services.
import { generateCommitMessageStream } from '../../services/index.ts';
import { GitBranchIcon } from '../icons/index.ts';
import { LoadingSpinner } from '../shared/index.tsx';

const exampleDiff = `diff --git a/src/components/Button.tsx
b/src/components/Button.tsx
index 1b2c3d4..5e6f7g8 100644
--- a/src/components/Button.tsx
+++ b/src/components/Button.tsx
@@ -1,7 +1,7 @@
 import React from 'react';

 interface ButtonProps {
-  text: string;
+  label: string;
   onClick: () => void;
 }
`;
```

```
export const AiCommitGenerator: React.FC<{ diff?: string }> = ({ diff:
initialDiff }) => {
    const [diff, setDiff] = useState<string>(initialDiff || exampleDiff);
    const [message, setMessage] = useState<string>('');
    const [isLoading, setIsLoading] = useState<boolean>(false);
    const [error, setError] = useState<string>('');

    const handleGenerate = useCallback(async (diffToAnalyze: string) => {
        if (!diffToAnalyze.trim()) {
            setError('Please paste a diff to generate a message.');
            return;
        }
        setIsLoading(true);
        setError('');
        setMessage('');
        try {
            const stream = generateCommitMessageStream(diffToAnalyze);
            let fullResponse = '';
            for await (const chunk of stream) {
                fullResponse += chunk;
                setMessage(fullResponse);
            }
        } catch (err) {
            const errorMessage = err instanceof Error ? err.message : 'An unknown error
            occurred.';
            setError(`Failed to generate message: ${errorMessage}`);
        } finally {
            setIsLoading(false);
        }
    }, []);

    useEffect(() => {
        if (initialDiff) {
            setDiff(initialDiff);
            handleGenerate(initialDiff);
        }
    }, [initialDiff, handleGenerate]);

    const handleCopy = () => {
        navigator.clipboard.writeText(message);
    };

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8">
            <header className="mb-6">
                <h1 className="text-3xl flex items-center">
                    <GitBranchIcon />
                    <span className="ml-3">AI Commit Message Generator</span>
                </h1>
                <p className="text-slate-400 mt-1">Paste your diff and let Gemini craft the
                perfect commit message.</p>
            </header>
            <div className="flex-grow grid grid-cols-1 lg:grid-cols-2 gap-6 h-full overflow-
            hidden">
                <div className="flex flex-col h-full">
                    <label htmlFor="diff-input" className="text-sm font-medium text-slate-400
                    mb-2">Git

// ===== ScreenshotToComponent_57.tsx =====

import React, { useState, useCallback, useRef } from 'react';
// FIX: Corrected import path for ai services.
```

```
import { generateComponentFromImageStream } from '../../services/index.ts';
// FIX: Corrected import path for icons.
import { PhotoIcon, ArrowDownTrayIcon } from '../icons/index.ts';
import { LoadingSpinner, MarkdownRenderer } from '../shared/index.tsx';
import { fileToBase64, blobToDataURL, downloadFile } from
'../../services/fileUtils.ts';

export const ScreenshotToComponent: React.FC = () => {
    const [previewImage, setPreviewImage] = useState<string | null>(null);
    const [rawCode, setRawCode] = useState('');
    const [isLoading, setIsLoading] = useState(false);
    const [error, setError] = useState('');
    const fileInputRef = useRef<HTMLInputElement>(null);

    const handleGenerate = async (base64Image: string) => {
        setIsLoading(true);
        setError('');
        setRawCode('');
        try {
            const stream = generateComponentFromImageStream(base64Image);
            let fullResponse = '';
            for await (const chunk of stream) {
                fullResponse += chunk;
                setRawCode(fullResponse.replace(/^```(?:\w+\n)?/, '').replace(/```$/, ''));
            }
        } catch (err) {
            setError(err instanceof Error ? err.message : 'An unknown error occurred.');
        } finally {
            setIsLoading(false);
        }
    };

    const processImageBlob = async (blob: Blob) => {
        try {
            const [dataUrl, base64Image] = await Promise.all([blobToDataURL(blob),
            fileToBase64(blob as File)]);
            setPreviewImage(dataUrl);
            handleGenerate(base64Image);
        } catch (e) {
            setError('Could not process the image.');
        }
    };

    const handlePaste = useCallback(async (event: React.ClipboardEvent) => {
        const items = event.clipboardData.items;
        for (const item of items) {
            if (item.type.indexOf('image') !== -1) {
                const blob = item.getAsFile();
                if (blob) await processImageBlob(blob);
                return;
            }
        }
    }, []);

    const handleFileChange = async (event: React.ChangeEvent<HTMLInputElement>) => {
        const file = event.target.files?.[0];
        if (file) await processImageBlob(file);
    };

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
            <header className="mb-6"><h1 className="text-3xl font-bold flex items-
```

```
            center"><PhotoIcon /><span className="ml-3">AI Screenshot-to-
            Component</span></h1><p className="text-text-secondary mt-1">Paste or upload a
            screenshot of a UI element to generate React/Tailwind code.</p></header>
            <div className="flex-grow grid grid-cols-1 lg:grid-cols-2 gap-6 min-h-0">
                <div onPaste={handlePaste} className="flex flex-col items-center justify-center
                bg-surface p-6 rounded-lg border-2 border-dashed border-border focus:outline-
                none focus:border-primary overflow-y-auto" tabIndex={0}>
                    {previewImage ? (<img src={previewImage} alt="Pasted content" className="max-w-
                    full max-h-full object-contain rounded-md shadow-lg" />) : (<div
                    className="text-center text-text-secondary">
                            <h2 className="text-xl font-bold text-text-primary">Paste an image here</h2>
                            <p className="mb-2">(Cmd/Ctrl + V)</p>
                            <p className="text-sm">or</p>
                            <button onClick={() => fileInputRef.current?.click()} className="mt-2 btn-
                            primary px-4 py-2 text-sm">Upload File</button>
                            <input type="file" ref={fileInputRef} onChange={handleFileChange}
                            accept="image/*" className="hidden"/>
                        </div>)}
                </div>
                <div className="flex flex-col h-full">
                    <div className="flex justify-between items-center mb-2">
                        <label className="text-sm font-medium text-text-secondary">Generated
                        Code</label>
                        {rawCode && !isLoading && (
                            <div className="flex items-center gap-2">
                                <button onClick={() => navigator.clipboard.writeText(rawCode)} className="px-3
                                py-1 bg-gray-100 text-xs rounded-md hover:bg-gray-200">Copy Code</button>
                                <button onClick={() => downloadFile(rawCode, 'Component.tsx',
                                'text/typescript')} className="flex items-center gap-1 px-3 py-1 bg-gray-100
                                text-xs rounded-md hover:bg-gray-200">
                                    <ArrowDownTrayIcon className="w-4 h-4" /> Download
                                </button>
                            </div>
                        )}
                    </div>
                    <div className="flex-grow bg-background border border-border rounded-md
                    overflow-y-auto">
                        {isLoading && (<div className="flex items-center justify-center
                        h-full"><LoadingSpinner /></div>)}
                        {error && <p className="p-4 text-red-500">{error}</p>}
                        {rawCode && !isLoading && <MarkdownRenderer
                        content={`\`\`\`tsx\n${rawCode}\n\`\`\`} />}
                        {!isLoading && !rawCode && !error && (<div className="text-text-secondary h-full
                        flex items-center justify-center">Generated component code will appear
                        here.</div>)}
                    </div>
                </div>
            </div>
        </div>
    );
};

// ===== AiImageGenerator_55.tsx =====

import React, { useState, useCallback, useRef } from 'react';
// FIX: Corrected import path for ai services.
import { generateImage, generateImageFromImageAndText } from
'../../services/index.ts';
import { fileToBase64, blobToDataURL } from '../../services/fileUtils.ts';
// FIX: Corrected import path for icons.
import { ImageGeneratorIcon, SparklesIcon, ArrowDownTrayIcon, XMarkIcon } from
'../icons/index.ts';
```

```
import { LoadingSpinner } from '../shared/index.tsx';

const surprisePrompts = [
    'A majestic lion wearing a crown, painted in the style of Van Gogh.',
    'A futuristic cityscape on another planet with two moons in the sky.',
    'A cozy, magical library inside a giant tree.',
    'A surreal image of a ship sailing on a sea of clouds.',
    'An astronaut riding a space-themed bicycle on the moon.',
];

interface UploadedImage {
    base64: string;
    dataUrl: string;
    mimeType: string;
}

export const AiImageGenerator: React.FC = () => {
    const [prompt, setPrompt] = useState<string>('A photorealistic image of a
    futuristic city at sunset, with flying cars.');
    const [uploadedImage, setUploadedImage] = useState<UploadedImage | null>(null);
    const [generatedImageUrl, setGeneratedImageUrl] = useState<string | null>(null);
    const [isLoading, setIsLoading] = useState<boolean>(false);
    const [error, setError] = useState<string>('');
    const fileInputRef = useRef<HTMLInputElement>(null);

    const handleGenerate = useCallback(async () => {
        if (!prompt.trim()) {
            setError('Please enter a prompt to generate an image.');
            return;
        }
        setIsLoading(true);
        setError('');
        setGeneratedImageUrl(null);
        try {
            let resultUrl: string;
            if (uploadedImage) {
                resultUrl = await generateImageFromImageAndText(prompt, uploadedImage.base64,
                uploadedImage.mimeType);
            } else {
                resultUrl = await generateImage(prompt);
            }
            setGeneratedImageUrl(resultUrl);
        } catch (err) {
            const errorMessage = err instanceof Error ? err.message : 'An unknown error
            occurred.';
            setError(`Failed to generate image: ${errorMessage}`);
        } finally {
            setIsLoading(false);
        }
    }, [prompt, uploadedImage]);

    const handleSurpriseMe = () => {
        const randomPrompt = surprisePrompts[Math.floor(Math.random() *
        surprisePrompts.length)];
        setPrompt(randomPrompt);
    };

    const processImageBlob = async (blob: Blob) => {
        try {
            const [dataUrl, base64] = await Promise.all([
                blobToDataURL(blob),
                fileToBase64(blob as File)
```

```
        ]);
        setUploadedImage({ dataUrl, base64, mimeType: blob.type });
    } catch (e) {
        setError('Could not process the image.');
    }
};

const handlePaste = useCallback(async (event: React.ClipboardEvent) => {
    const items = event.clipboardData.items;
    for (const item of items) {
        if (item.type.indexOf('image') !== -1) {
            const blob = item.getAsFile();
            if (blob) {
                await processImageBlob(blob);
                return;
            }
        }
    }
}, []);

const handleFileChange = async (event: React.ChangeEvent<HTMLInputElement>) => {
    const file = event.target.files?.[0];
    if (file) {
        await processImageBlob(file);
    }
};

const handleDownload = () => {
    if (!generatedImageUrl) return;
    const link = document.createElement('a');
    link.href = generatedImageUrl;
    link.download = `${prompt.slice(0, 30).replace(/\s/g, '_')}.png`;
    document.body.appendChild(link);
    link.click();
    document.body.removeChild(link);
}

return (
    <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
        <header className="mb-6">
            <h1 className="text-3xl font-bold flex items-center">
                <ImageGeneratorIcon />
                <span className="ml-3">AI Image Generator</span>
            </h1>
            <p className="text-text-secondary mt-1">Generate images from text, or provide an
            image for inspiration.</p>
        </header>

        <div className="flex-grow grid grid-cols-1 lg:grid-cols-2 gap-6 min-h-0">
            {/* Left Column: Inputs */}
            <div className="flex flex-col gap-4">
                <div>
                    <label htmlFor="prompt-input" className="text-sm font-medium text-text-
                    secondary">Your Prompt</label>
                    <textarea
                        id="prompt-input"
                        value={prompt}
                        onChange={(e) => setPrompt(e.target.value)}
                        placeholder="e.g., A cute cat wearing a wizard hat"
                        className="w-full p-3 mt-1 rounded-md bg-surface border border-border
                        focus:ring-2 focus:ring-primary focus:outline-none resize-y"
                        rows={3}
```

```
        />
    </div>

    <div className="flex flex-col flex-grow min-h-[200px]">
         <label className="text-sm font-medium text-text-secondary mb-1">Inspiration
         Image (Optional)</label>
         <div onPaste={handlePaste} className="relative flex-grow flex flex-col items-
         center justify-center bg-surface p-4 rounded-lg border-2 border-dashed border-
         border focus:outline-none focus:border-primary" tabIndex={0}>
            {uploadedImage ? (
                <>
                    <img src={uploadedImage.dataUrl} alt="Uploaded content" className="max-w-full
                    max-h-full object-contain rounded-md shadow-lg" />
                    <button onClick={() => setUploadedImage(null)} className="absolute top-2 right-2
                    p-1 bg-black/30 text-white rounded-full hover:bg-black/50"><XMarkIcon
                    /></button>
                </>
            ) : (
                <div className="text-center text-text-secondary">
                    <h2 className="text-lg font-bold text-text-primary">Paste an image here</h2>
                    <p className="text-sm">(Cmd/Ctrl + V)</p>
                    <p className="text-xs my-1">or</p>
                    <button onClick={() => fileInputRef.current?.click()} className="text-sm font-
                    semibold text-primary hover:underline">Upload File</button>
                    <input type="file" ref={fileInputRef} onChange={handleFileChange}
                    accept="image/*" className="hidden"/>
                </div>
            )}
         </div>
    </div>

    <div className="flex gap-2">
        <button
            onClick={handleGenerate}
            disabled={isLoading}
            className="btn-primary w-full flex items-center justify-center px-6 py-3"
        >
            {isLoading ? <LoadingSpinner /> : 'Generate Image'}
        </button>
        <button
            onClick={handleSurpriseMe}
            disabled={isLoading}
            className="px-4 py-3 bg-surface border border-border rounded-md hover:bg-
            gray-100 transition-colors"
            title="Surprise Me!"
        >
            <SparklesIcon />
        </button>
    </div>
</div>

{/* Right Column: Output */}
<div className="flex flex-col h-full">
    <label className="text-sm font-medium text-text-secondary mb-2">Generated
    Image</label>
    <div className="flex-grow flex items-center justify-center bg-background
    border-2 border-dashed border-border rounded-lg p-4 relative overflow-auto">
        {isLoading && <LoadingSpinner />}
        {error && <p className="text-red-500 text-center">{error}</p>}
        {generatedImageUrl && !isLoading && (
            <>
                <img src={generatedImageUrl} alt={prompt || "Generated by AI"} className="max-w-
```

```
                                full max-h-full object-contain rounded-md shadow-lg" />
                                <button
                                  onClick={handleDownload}
                                  className="absolute top-4 right-4 p-2 bg-black/30 text-white rounded-full
                                  hover:bg-black/50 backdrop-blur-sm"
                                  title="Download Image"
                                >
                                    <ArrowDownTrayIcon />
                                </button>
                            </>
                        )}
                        {!isLoading && !generatedImageUrl && !error && (
                            <div className="text-center text-text-secondary">
                                <p>Your generated image will appear here.</p>
                            </div>
                        )}
                    </div>
                </div>
            </div>
        </div>
    );
};

// ===== XbrlConverter_55.tsx =====

import React, { useState, useCallback } from 'react';
// FIX: Corrected import path for ai services.
import { convertJsonToXbrlStream } from '../../services/index.ts';
// FIX: Corrected import path for icons.
import { XbrlConverterIcon } from '../icons/index.ts';
import { LoadingSpinner, MarkdownRenderer } from '../shared/index.tsx';

const exampleJson = `{
  "company": "ExampleCorp",
  "year": 2024,
  "quarter": 2,
  "revenue": {
    "amount": 1500000,
    "currency": "USD"
  },
  "profit": {
    "amount": 250000,
    "currency": "USD"
  }
}`;

export const XbrlConverter: React.FC<{ jsonInput?: string }> = ({ jsonInput:
initialJsonInput }) => {
    const [jsonInput, setJsonInput] = useState<string>(initialJsonInput ||
    exampleJson);
    const [xbrlOutput, setXbrlOutput] = useState<string>('');
    const [isLoading, setIsLoading] = useState<boolean>(false);
    const [error, setError] = useState<string>('');

    const handleConvert = useCallback(async (jsonToConvert: string) => {
        if (!jsonToConvert.trim()) {
            setError('Please enter valid JSON to convert.');
            return;
        }
        setIsLoading(true);
        setError('');
        setXbrlOutput('');
```

```
    try {
        const stream = convertJsonToXbrlStream(jsonToConvert);
        let fullResponse = '';
        for await (const chunk of stream) {
            fullResponse += chunk;
            setXbrlOutput(fullResponse);
        }
    } catch (err) {
        const errorMessage = err instanceof Error ? err.message : 'An unknown error
        occurred.';
        setError(`Failed to convert: ${errorMessage}`);
    } finally {
        setIsLoading(false);
    }
}, []);

return (
    <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
        <header className="mb-6">
            <h1 className="text-3xl font-bold flex items-center">
                <XbrlConverterIcon />
                <span className="ml-3">JSON to XBRL Converter</span>
            </h1>
            <p className="text-text-secondary mt-1">Convert JSON data into a simplified
            XBRL-like XML format using AI.</p>
        </header>
        <div className="flex-grow flex flex-col gap-4 min-h-0">
            <div className="flex flex-col flex-1 min-h-0">
                <label htmlFor="json-input" className="text-sm font-medium text-text-secondary
                mb-2">JSON Input</label>
                <textarea
                    id="json-input"
                    value={jsonInput}
                    onChange={(e) => setJsonInput(e.target.value)}
                    placeholder="Paste your JSON here..."
                    className="flex-grow p-4 bg-surface border border-border rounded-md resize-none
                    font-mono text-sm"
                />
            </div>
             <div className="flex-shrink-0">
                <button
                    onClick={() => handleConvert(jsonInput)}
                    disabled={isLoading}
                    className="btn-primary w-full max-w-xs mx-auto flex items-center justify-center
                    px-6 py-3"
                >
                    {isLoading ? <LoadingSpinner /> : 'Convert to XBRL'}
                </button>
            </div>
            <div className="flex flex-col flex-1 min-h-0">
                <label className="text-sm font-medium text-text-secondary mb-2">XBRL-like XML
                Output</label>
                <div className="relative flex-grow p-1 bg-background border border-border
                rounded-md overflow-y-auto">
                    {isLoading && !xbrlOutput && <div className="flex items-center justify-center
                    h-full"><LoadingSpinner /></div>}
                    {error && <p className="p-4 text-red-500">{error}</p>}
                    {xbrlOutput && <MarkdownRenderer content={'```xml\n' +
                    xbrlOutput.replace(/```xml\n|```/g, '') + '\n```'} />}
                    {!isLoading && xbrlOutput && <button onClick={() =>
                    navigator.clipboard.writeText(xbrlOutput)} className="absolute top-2 right-2
                    px-2 py-1 bg-gray-100 hover:bg-gray-200 rounded-md text-xs">Copy XML</button>}
```

```
                        {!isLoading && !xbrlOutput && !error && <div className="text-text-secondary
                        h-full flex items-center justify-center">Output will appear here.</div>}
                    </div>
                </div>
            </div>
        </div>
    );
};

// ===== ChangelogGenerator_57.tsx =====

import React, { useState, useCallback } from 'react';
// FIX: Corrected import path for ai services.
import { generateChangelogFromLogStream } from '../../services/index.ts';
// FIX: Corrected import path for icons.
import { GitBranchIcon } from '../icons/index.ts';
import { LoadingSpinner } from '../shared/index.tsx';
import { MarkdownRenderer } from '../shared/index.tsx';

const exampleLog = `commit 3a4b5c...
Author: Dev One <dev.one@example.com>
Date:   Mon Jul 15 11:30:00 2024 -0400

    feat: add user login page

commit 1a2b3c...
Author: Dev Two <dev.two@example.com>
Date:   Mon Jul 15 10:00:00 2024 -0400

    fix: correct typo in header
`;

export const ChangelogGenerator: React.FC = () => {
    const [log, setLog] = useState(exampleLog);
    const [changelog, setChangelog] = useState('');
    const [isLoading, setIsLoading] = useState(false);
    const [error, setError] = useState('');

    const handleGenerate = useCallback(async () => {
        if (!log.trim()) {
            setError('Please paste your git log output.');
            return;
        }
        setIsLoading(true);
        setError('');
        setChangelog('');
        try {
            const stream = generateChangelogFromLogStream(log);
            let fullResponse = '';
            for await (const chunk of stream) {
                fullResponse += chunk;
                setChangelog(fullResponse);
            }
        } catch (err) {
            setError(err instanceof Error ? err.message : 'An unknown error occurred.');
        } finally {
            setIsLoading(false);
        }
    }, [log]);

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
```

```
            <header className="mb-6">
                <h1 className="text-3xl font-bold flex items-center">
                    <GitBranchIcon />
                    <span className="ml-3">AI Changelog Generator</span>
                </h1>
                <p className="text-text-secondary mt-1">Generate a markdown changelog from your
                raw git log.</p>
            </header>
            <div className="flex-grow flex flex-col gap-4 min-h-0">
                <div className="flex flex-col flex-1 min-h-0">
                    <label htmlFor="commit-input" className="text-sm font-medium text-text-secondary
                    mb-2">Raw Git Log</label>
                    <textarea
                        id="commit-input"
                        value={log}
                        onChange={(e) => setLog(e.target.value)}
                        className="flex-grow p-4 bg-surface border border-border rounded-md resize-none
                        font-mono text-sm"
                    />
                </div>
                <div className="flex-shrink-0">
                    <button onClick={handleGenerate} disabled={isLoading} className="btn-primary
                    w-full max-w-xs mx-auto flex items-center justify-center px-6 py-3">
                        {isLoading ? <LoadingSpinner /> : 'Generate Changelog'}
                    </button>
                </div>
                <div className="flex flex-col flex-1 min-h-0">
                    <label className="text-sm font-medium text-text-secondary mb-2">Generated
                    Changelog.md</label>
                    <div className="relative flex-grow p-4 bg-background border border-border
                    rounded-md overflow-y-auto">
                        {isLoading && !changelog && <div className="flex items-center justify-center
                        h-full"><LoadingSpinner /></div>}
                        {error && <p className="text-red-500">{error}</p>}
                        {changelog && <MarkdownRenderer content={changelog} />}
                        {!isLoading && changelog && <button onClick={() =>
                        navigator.clipboard.writeText(changelog)} className="absolute top-2 right-2 px-2
                        py-1 bg-gray-100 text-xs rounded-md hover:bg-gray-200">Copy</button>}
                    </div>
                </div>
            </div>
        </div>
    );
};

// ===== AiCodingChallenge_55.tsx =====

import React, { useState, useCallback, useEffect } from 'react';
// FIX: Corrected import path for ai services.
import { generateCodingChallengeStream } from '../../services/index.ts';
import { BeakerIcon } from '../icons/index.ts';
import { LoadingSpinner } from '../shared/index.tsx';
import { MarkdownRenderer } from '../shared/index.tsx';

export const AiCodingChallenge: React.FC = () => {
    const [challenge, setChallenge] = useState<string>('');
    const [isLoading, setIsLoading] = useState<boolean>(false);
    const [error, setError] = useState<string>('');

    const handleGenerate = useCallback(async () => {
        setIsLoading(true);
        setError('');
```

```
        setChallenge('');
        try {
            const stream = generateCodingChallengeStream(null);
            let fullResponse = '';
            for await (const chunk of stream) {
                fullResponse += chunk;
                setChallenge(fullResponse);
            }
        } catch (err) {
            const errorMessage = err instanceof Error ? err.message : 'An unknown error
            occurred.';
            setError(`Failed to generate challenge: ${errorMessage}`);
        } finally {
            setIsLoading(false);
        }
    }, []);

    useEffect(() => {
        // Generate a challenge on initial load for a better user experience
        handleGenerate();
    // eslint-disable-next-line react-hooks/exhaustive-deps
    }, []);

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
            <header className="mb-6 flex justify-between items-center">
                <div>
                    <h1 className="text-3xl font-bold flex items-center">
                        <BeakerIcon />
                        <span className="ml-3">AI Coding Challenge Generator</span>
                    </h1>
                    <p className="text-text-secondary mt-1">Generate a unique coding problem to test
                    your skills.</p>
                </div>
                <button
                    onClick={handleGenerate}
                    disabled={isLoading}
                    className="btn-primary flex items-center justify-center px-6 py-3"
                >
                    {isLoading ? <LoadingSpinner /> : 'Generate New Challenge'}
                </button>
            </header>
            <div className="flex-grow p-4 bg-surface border border-border rounded-md
            overflow-y-auto">
                {isLoading && (
                     <div className="flex items-center justify-center h-full">
                        <LoadingSpinner />
                     </div>
                )}
                {error && <p className="text-red-500">{error}</p>}
                {challenge && !isLoading && (
                    <MarkdownRenderer content={challenge} />
                )}
                 {!isLoading && !challenge && !error && (
                    <div className="text-text-secondary h-full flex items-center justify-center">
                        Click "Generate New Challenge" to start.
                    </div>
                )}
            </div>
        </div>
    );
};
```

```
// ===== SecurityScanner_45.tsx =====

import React, { useState } from 'react';
// FIX: Corrected import path for ai services.
import { analyzeCodeForVulnerabilities } from '../../services/index.ts';
import { runStaticScan, SecurityIssue } from
'../../services/security/staticAnalysisService.ts';
import type { SecurityVulnerability } from '../../types.ts';
// FIX: Corrected import path for icons.
import { ShieldCheckIcon, SparklesIcon } from '../icons/index.ts';
import { LoadingSpinner, MarkdownRenderer } from '../shared/index.tsx';

const exampleCode = `function UserProfile({ user }) {
  // TODO: remove this temporary api key
  const API_KEY = "sk-xxxxxxxxxxxxxxxxxxxxxxxxxxxx";
  const userContent = user.bio; // This might contain malicious scripts

  return (
    <div>
      <h2>{user.name}</h2>
      <div dangerouslySetInnerHTML={{ __html: userContent }} />
    </div>
  );
}`;

export const SecurityScanner: React.FC = () => {
    const [code, setCode] = useState(exampleCode);
    const [localIssues, setLocalIssues] = useState<SecurityIssue[]>([]);
    const [aiIssues, setAiIssues] = useState<SecurityVulnerability[]>([]);
    const [isLoading, setIsLoading] = useState(false);
    const [error, setError] = useState('');

    const handleScan = async () => {
        if (!code.trim()) {
            setError('Please enter code to scan.');
            return;
        }
        setIsLoading(true);
        setError('');
        setLocalIssues([]);
        setAiIssues([]);
        try {
            // Run local scan first
            const staticIssues = runStaticScan(code);
            setLocalIssues(staticIssues);

            // Then run AI scan
            const geminiIssues = await analyzeCodeForVulnerabilities(code);
            setAiIssues(geminiIssues);

        } catch (err) {
            setError(err instanceof Error ? err.message : 'An error occurred during
            scanning.');
        } finally {
            setIsLoading(false);
        }
    };

    const SeverityBadge: React.FC<{ severity: string }> = ({ severity }) => {
        const colors: Record<string, string> = {
            'Critical': 'bg-red-500 text-white',
            'High': 'bg-red-400 text-white',
```

```
        'Medium': 'bg-yellow-400 text-yellow-900',
        'Low': 'bg-blue-400 text-white',
        'Informational': 'bg-gray-400 text-gray-900',
    };
    return <span className={`px-2 py-0.5 text-xs font-bold rounded-full
    ${colors[severity] || 'bg-gray-300'}`}>{severity}</span>
}

return (
    <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
        <header className="mb-6">
            <h1 className="text-3xl font-bold flex items-center"><ShieldCheckIcon /><span
            className="ml-3">AI Security Co-Pilot</span></h1>
            <p className="text-text-secondary mt-1">Find vulnerabilities in your code with
            static analysis and AI.</p>
        </header>
        <div className="flex-grow grid grid-cols-1 lg:grid-cols-2 gap-6 min-h-0">
            <div className="flex flex-col">
                <label className="text-sm mb-2">Code to Scan</label>
                <textarea value={code} onChange={e => setCode(e.target.value)} className="w-full
                flex-grow p-2 bg-surface border rounded font-mono text-xs" />
                <button onClick={handleScan} disabled={isLoading} className="btn-primary w-full
                mt-4 py-2 flex justify-center items-center gap-2">{isLoading ? <LoadingSpinner/>
                : 'Scan Code'}</button>
            </div>
            <div className="flex flex-col bg-surface p-4 border rounded-lg">
                <h3 className="text-lg font-bold mb-2">Scan Results</h3>
                {error && <p className="text-red-500">{error}</p>}
                <div className="flex-grow overflow-y-auto pr-2 space-y-4">
                    {isLoading && <div className="flex justify-center items-center
                    h-full"><LoadingSpinner/></div>}
                    {!isLoading && localIssues.length === 0 && aiIssues.length === 0 && <p
                    className="text-text-secondary text-center mt-8">No issues found. Run a scan to
                    begin.</p>}

                    {localIssues.length > 0 && <div>
                        <h4 className="font-semibold text-sm mb-1">Static Analysis Findings</h4>
                        {localIssues.map((issue, i) => <div key={`local-${i}`} className="p-2 bg-
                        background border rounded mb-2"><p className="font-bold flex items-center
                        gap-2">{issue.type} <SeverityBadge severity={issue.severity} /></p><p
                        className="text-xs">Line {issue.line}: {issue.description}</p></div>)}
                    </div>}

                     {aiIssues.length > 0 && <div>
                        <h4 className="font-semibold text-sm mb-1 flex items-center
                        gap-1"><SparklesIcon/> AI-Powered Findings</h4>
                        {aiIssues.map((issue, i) => (
                            <details key={`ai-${i}`} className="p-2 bg-background border rounded mb-2">
                                <summary className="cursor-pointer font-bold flex items-center
                                gap-2">{issue.vulnerability} <SeverityBadge severity={issue.severity}
                                /></summary>
                                <div className="mt-2 pt-2 border-t text-xs space-y-2">
                                    <p><strong>Description:</strong> {issue.description}</p>
                                    <p><strong>Mitigation:</strong> {issue.mitigation}</p>
                                    {issue.exploitSuggestion && (
                                        <div>
                                            <strong>Exploit Simulation:</strong>
                                            <div className="mt-1 p-2 bg-gray-50 rounded">
                                                 <MarkdownRenderer content={'```bash\n' + issue.exploitSuggestion + '\n```'}/>
                                            </div>
                                        </div>
                                    )}
```

```
                                    </div>
                                </details>
                            ))}
                        </div>}
                    </div>
                </div>
            </div>
        </div>
    );
};

// ===== OneClickRefactor_30.tsx =====

import React, { useState, useCallback } from 'react';
import * as Diff from 'diff';
// FIX: Corrected import path for ai services.
import { refactorForPerformance, refactorForReadability, generateJsDoc,
convertToFunctionalComponent } from '../../services/index.ts';
// FIX: Corrected import path for icons.
import { SparklesIcon } from '../icons/index.ts';
import { LoadingSpinner } from '../shared/index.tsx';

type RefactorAction = 'readability' | 'performance' | 'jsdoc' | 'functional' |
'custom';

const exampleCode = `const MyComponent = ({ data }) => {
  // A less readable component
  let transformedData = [];
  for (let i = 0; i < data.length; i++) {
    if (data[i].value > 50) {
      let item = { ...data[i], status: 'high' };
      transformedData.push(item);
    }
  }
  return (
    <div>
      {transformedData.map(d => <p key={d.id}>{d.name}</p>)}
    </div>
  );
}`;

const DiffViewer: React.FC<{ oldCode: string, newCode: string }> = ({ oldCode,
newCode }) => {
    const diff = Diff.diffLines(oldCode, newCode);

    return (
        <pre className="whitespace-pre-wrap font-mono text-xs">
            {diff.map((part, index) => {
                const color = part.added ? 'bg-green-500/20' : part.removed ? 'bg-red-500/20' :
                'bg-transparent';
                return <div key={index} className={color}>{part.value}</div>;
            })}
        </pre>
    );
};


export const OneClickRefactor: React.FC = () => {
    const [code, setCode] = useState(exampleCode);
    const [refactoredCode, setRefactoredCode] = useState('');
    const [loadingAction, setLoadingAction] = useState<RefactorAction | null>(null);
    const handleRefactor = useCallback(async (action: RefactorAction) => {
```

```
        if (!code.trim()) return;
        setLoadingAction(action);
        setRefactoredCode('');

        let stream;
        switch(action) {
            case 'readability':
                stream = refactorForReadability(code);
                break;
            case 'performance':
                stream = refactorForPerformance(code);
                break;
            case 'jsdoc':
                stream = generateJsDoc(code);
                break;
            case 'functional':
                stream = convertToFunctionalComponent(code);
                break;
            default:
                setLoadingAction(null);
                return;
        }

        try {
            let fullResponse = '';
            for await (const chunk of stream) {
                fullResponse += chunk;
                setRefactoredCode(

// ===== WeeklyDigestGenerator_39.tsx =====

import React, { useState, useCallback, useEffect } from 'react';
import { generateWeeklyDigest } from '../../services/index.ts';
import { getCommitHistory } from '../../services/githubService.ts';
import { useNotification } from '../../contexts/NotificationContext.tsx';
import { useGlobalState } from '../../contexts/GlobalStateContext.tsx';
import { useOctokit } from '../../contexts/OctokitContext.tsx';
import { MailIcon, SparklesIcon } from '../icons.tsx';
import { LoadingSpinner } from '../shared/index.tsx';

const dummyTelemetry = {
    avgPageLoad: 120,
    errorRate: '0.5%',
    uptime: '99.98%'
};

export const WeeklyDigestGenerator: React.FC = () => {
    const { addNotification } = useNotification();
    const { state } = useGlobalState();
    const { selectedRepo } = state;
    const { octokit, reinitialize } = useOctokit();

    const [emailHtml, setEmailHtml] = useState('');
    const [isLoading, setIsLoading] = useState(false);

    // Re-check for Octokit client if it's not available initially
    useEffect(() => {
        if (!octokit) {
            reinitialize();
        }
    }, [octokit, reinitialize]);
    const handleGenerate = useCallback(async () => {
```

```
    if (!selectedRepo || !octokit) {
        addNotification('Please select a repository and ensure GitHub is connected.',
        'error');
        return;
    }

    setIsLoading(true);
    setEmailHtml('');
    try {
        const [owner, repo] = selectedRepo.full_name.split('/');
        const commits = await getCommitHistory(octokit, owner, repo);
        const commitLogs = commits.map(c => c.commit.message).join('\n');

        const html = await generateWeeklyDigest(commitLogs, dummyTelemetry);
        setEmailHtml(html);
        addNotification('Digest content generated!', 'success');
    } catch (e) {
        addNotification(e instanceof Error ? e.message : 'Failed to generate digest',
        'error');
    } finally {
        setIsLoading(false);
    }
}, [addNotification, octokit, selectedRepo]);


return (
    <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
        <header className="mb-6">
            <h1 className="text-3xl font-bold flex items-center"><MailIcon /><span
            className="ml-3">Weekly Digest Generator</span></h1>
            <p className="text-text-secondary mt-1">Generate an AI-powered weekly summary
            based on project data.</p>
        </header>

        <div className="flex-grow grid grid-cols-1 lg:grid-cols-2 gap-6 min-h-0">
            <div className="bg-surface p-4 border border-border rounded-lg flex flex-col
            items-center justify-center text-center">
                <h3 className="text-lg font-bold">Generate Digest</h3>
                <p className="text-sm text-text-secondary my-4">
                    This tool will use the commit history from your selected repository
                    ({selectedRepo ? selectedRepo.full_name : 'none selected'}) to generate a
                    summary. The send functionality has been removed due to updated permissions.
                </p>
                <div className="flex flex-col gap-4 w-full max-w-xs">
                    <button onClick={handleGenerate} disabled={isLoading || !selectedRepo ||
                    !octokit} className="btn-primary flex items-center justify-center gap-2 py-3">
                        {isLoading ? <LoadingSpinner /> : <><SparklesIcon /> Generate Digest</>}
                    </button>
                </div>
            </div>

            <div className="bg-surface p-4 border border-border rounded-lg flex flex-col">
                <h3 className="text-lg font-bold mb-2">Email Preview</h3>
                <div className="flex-grow bg-white border rounded overflow-hidden">
                    {isLoading && <div className="flex justify-center items-center
                    h-full"><LoadingSpinner /></div>}
                    {emailHtml && <iframe srcDoc={emailHtml} title="Email Preview" className="w-full
                    h-full" />}
                    {!isLoading && !emailHtml && <div className="flex justify-center items-center
                    h-full text-text-secondary">Preview will appear here.</div>}
                </div>
            </div>
```

```
            </div>
        </div>
    );
};

// ===== PrSummaryGenerator_60.tsx =====

// This feature is an alias for the more comprehensively named
AiPullRequestAssistant.
// Re-exporting it here to keep the codebase DRY while satisfying the feature
registry.
export { AiPullRequestAssistant as PrSummaryGenerator } from
'./AiPullRequestAssistant.tsx';

// ===== ProjectMoodboard_60.tsx =====

import React, { useState } from 'react';
import { PhotoIcon } from '../icons.tsx';
import { useLocalStorage } from '../../hooks/useLocalStorage.ts';

interface MoodboardItem {
    id: number;
    text: string;
    x: number;
    y: number;
    color: string;
}

const colors = ['bg-yellow-200', 'bg-green-200', 'bg-blue-200', 'bg-pink-200',
'bg-purple-200', 'bg-orange-200'];
const textColors = ['text-yellow-800', 'text-green-800', 'text-blue-800', 'text-
pink-800', 'text-purple-800', 'text-orange-800'];

export const ProjectMoodboard: React.FC = () => {
    const [items, setItems] =
    useLocalStorage<MoodboardItem[]>('devcore_moodboard_items', []);
    const [dragging, setDragging] = useState<{ id: number; offsetX: number; offsetY:
    number } | null>(null);

    const addItem = () => {
        const newItem: MoodboardItem = {
            id: Date.now(),
            text: 'New Idea',
            x: 50,
            y: 50,
            color: colors[items.length % colors.length],
        };
        setItems([...items, newItem]);
    };

    const deleteItem = (id: number, e: React.MouseEvent) => {
        e.stopPropagation();
        setItems(items.filter((n) => n.id !== id));
    };

    const updateItem = (id: number, updates: Partial<MoodboardItem>) => {
        setItems(items.map((n) => n.id === id ? { ...n, ...updates } : n));
    };

    const onMouseDown = (e: React.MouseEvent<HTMLDivElement>, id: number) => {
        if ((e.target as HTMLElement).tagName === 'TEXTAREA') return;
        const noteElement = e.currentTarget;
```

```
        const rect = noteElement.getBoundingClientRect();
        setDragging({ id, offsetX: e.clientX - rect.left, offsetY: e.clientY - rect.top
        });
    };

    const onMouseMove = (e: React.MouseEvent<HTMLDivElement>) => {
        if (!dragging) return;
        const boardRect = e.currentTarget.getBoundingClientRect();
        updateItem(dragging.id, {
            x: e.clientX - dragging.offsetX - boardRect.left,
            y: e.clientY - dragging.offsetY - boardRect.top
        });
    };

    const onMouseUp = () => setDragging(null);

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
            <header className="mb-6 flex justify-between items-center">
                 <div>
                    <h1 className="text-3xl font-bold flex items-center"><PhotoIcon /><span
                    className="ml-3">Project Moodboard</span></h1>
                    <p className="text-text-secondary mt-1">A visual space to gather ideas, images,
                    and notes.</p>
                </div>
                <button onClick={addItem} className="btn-primary px-6 py-2">Add Item</button>
            </header>
            <div
                className="relative flex-grow bg-background border-2 border-dashed border-border
                rounded-lg overflow-hidden"
                onMouseMove={onMouseMove} onMouseUp={onMouseUp} onMouseLeave={onMouseUp}
            >
                {items.map((item) => (
                    <div
                        key={item.id}
                        className={`group absolute w-48 h-48 p-2 flex flex-col shadow-lg cursor-grab
                        active:cursor-grabbing rounded-md transition-transform duration-100 border
                        border-black/20 ${item.color} ${textColors[colors.indexOf(item.color)]}`}
                        style={{ top: item.y, left: item.x, transform: dragging?.id === item.id ?
                        'scale(1.05)' : 'scale(1)' }}
                        onMouseDown={e => onMouseDown(e, item.id)}
                    >
                        <button onClick={(e) => deleteItem(item.id, e)} className="absolute -top-2
                        -right-2 w-6 h-6 rounded-full bg-gray-700 text-white font-bold text-xs flex
                        items-center justify-center opacity-0 group-hover:opacity-100 hover:bg-red-500
                        transition-all">&times;</button>
                        <textarea
                            value={item.text}
                            onChange={(e) => updateItem(item.id, { text: e.target.value })}
                            className="w-full h-full bg-transparent resize-none focus:outline-none font-
                            medium p-1"
                        />
                    </div>
                ))}
            </div>
        </div>
    );
};

// ===== ProjectExplorer_58.tsx =====

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

```
import { useGlobalState } from '../../contexts/GlobalStateContext.tsx';
import { useNotification } from '../../contexts/NotificationContext.tsx';
import { useOctokit } from '../../contexts/OctokitContext.tsx';
import { getRepos, getRepoTree, getFileContent, commitFiles } from
'../../services/githubService.ts';
import { generateCommitMessageStream } from '../../services/index.ts';
import type { Repo, FileNode } from '../../types.ts';
import { FolderIcon, DocumentIcon } from '../icons.tsx';
import { LoadingSpinner } from '../shared/index.tsx';
import * as Diff from 'diff';

const FileTree: React.FC<{ node: FileNode, onFileSelect: (path: string, name:
string) => void, activePath: string | null }> = ({ node, onFileSelect,
activePath }) => {
    const [isOpen, setIsOpen] = useState(true);

    if (node.type === 'file') {
        const isActive = activePath === node.path;
        return (
            <div
                className={`flex items-center space-x-2 pl-4 py-1 cursor-pointer rounded
                ${isActive ? 'bg-primary/10 text-primary' : 'hover:bg-gray-100 dark:hover:bg-
                slate-700'}`}
                onClick={() => onFileSelect(node.path, node.name)}
            >
                <DocumentIcon />
                <span>{node.name}</span>
            </div>
        );
    }

    return (
        <div>
            <div
                className="flex items-center space-x-2 py-1 cursor-pointer hover:bg-gray-100
                dark:hover:bg-slate-700 rounded"
                onClick={() => setIsOpen(!isOpen)}
            >
                <div className={`transform transition-transform ${isOpen ? 'rotate-90' :
                ''}`}>■</div>
                <FolderIcon />
                <span className="font-semibold">{node.name}</span>
            </div>
            {isOpen && node.children && (
                <div className="pl-4 border-l border-border ml-3">
                    {node.children.map(child => <FileTree key={child.path} node={child}
                    onFileSelect={onFileSelect} activePath={activePath} />)}
                </div>
            )}
        </div>
    );
};

export const ProjectExplorer: React.FC = () => {
    const { state, dispatch } = useGlobalState();
    const { user, githubUser, selectedRepo, projectFiles } = state;
    const { addNotification } = useNotification();
    const { octokit, reinitialize } = useOctokit();
    const [repos, setRepos] = useState<Repo[]>([]);
    const [isLoading, setIsLoading] = useState<'repos' | 'tree' | 'file' | 'commit'
    | null>(null);
    const [error, setError] = useState('');
```

```
const [activeFile, setActiveFile] = useState<{ path: string; name: string;
originalContent: string; editedContent: string} | null>(null);

const handleApiError = useCallback((err: any) => {
    if (err.status === 401) {
        dispatch({ type: 'SET_GITHUB_USER', payload: null });
        addNotification('GitHub token is invalid or expired. Please update it in the
        Connections Hub.', 'error');
        setError('GitHub authentication failed. Please update your token.');
    } else {
        setError(err instanceof Error ? err.message : 'An unexpected error occurred.');
    }
}, [dispatch, addNotification]);

useEffect(() => {
    if (!octokit && githubUser) {
        reinitialize();
    }
}, [octokit, githubUser, reinitialize]);

useEffect(() => {
    const loadRepos = async () => {
        if (user && githubUser && octokit) {
            setIsLoading('repos');
            setError('');
            try {
                const userRepos = await getRepos(octokit);
                setRepos(userRepos);
            } catch (err) {
                handleApiError(err);
            } finally {
                setIsLoading(null);
            }
        } else {
            setRepos([]);
        }
    };
    loadRepos();
}, [user, githubUser, octokit, handleApiError]);

const loadTree = useCallback(async (repoToLoad: { owner: { login: string },
name: string, full_name: string }) => {
    if (user && githubUser && octokit) {
        setIsLoading('tree');
        setError('');
        setActiveFile(null);
        try {
            const tree = await getRepoTree(octokit, repoToLoad.owner.login,
            repoToLoad.name);
            dispatch({ type: 'LOAD_PROJECT_FILES', payload: tree });
        } catch (err) {
            handleApiError(err);
        } finally {
            setIsLoading(null);
        }
    }
}, [user, githubUser, octokit, dispatch, handleApiError]);

// Re-fetches the tree if a repo is selected from a previous session
useEffect(() => {
    if (selectedRepo && octokit && (!projectFiles || projectFiles.name !==
    selectedRepo.repo)) {
```

```
          loadTree({
             name: selectedRepo.repo,
             full_name: selectedRepo.full_name,
             owner: { login: selectedRepo.owner }
          });
      }
  }, [selectedRepo, projectFiles, octokit, loadTree]);

  const handleFileSelect = async (path: string, name: string) => {
      if (!selectedRepo || !octokit) return;
      setIsLoading('file');
      try {
          const content = await getFileContent(octokit, selectedRepo.owner,
          selectedRepo.repo, path);
          setActiveFile({ path, name, originalContent: content, editedContent: content });
      } catch (err) {
          handleApiError(err);
      } finally {
          setIsLoading(null);
      }
  };

  const handleCommit = async () => {
      if (!activeFile || !selectedRepo || !octokit || activeFile.originalContent ===
      activeFile.editedContent) return;

      setIsLoading('commit');
      setError('');
      try {
          const diff = Diff.createPatch(activeFile.path, activeFile.originalContent,
          activeFile.editedContent);

          const stream = generateCommitMessageStream(diff);
          let commitMessage = '';
          for await (const chunk of stream) { commitMessage += chunk; }

          const finalMessage = window.prompt("Confirm or edit commit message:",
          commitMessage);
          if (!finalMessage) {
              setIsLoading(null);
              return;
          }

          await commitFiles(
              octokit,
              selectedRepo.owner,
              selectedRepo.repo,
              [{ path: activeFile.path, content: activeFile.editedContent }],
              finalMessage
          );

          addNotification(`Successfully committed to ${selectedRepo.repo}`, 'success');
          setActiveFile(prev => prev ? { ...prev, originalContent: prev.editedContent } :
          null);

      } catch (err) {
          handleApiError(err);
      } finally {
          setIsLoading(null);
      }
  };
  if (!user) {
```

```
    return (
        <div className="h-full flex flex-col items-center justify-center text-center
        text-text-secondary p-4">
            <FolderIcon />
            <h2 className="text-lg font-semibold mt-2">Please Sign In</h2>
            <p>Sign in to explore your repositories.</p>
        </div>
    );
}

if (!githubUser) {
     return (
        <div className="h-full flex flex-col items-center justify-center text-center
        text-text-secondary p-4">
            <FolderIcon />
            <h2 className="text-lg font-semibold mt-2">Connect to GitHub</h2>
            <p>Please go to the "Connections" tab and provide a Personal Access Token to
            explore your repositories.</p>
        </div>
    );
}

const hasChanges = activeFile ? activeFile.originalContent !==
activeFile.editedContent : false;

return (
    <div className="h-full flex flex-col text-text-primary">
        <header className="p-4 border-b border-border flex-shrink-0">
            <h1 className="text-xl font-bold flex items-center"><FolderIcon /><span
            className="ml-3">Project Explorer</span></h1>
            <div className="mt-2">
                <select
                    value={selectedRepo?.full_name ?? ''}
                    onChange={e => {
                        const repo = repos.find(r => r.full_name === e.target.value);
                        if (repo) {
                            dispatch({ type: 'SET_SELECTED_REPO', payload: { owner: repo.owner.login, repo:
                            repo.name, full_name: repo.full_name, name: repo.name } });
                        }
                    }}
                    className="w-full p-2 bg-surface border border-border rounded-md text-sm"
                >
                    <option value="" disabled>{isLoading === 'repos' ? 'Loading...' : 'Select a
                    repository'}</option>
                    {repos.map(r => <option key={r.id} value={r.full_name}>{r.full_name}</option>)}
                </select>
            </div>
            {error && <p className="text-red-500 text-xs mt-2">{error}</p>}
        </header>
        <div className="flex-grow flex min-h-0">
            <aside className="w-1/3 bg-background border-r border-border p-4 overflow-y-
            auto">
                {isLoading === 'tree' && <div className="flex justify-center"><LoadingSpinner
                /></div>}
                {projectFiles && <FileTree node={projectFiles} onFileSelect={handleFileSelect}
                activePath={activeFile?.path ?? null} />}
            </aside>
            <main className="flex-1 bg-surface flex flex-col">
                 <div className="flex justify-between items-center p-2 border-b border-border bg-
                 gray-50 dark:bg-slate-800">
                    <span className="text-sm font-semibold">{activeFile?.name || 'No file
                    selected'}</span>
```

```
                        <button onClick={handleCommit} disabled={!hasChanges || isLoading === 'commit'}
                        className="btn-primary px-4 py-1 text-sm flex items-center justify-center
                        min-w-[100px]">
                           {isLoading === 'commit' ? <LoadingSpinner/> : 'Commit'}
                        </button>
                    </div>
                    {isLoading === 'file' ? <div className="flex items-center justify-center
                    h-full"><LoadingSpinner /></div> :
                        <textarea
                            value={activeFile?.editedContent ?? 'Select a file to view its content.'}
                            onChange={e => setActiveFile(prev => prev ? { ...prev, editedContent:
                            e.target.value } : null)}
                            disabled={!activeFile}
                            className="w-full h-full p-4 text-sm font-mono bg-transparent resize-none
                            focus:outline-none"
                        />
                    }
                </main>
            </div>
        </div>
    );
};

// ===== PrGenerator_60.tsx =====

// This feature is an alias for the more comprehensively named
AiPullRequestAssistant.
// Re-exporting it here to keep the codebase DRY while satisfying the feature
registry.
export { AiPullRequestAssistant as PrGenerator } from
'./AiPullRequestAssistant.tsx';

// ===== FontPreviewPicker_60.tsx =====

import React, { useState, useEffect } from 'react';
import { TypographyLabIcon } from '../icons.tsx';

const popularFonts = [
    'Roboto', 'Open Sans', 'Lato', 'Montserrat', 'Oswald', 'Source Sans Pro',
    'Raleway', 'Poppins', 'Nunito', 'Merriweather',
    'Playfair Display', 'Lora', 'Noto Sans', 'Ubuntu', 'PT Sans', 'Slabo 27px',
    'Great Vibes', 'EB Garamond'
];

export const FontPreviewPicker: React.FC = () => {
    const [text, setText] = useState('The quick brown fox jumps over the lazy
    dog.');
    const [fontSize, setFontSize] = useState(24);

    useEffect(() => {
        const fontsToLoad = popularFonts.join('|').replace(/ /g, '+');
        const linkId = 'font-picker-stylesheet';
        let link = document.getElementById(linkId) as HTMLLinkElement;
        if (!link) {
            link = document.createElement('link');
            link.id = linkId;
            link.rel = 'stylesheet';
            document.head.appendChild(link);
        }
        link.href =
        `https://fonts.googleapis.com/css?family=${fontsToLoad}:400,700&display=swap`;
    }, []);
```

```
    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
            <header className="mb-6">
                <h1 className="text-3xl font-bold flex items-center">
                    <TypographyLabIcon />
                    <span className="ml-3">Font Preview Picker</span>
                </h1>
                <p className="text-text-secondary mt-1">Enter your text and see how it looks
                with different fonts.</p>
            </header>

            <div className="flex flex-col md:flex-row gap-4 mb-6">
                <div className="flex-grow">
                    <label htmlFor="preview-text" className="text-sm font-medium">Preview
                    Text</label>
                    <input
                        id="preview-text"
                        type="text"
                        value={text}
                        onChange={e => setText(e.target.value)}
                        className="w-full mt-1 p-2 bg-surface border border-border rounded-md"
                    />
                </div>
                <div>
                    <label htmlFor="font-size" className="text-sm font-medium">Font Size
                    ({fontSize}px)</label>
                    <input
                        id="font-size"
                        type="range"
                        min="12"
                        max="72"
                        value={fontSize}
                        onChange={e => setFontSize(Number(e.target.value))}
                        className="w-full mt-1"
                    />
                </div>
            </div>

            <div className="flex-grow bg-surface border border-border rounded-lg p-4
            overflow-y-auto">
                <div className="space-y-4">
                    {popularFonts.map(font => (
                        <div key={font} className="border-b border-border pb-2">
                            <p className="text-sm text-text-secondary">{font}</p>
                            <p style={{ fontFamily: `'${font}', sans-serif`, fontSize: `${fontSize}px` }}>
                                {text}
                            </p>
                        </div>
                    ))}
                </div>
            </div>
        </div>
    );
};

// ===== FontPairingTool_60.tsx =====

// This feature is an alias for the Typography Lab.
// Re-exporting it here to keep the codebase DRY while satisfying the feature
registry.
export { TypographyLab as FontPairingTool } from './TypographyLab.tsx';
// ===== DevNotesStickyPanel_60.tsx =====
```

```tsx
import React from 'react';

export const DevNotesStickyPanel: React.FC = () => {
    return (
        <div className="p-4 bg-yellow-100 border-l-4 border-yellow-500 text-yellow-700">
            <h4 className="font-bold">Developer Notes</h4>
            <p>This is a placeholder component for developer notes.</p>
        </div>
    );
};


// ===== ApiMockGenerator_48.tsx =====

import React, { useState, useEffect, useCallback } from 'react';
import { generateMockData } from '../../services/aiService.ts';
import { startMockServer, stopMockServer, setMockRoutes, isMockServerRunning }
from '../../services/mocking/mockServer.ts';
import { saveMockCollection, getAllMockCollections } from
'../../services/mocking/db.ts';
import { ServerStackIcon, SparklesIcon, PlusIcon } from '../icons.tsx';
import { LoadingSpinner } from '../shared/index.tsx';

interface MockCollection {
    id: string;
    schemaDescription: string;
    data: any[];
}

interface MockRoute {
    id: number;
    path: string;
    method: 'GET' | 'POST';
    collectionId: string;
}

const exampleSchema = "a user with an id, name, email, and a nested address
object containing a city and country";

export const ApiMockGenerator: React.FC = () => {
    const [schema, setSchema] = useState(exampleSchema);
    const [count, setCount] = useState(5);
    const [collectionName, setCollectionName] = useState('users');
    const [collections, setCollections] = useState<MockCollection[]>([]);
    const [generatedData, setGeneratedData] = useState<any[] | null>(null);
    const [isLoading, setIsLoading] = useState(false);
    const [isServerLoading, setIsServerLoading] = useState(false);
    const [error, setError] = useState('');
    const [isServerRunning, setIsServerRunning] = useState(isMockServerRunning());
    const [routes, setRoutes] = useState<MockRoute[]>([{ id: 1, path: '/api/users',
    method: 'GET', collectionId: 'users' }]);

    useEffect(() => {
        const loadCollections = async () => {
            const storedCollections = await getAllMockCollections();
            setCollections(storedCollections);
        };
        loadCollections();
    }, []);

    const handleGenerate = async () => {
        if (!schema.trim() || !collectionName.trim()) {
```

```
        setError('Schema description and collection name are required.');
        return;
    }
    setIsLoading(true);
    setError('');
    try {
        const data = await generateMockData(schema, count);
        setGeneratedData(data);
        const collectionId = collectionName.toLowerCase().replace(/\s/g, '-');
        await saveMockCollection({ id: collectionId, schemaDescription: schema, data });
        setCollections(await getAllMockCollections());
    } catch (err) {
        setError(err instanceof Error ? err.message : 'Failed to generate data.');
    } finally {
        setIsLoading(false);
    }
};

const handleServerToggle = async () => {
    setIsServerLoading(true);
    if (isServerRunning) {
        await stopMockServer();
        setIsServerRunning(false);
    } else {
        try {
            await startMockServer();
            setIsServerRunning(true);
            updateRoutesOnServer();
        } catch (err) {
            setError(err instanceof Error ? err.message : 'Could not start server.');
        }
    }
    setIsServerLoading(false);
};

const updateRoutesOnServer = useCallback(() => {
    const mockRoutes = routes.map(route => {
        const collection = collections.find(c => c.id === route.collectionId);
        return {
            path: route.path,
            method: route.method,
            response: {
                status: 200,
                body: collection ? collection.data : { message: `No data found for collection
                '${route.collectionId}'.` }
            }
        };
    });
    setMockRoutes(mockRoutes as any);
}, [routes, collections]);


useEffect(() => {
    if (isServerRunning) {
        updateRoutesOnServer();
    }
}, [routes, collections, isServerRunning, updateRoutesOnServer]);

const handleRouteUpdate = (id: number, field: keyof MockRoute, value: string) =>
{
    setRoutes(routes.map(r => r.id === id ? { ...r, [field]: value } : r));
};
```

```
const handleAddRoute = () => {
    const newRoute: MockRoute = {
        id: Date.now(),
        path: '/api/new-route',
        method: 'GET',
        collectionId: collections.length > 0 ? collections[0].id : ''
    };
    setRoutes([...routes, newRoute]);
};

const getServerStatusText = () => {
    if (isServerLoading) return 'Starting...';
    return isServerRunning ? 'Server Running' : 'Server Stopped';
}

return (
    <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
        <header className="mb-6 flex justify-between items-start">
            <div>
                <h1 className="text-3xl font-bold flex items-center"><ServerStackIcon /><span
                className="ml-3">AI API Mock Server</span></h1>
                <p className="text-text-secondary mt-1">Generate and serve mock API data locally
                using a service worker.</p>
            </div>
            <button onClick={handleServerToggle} disabled={isServerLoading} className={`px-4
            py-2 rounded-md font-semibold flex items-center gap-2 ${isServerRunning ? 'bg-
            green-100 text-green-700' : 'bg-gray-100'}`}>
                <span className={`w-3 h-3 rounded-full ${isServerRunning ? 'bg-green-500' : 'bg-
                gray-400'} ${isServerLoading ? 'animate-pulse' : ''}`}></span>
                {getServerStatusText()}
            </button>
        </header>
        <div className="flex-grow grid grid-cols-1 lg:grid-cols-3 gap-6 min-h-0">
            <div className="lg:col-span-1 flex flex-col gap-4 bg-surface p-4 border border-
            border rounded-lg">
                <h3 className="text-lg font-bold">1. Generate Data</h3>
                <div><label className="text-sm">Describe the data schema</label><textarea
                value={schema} onChange={e => setSchema(e.target.value)} className="w-full mt-1
                p-2 bg-background border border-border rounded" rows={4}/></div>
                <div className="flex gap-2">
                    <div className="flex-grow"><label className="text-sm">Collection
                    Name</label><input type="text" value={collectionName} onChange={e =>
                    setCollectionName(e.target.value)} className="w-full mt-1 p-2 bg-background
                    border border-border rounded"/></div>
                    <div><label className="text-sm">Count</label><input type="number" value={count}
                    onChange={e => setCount(Number(e.target.value))} className="w-20 mt-1 p-2 bg-
                    background border border-border rounded"/></div>
                </div>
                <button onClick={handleGenerate} disabled={isLoading} className="btn-primary
                py-2 flex items-center justify-center gap-2">{isLoading ? <LoadingSpinner/> :
                <><SparklesIcon/> Generate & Save</>}</button>
                {error && <p className="text-red-500 text-xs">{error}</p>}
            </div>

            <div className="lg:col-span-2 flex flex-col gap-4 min-h-0">
               <div className="bg-surface p-4 border border-border rounded-lg flex-grow flex
               flex-col min-h-0">
                    <h3 className="text-lg font-bold mb-2">2. View Data & Configure Routes</h3>
                    <div className="flex-grow grid grid-cols-2 gap-4 min-h-0">
                        <div className="overflow-y-auto">
                            <h4 className="font-semibold text-sm mb-1">Saved Collections</h4>
                            {collections.map(c => <div key={c.id} className="text-xs p-2 bg-background
```

```
                                rounded border border-border mb-1">{c.id} ({c.data.length} items)</div>)}
                                <h4 className="font-semibold text-sm mb-1 mt-2">Last Generated Data</h4>
                                <pre className="text-xs p-2 bg-background rounded border border-border
                                whitespace-pre-wrap">{generatedData ? JSON.stringify(generatedData, null, 2) :
                                'No data generated yet.'}</pre>
                            </div>
                            <div className="overflow-y-auto">
                                <h4 className="font-semibold text-sm mb-1">Mock Routes</h4>
                                {routes.map((r) => (
                                <div key={r.id} className="grid grid-cols-3 gap-1 items-center mb-1">
                                    <input type="text" value={r.path} onChange={e => handleRouteUpdate(r.id, 'path',
                                    e.target.value)} className="col-span-2 p-1 text-xs bg-background border rounded"
                                    />
                                    <select value={r.collectionId} onChange={e => handleRouteUpdate(r.id,
                                    'collectionId', e.target.value)} className="p-1 text-xs bg-background border
                                    rounded">
                                        <option value="">Select Collection</option>
                                        {collections.map(c => <option key={c.id} value={c.id}>{c.id}</option>)}
                                    </select>
                                </div>
                                ))}
                                <button onClick={handleAddRoute} className="text-xs mt-2 p-1 bg-gray-100 rounded
                                hover:bg-gray-200"><PlusIcon/></button>
                            </div>
                        </div>
                    </div>
                </div>
            </div>
        );
};

// ===== AiCodeExplainer_58.tsx =====

import React, { useState, useCallback, useEffect, useMemo, useRef } from
'react';
import mermaid from 'mermaid';
import { explainCodeStructured, generateMermaidJs } from
'../../services/index.ts';
import type { StructuredExplanation } from '../../types.ts';
import { useTheme } from '../../hooks/useTheme.ts';
import { CpuChipIcon } from '../icons.tsx';
import { MarkdownRenderer, LoadingSpinner } from '../shared/index.tsx';

const exampleCode = `const bubbleSort = (arr) => {
  for (let i = 0; i < arr.length; i++) {
    for (let j = 0; j < arr.length - i - 1; j++) {
      if (arr[j] > arr[j + 1]) {
        [arr[j], arr[j + 1]] = [arr[j + 1], arr[j]];
      }
    }
  }
  return arr;
};`;

type ExplanationTab = 'summary' | 'lineByLine' | 'complexity' | 'suggestions' |
'flowchart';

const simpleSyntaxHighlight = (code: string) => {
    const escapedCode = code
        .replace(/&/g, '&amp;')
        .replace(/</g, '&lt;')
```

```
        .replace(/>/g, '&gt;');

    return escapedCode
        .replace(/\b(const|let|var|function|return|if|for|=>|import|from|export|default)
        \b/g, '<span class="text-indigo-400 font-semibold">$1</span>')
        .replace(/(\`|'|")(.*?)(\`|'|")/g, '<span class="text-
        emerald-400">$1$2$3</span>')
        .replace(/(\/\/.*)/g, '<span class="text-gray-400 italic">$1</span>')
        .replace(/(\{|\}|\(|\)|\[|\])/g, '<span class="text-gray-400">$1</span>');
};

mermaid.initialize({ startOnLoad: false, securityLevel: 'loose' });

export const AiCodeExplainer: React.FC<{ initialCode?: string }> = ({
initialCode }) => {
    const [code, setCode] = useState<string>(initialCode || exampleCode);
    const [explanation, setExplanation] = useState<StructuredExplanation |
    null>(null);
    const [mermaidCode, setMermaidCode] = useState<string>('');
    const [isLoading, setIsLoading] = useState<boolean>(false);
    const [error, setError] = useState<string>('');
    const [activeTab, setActiveTab] = useState<ExplanationTab>('summary');
    const [themeState] = useTheme();
    const textareaRef = useRef<HTMLTextAreaElement>(null);
    const preRef = useRef<HTMLPreElement>(null);
    const mermaidContainerRef = useRef<HTMLDivElement>(null);

    const handleExplain = useCallback(async (codeToExplain: string) => {
        if (!codeToExplain.trim()) {
            setError('Please enter some code to explain.');
            return;
        }
        setIsLoading(true);
        setError('');
        setExplanation(null);
        setMermaidCode('');
        setActiveTab('summary');
        try {
            const [explanationResult, mermaidResult] = await Promise.all([
                explainCodeStructured(codeToExplain),
                generateMermaidJs(codeToExplain)
            ]);
            setExplanation(explanationResult);
            setMermaidCode(mermaidResult.replace(/```mermaid\n|```/g, ''));

        } catch (err) {
            const errorMessage = err instanceof Error ? err.message : 'An unknown error
            occurred.';
            setError(`Failed to get explanation: ${errorMessage}`);
        } finally {
            setIsLoading(false);
        }
    }, []);

    useEffect(() => {
        if (initialCode) {
            setCode(initialCode);
            handleExplain(initialCode);
        }
    }, [initialCode, handleExplain]);

    useEffect(() => {
```

```
    const renderMermaid = async () => {
         if (activeTab === 'flowchart' && mermaidCode && mermaidContainerRef.current) {
            try {
                mermaid.initialize({ startOnLoad: false, theme: themeState.mode === 'dark' ?
                'dark' : 'neutral', securityLevel: 'loose' });
                mermaidContainerRef.current.innerHTML = ''; // Clear previous
                const { svg } = await mermaid.render(`mermaid-graph-${Date.now()}`,
                mermaidCode);
                mermaidContainerRef.current.innerHTML = svg;
            } catch (e) {
                console.error("Mermaid rendering error:", e);
                mermaidContainerRef.current.innerHTML = `<p class="text-red-500">Error rendering
                flowchart.</p>`;
            }
        }
    }
    renderMermaid();
}, [activeTab, mermaidCode, themeState.mode]);


const handleScroll = () => {
    if (preRef.current && textareaRef.current) {
        preRef.current.scrollTop = textareaRef.current.scrollTop;
        preRef.current.scrollLeft = textareaRef.current.scrollLeft;
    }
};

const highlightedCode = useMemo(() => simpleSyntaxHighlight(code), [code]);

const renderTabContent = () => {
    if (!explanation) return null;
    switch(activeTab) {
        case 'summary':
            return <MarkdownRenderer content={explanation.summary} />;
        case 'lineByLine':
            return (
                <div className="space-y-3">
                    {explanation.lineByLine.map((item, index) => (
                        <div key={index} className="p-3 bg-background rounded-md border border-border">
                            <p className="font-mono text-xs text-primary mb-1">Lines: {item.lines}</p>
                            <p className="text-sm">{item.explanation}</p>
                        </div>
                    ))}
                </div>
            );
        case 'complexity':
            return (
                <div>
                    <p><strong>Time Complexity:</strong> <span className="font-mono text-
                    amber-600">{explanation.complexity.time}</span></p>
                    <p><strong>Space Complexity:</strong> <span className="font-mono text-
                    amber-600">{explanation.complexity.space}</span></p>
                </div>
            );
        case 'suggestions':
            return (
                 <ul className="list-disc list-inside space-y-2">
                    {explanation.suggestions.map((item, index) => <li key={index}>{item}</li>)}
                </ul>
            );
        case 'flowchart':
            return (
```

```
                <div ref={mermaidContainerRef} className="w-full h-full flex items-center
                justify-center">
                    <LoadingSpinner />
                </div>
            );
    }
}

return (
    <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
        <header className="mb-6 flex-shrink-0">
            <h1 className="text-3xl font-bold flex items-center">
                <CpuChipIcon />
                <span className="ml-3">AI Code Explainer</span>
            </h1>
            <p className="text-text-secondary mt-1">Get a detailed, structured analysis of
            any code snippet.</p>
        </header>
        <div className="flex-grow grid grid-cols-1 md:grid-cols-2 gap-6 min-h-0">

            {/* Left Column: Code Input */}
            <div className="flex flex-col min-h-0 md:col-span-1">
                <label htmlFor="code-input" className="text-sm font-medium text-text-secondary
                mb-2">Your Code</label>
                <div className="relative flex-grow bg-surface border border-border rounded-md
                focus-within:ring-2 focus-within:ring-primary overflow-hidden">
                    <textarea
                        ref={textareaRef}
                        id="code-input"
                        value={code}
                        onChange={(e) => setCode(e.target.value)}
                        onScroll={handleScroll}
                        placeholder="Paste your code here..."
                        spellCheck="false"
                        className="absolute inset-0 w-full h-full p-4 bg-transparent resize-none font-
                        mono text-sm text-transparent caret-primary outline-none z-10"
                    />
                    <pre
                        ref={preRef}
                        aria-hidden="true"
                        className="absolute inset-0 w-full h-full p-4 font-mono text-sm text-text-
                        primary pointer-events-none z-0 whitespace-pre-wrap overflow-auto no-scrollbar"
                        dangerouslySetInnerHTML={{ __html: highlightedCode + '\n' }}
                    />
                </div>
                <div className="mt-4 flex-shrink-0">
                    <button
                        onClick={() => handleExplain(code)}
                        disabled={isLoading}
                        className="btn-primary w-full flex items-center justify-center px-6 py-3"
                    >
                        {isLoading ? <LoadingSpinner/> : 'Analyze Code'}
                    </button>
                </div>
            </div>

            {/* Right Column: AI Analysis */}
            <div className="flex flex-col min-h-0 md:col-span-1">
                <label className="text-sm font-medium text-text-secondary mb-2">AI
                Analysis</label>
                <div className="relative flex-grow flex flex-col bg-surface border border-border
                rounded-md overflow-hidden">
```

```
                        <div className="flex-shrink-0 flex border-b border-border">
                           {(['summary', 'lineByLine', 'complexity', 'suggestions', 'flowchart'] as
                           ExplanationTab[]).map(tab => (
                               <button key={tab} onClick={() => setActiveTab(tab)} disabled={!explanation}
                                   className={`px-4 py-2 text-sm font-medium capitalize transition-colors
                                   ${activeTab === tab ? 'bg-background text-primary font-semibold' : 'text-text-
                                   secondary hover:bg-gray-100 dark:hover:bg-slate-700 disabled:text-gray-400
                                   dark:disabled:text-slate-500'}`}>
                                      {tab.replace(/([A-Z])/g, ' $1')}
                               </button>
                           ))}
                        </div>
                        <div className="p-4 flex-grow overflow-y-auto">
                            {isLoading && <div className="flex items-center justify-center
                            h-full"><LoadingSpinner /></div>}
                            {error && <p className="text-red-500">{error}</p>}
                            {explanation && !isLoading && renderTabContent()}
                            {!isLoading && !explanation && !error && <div className="text-text-secondary
                            h-full flex items-center justify-center">The analysis will appear here.</div>}
                        </div>
                    </div>
                </div>
            </div>
        </div>
    );
};

// ===== FeatureForge_4.tsx =====

import React, { useState, useEffect, useCallback } from 'react';
import { generateAppFeatureComponent } from '../../services/aiService.ts';
import { getAllCustomFeatures, saveCustomFeature, deleteCustomFeature } from
'../../services/dbService.ts';
import type { CustomFeature } from '../../types.ts';
import { CpuChipIcon, PlusIcon, TrashIcon } from '../icons.tsx';
import { LoadingSpinner } from '../shared/index.tsx';
import { useNotification } from '../../contexts/NotificationContext.tsx';
import { ALL_FEATURES } from './index.ts';
import { CustomFeatureRunner } from '../shared/CustomFeatureRunner.tsx';

const ICON_MAP: Record<string, React.FC> = ALL_FEATURES.reduce((acc, feature) =>
{
    const iconType = (feature.icon as React.ReactElement)?.type;
    if (typeof iconType === 'function' && iconType.name) {
      const iconName = iconType.name;
      acc[iconName] = iconType as React.FC;
    }
    return acc;
  }, {} as Record<string, React.FC>);


export const FeatureForge: React.FC = () => {
    const [customFeatures, setCustomFeatures] = useState<CustomFeature[]>([]);
    const [isLoading, setIsLoading] = useState<'list' | 'generate' | false>(false);
    const [isGenerating, setIsGenerating] = useState(false);
    const [prompt, setPrompt] = useState('A tool to convert JSON to YAML');
    const [generatedFeature, setGeneratedFeature] = useState<Omit<CustomFeature,
    'id'> | null>(null);
    const { addNotification } = useNotification();

    const fetchFeatures = useCallback(async () => {
        setIsLoading('list');
```

```
    const features = await getAllCustomFeatures();
    setCustomFeatures(features);
    setIsLoading(false);
}, []);

useEffect(() => {
    fetchFeatures();
}, [fetchFeatures]);

const handleGenerate = async () => {
    if (!prompt.trim()) return;
    setIsGenerating(true);
    setGeneratedFeature(null);
    try {
        const result = await generateAppFeatureComponent(prompt);
        setGeneratedFeature(result);
        addNotification('Feature code generated! Review and save.', 'info');
    } catch (err) {
        addNotification(err instanceof Error ? err.message : 'Failed to generate
        feature', 'error');
    } finally {
        setIsGenerating(false);
    }
};

const handleSave = async () => {
    if (!generatedFeature) return;
    const newFeature: CustomFeature = {
        ...generatedFeature,
        id: `custom-${Date.now()}`
    };
    await saveCustomFeature(newFeature);
    // Dispatch event to notify other parts of the app (like the desktop view) to
    reload features
    window.dispatchEvent(new CustomEvent('custom-feature-update'));

    setGeneratedFeature(null);
    setPrompt('');
    fetchFeatures();
    addNotification(`Feature "${newFeature.name}" saved! It's now available on your
    desktop.`, 'success');
};

const handleDelete = async (id: string) => {
    if (window.confirm("Are you sure you want to delete this feature?")) {
        await deleteCustomFeature(id);
        // Dispatch event to notify other parts of the app (like the desktop view) to
        reload features
        window.dispatchEvent(new CustomEvent('custom-feature-update'));
        fetchFeatures();
        addNotification('Feature deleted.', 'info');
    }
};

const IconComponent = ({ name }: { name: string }) => {
    const Comp = ICON_MAP[name];
    return Comp ? <Comp /> : <CpuChipIcon />;
};

return (
    <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
        <header className="mb-6">
```

```
    <h1 className="text-3xl font-bold flex items-center"><CpuChipIcon /><span
    className="ml-3">Feature Forge</span></h1>
    <p className="text-text-secondary mt-1">Use AI to create new tools and add them
    to your desktop.</p>
</header>
<div className="flex-grow grid grid-cols-1 lg:grid-cols-2 gap-6 min-h-0">
    {/* Left: Generator & Preview */}
    <div className="lg:col-span-1 flex flex-col gap-4">
        <div className="bg-surface p-4 border border-border rounded-lg">
            <h3 className="text-lg font-bold">1. Create a New Feature</h3>
            <div className="flex flex-col mt-2">
                <label className="text-sm">Describe the tool you want to build</label>
                <textarea value={prompt} onChange={e => setPrompt(e.target.value)}
                className="w-full mt-1 p-2 bg-background border border-border rounded"
                rows={3}/>
            </div>
            <button onClick={handleGenerate} disabled={isGenerating} className="btn-primary
            w-full mt-2 py-2 flex items-center justify-center gap-2">{isGenerating ?
            <LoadingSpinner/> : 'Generate Feature'}</button>
        </div>
        {generatedFeature && (
            <div className="flex-grow flex flex-col bg-surface p-4 border border-dashed
            rounded-lg space-y-2 animate-pop-in min-h-0">
                <div className="flex justify-between items-center">
                    <h4 className="font-bold">2. Review & Save</h4>
                    <button onClick={handleSave} className="px-4 py-1 bg-green-600 text-white font-
                    bold rounded-md text-sm">Save Feature</button>
                </div>
                <p className="text-sm"><strong>Name:</strong> {generatedFeature.name}</p>
                <div className="flex-grow border rounded-md overflow-hidden min-h-[200px]">
                     <CustomFeatureRunner feature={{ ...generatedFeature, id: 'preview' }} />
                </div>
            </div>
        )}
    </div>

    {/* Right: Existing Custom Features */}
    <div className="lg:col-span-1 flex flex-col gap-4 min-h-0">
       <div className="bg-surface p-4 border border-border rounded-lg flex-grow flex
       flex-col min-h-0">
            <h3 className="text-lg font-bold mb-2">3. Your Custom Features</h3>
            <div className="flex-grow overflow-y-auto pr-2">
                {isLoading === 'list' && <LoadingSpinner />}
                {customFeatures.length === 0 && !isLoading && <p className="text-text-secondary
                text-center py-8">You haven't created any features yet.</p>}
                <div className="space-y-3">
                    {customFeatures.map(feature => (
                        <div key={feature.id} className="group bg-background p-3 rounded-lg border
                        border-border flex items-center justify-between">
                            <div className="flex items-center gap-3">
                                <div className="text-primary"><IconComponent name={feature.icon} /></div>
                                <div>
                                    <h4 className="font-semibold">{feature.name}</h4>
                                    <p className="text-xs text-text-secondary">{feature.description}</p>
                                </div>
                            </div>
                            <button onClick={() => handleDelete(feature.id)} className="opacity-0 group-
                            hover:opacity-100 text-red-500 hover:text-red-700 p-1"><TrashIcon /></button>
                        </div>
                    ))}
                </div>
            </div>
```

```
                </div>
            </div>
        </div>
    </div>
  );
};

// ===== WorkspaceConnectorHub_29.tsx =====

import React, { useState, useEffect, useMemo, useCallback } from 'react';
import { useGlobalState } from '../../contexts/GlobalStateContext.tsx';
import * as vaultService from '../../services/vaultService.ts';
import { useNotification } from '../../contexts/NotificationContext.tsx';
import { validateToken } from '../../services/authService.ts';
import { ACTION_REGISTRY, executeWorkspaceAction } from
'../../services/workspaceConnectorService.ts';
import { RectangleGroupIcon, GithubIcon, SparklesIcon } from '../icons.tsx';
import { LoadingSpinner } from '../shared/index.tsx';
import { signInWithGoogle } from '../../services/googleAuthService.ts';
import { useVaultModal } from '../../contexts/VaultModalContext.tsx';

const ServiceConnectionCard: React.FC<{
    serviceName: string;
    icon: React.ReactNode;
    fields: { id: string; label: string; placeholder: string }[];
    onConnect: (credentials: Record<string, string>) => Promise<void>;
    onDisconnect: () => Promise<void>;
    status: string;
    isLoading: boolean;
}> = ({ serviceName, icon, fields, onConnect, onDisconnect, status, isLoading })
=> {
    const [creds, setCreds] = useState<Record<string, string>>({});

    const handleConnect = () => {
        onConnect(creds);
    };

    const isConnected = status.startsWith('Connected');

    return (
        <div className="bg-surface border border-border rounded-lg p-6">
            <div className="flex items-center justify-between">
                <div className="flex items-center gap-4">
                    <div className="w-10 h-10">{icon}</div>
                    <div>
                        <h3 className="text-lg font-bold text-text-primary">{serviceName}</h3>
                        <p className={`text-sm ${isConnected ? 'text-green-600' : 'text-text-
                        secondary'}`}>{status}</p>
                    </div>
                </div>
                {isConnected && (
                    <button onClick={onDisconnect} className="px-4 py-2 bg-red-500/10 text-red-600
                    font-semibold rounded-lg hover:bg-red-500/20">
                        Disconnect
                    </button>
                )}
            </div>
            {!isConnected && (
                <div className="mt-4 pt-4 border-t border-border space-y-2">
                    {fields.map(field => (
                        <div key={field.id}>
                            <label className="text-xs text-text-secondary">{field.label}</label>
```

```
                            <input
                                type={field.id.includes('token') || field.id.includes('pat') ||
                                field.id.includes('key') ? 'password' : 'text'}
                                value={creds[field.id] || ''}
                                onChange={e => setCreds(prev => ({ ...prev, [field.id]: e.target.value }))}
                                placeholder={field.placeholder}
                                className="w-full mt-1 p-2 bg-background border border-border rounded-md text-
                                sm"
                            />
                        </div>
                    ))}
                    <button onClick={handleConnect} disabled={isLoading} className="btn-primary
                    w-full mt-2 py-2 flex items-center justify-center">
                        {isLoading ? <LoadingSpinner /> : 'Connect'}
                    </button>
                </div>
            )}
        </div>
    );
};


export const WorkspaceConnectorHub: React.FC = () => {
    const { state, dispatch } = useGlobalState();
    const { user, githubUser, vaultState } = state;
    const { addNotification } = useNotification();
    const { requestUnlock, requestCreation } = useVaultModal();
    const [loadingStates, setLoadingStates] = useState<Record<string, boolean>>({});
    const [connectionStatuses, setConnectionStatuses] = useState<Record<string,
    string>>({});

    // Manual action state
    const [selectedActionId, setSelectedActionId] =
    useState<string>([...ACTION_REGISTRY.keys()][0]);
    const [actionParams, setActionParams] = useState<Record<string, any>>({});
    const [isExecuting, setIsExecuting] = useState(false);
    const [actionResult, setActionResult] = useState<string>('');

    const services = useMemo(() => {
        const serviceMap = new Map();
        ACTION_REGISTRY.forEach(action => {
            if (!serviceMap.has(action.service)) {
                serviceMap.set(action.service, {
                    name: action.service,
                    actions: [],
                });
            }
            serviceMap.get(action.service).actions.push(action);
        });
        return Array.from(serviceMap.values());
    }, []);

    const checkConnections = useCallback(async () => {
        if (!user || !vaultState.isUnlocked) return;

        const checkCred = async (credId: string, serviceName: string, successMessage:
        string) => {
             const token = await vaultService.getDecryptedCredential(credId);
             setConnectionStatuses(s => ({ ...s, [serviceName]: token ? successMessage : 'Not
             Connected' }));
        };
        await checkCred('gemini_api_key', 'Google Gemini', 'Connected');
```

```
    await checkCred('github_pat', 'GitHub', githubUser ? `Connected as
    ${githubUser.login}`: 'Connected');
    await checkCred('jira_pat', 'Jira', 'Connected');
    await checkCred('slack_bot_token', 'Slack', 'Connected');

}, [user, vaultState.isUnlocked, githubUser]);

useEffect(() => {
    checkConnections();
}, [checkConnections]);

const withVault = useCallback(async (callback: () => Promise<void>) => {
    if (!vaultState.isInitialized) {
        const created = await requestCreation();
        if (!created) { addNotification('Vault setup is required.', 'error'); return; }
    }
    if (!vaultState.isUnlocked) {
        const unlocked = await requestUnlock();
        if (!unlocked) { addNotification('Vault must be unlocked to manage
        connections.', 'error'); return; }
    }
    await callback();
}, [vaultState, requestCreation, requestUnlock, addNotification]);


const handleConnect = async (serviceName: string, credentials: Record<string,
string>) => {
    await withVault(async () => {
        setLoadingStates(s => ({ ...s, [serviceName]: true }));
        try {
            for (const [key, value] of Object.entries(credentials)) {
                if (value) await vaultService.saveCredential(key, value);
            }
            if (serviceName === 'GitHub' && credentials.github_pat) {
                 const githubProfile = await validateToken(credentials.github_pat);
                 dispatch({ type: 'SET_GITHUB_USER', payload: githubProfile });
                 await vaultService.saveCredential('github_user', JSON.stringify(githubProfile));
            }
            addNotification(`${serviceName} connected successfully!`, 'success');
            checkConnections();
        } catch (e) {
            addNotification(`Failed to connect ${serviceName}: ${e instanceof Error ?
            e.message : 'Unknown error'}`, 'error');
        } finally {
            setLoadingStates(s => ({ ...s, [serviceName]: false }));
        }
    });
};

const handleDisconnect = async (serviceName: string, credIds: string[]) => {
   await withVault(async () => {
        setLoadingStates(s => ({ ...s, [serviceName]: true }));
        try {
            for (const id of credIds) {
                 await vaultService.saveCredential(id, ''); // Overwrite with empty string
            }
             if (serviceName === 'GitHub') {
                 dispatch({ type: 'SET_GITHUB_USER', payload: null });
                 await vaultService.saveCredential('github_user', '');
            }
            addNotification(`${serviceName} disconnected.`, 'info');
            checkConnections();
```

```
        } catch(e) {
            addNotification(`Failed to disconnect ${serviceName}.`, 'error');
        } finally {
            setLoadingStates(s => ({ ...s, [serviceName]: false }));
        }
   });
};

const handleExecuteAction = async () => {
    await withVault(async () => {
        setIsExecuting(true);
        setActionResult('');
        try {
            const result = await executeWorkspaceAction(selectedActionId, actionParams);
            setActionResult(JSON.stringify(result, null, 2));
            addNotification('Action executed successfully!', 'success');
        } catch(e) {
            setActionResult(`Error: ${e instanceof Error ? e.message : 'Unknown Error'}`);
            addNotification('Action failed.', 'error');
        } finally {
            setIsExecuting(false);
        }
    });
};

const handleSignIn = () => {
    signInWithGoogle();
    // The result is handled by the global callback set in App.tsx
};

const selectedAction = ACTION_REGISTRY.get(selectedActionId);
const actionParameters = selectedAction ? selectedAction.getParameters() : {};

if (!user) {
    return (
        <div className="h-full flex items-center justify-center">
            <div className="text-center bg-surface p-8 rounded-lg border border-border max-
            w-md">
                <h2 className="text-xl font-bold">Sign In Required</h2>
                <p className="text-text-secondary my-4">Please sign in with your Google account
                to manage workspace connections.</p>
                <button onClick={handleSignIn} disabled={loadingStates.google} className="btn-
                primary px-6 py-3 flex items-center justify-center gap-2 mx-auto">
                    {loadingStates.google ? <LoadingSpinner/> : 'Sign in with Google'}
                </button>
            </div>
        </div>
    );
}

return (
    <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
         <header className="mb-8">
            <h1 className="text-4xl font-extrabold tracking-tight flex items-
            center"><RectangleGroupIcon /><span className="ml-3">Workspace Connector
            Hub</span></h1>
            <p className="mt-2 text-lg text-text-secondary">Connect to your development
            services to unlock cross-platform AI actions.</p>
        </header>
        <div className="flex-grow grid grid-cols-1 lg:grid-cols-2 gap-8 min-h-0">
            <div className="flex flex-col gap-6 overflow-y-auto pr-4">
                <h2 className="text-2xl font-bold">Service Connections</h2>
```

```
    <ServiceConnectionCard
        serviceName="Google Gemini"
        icon={<SparklesIcon />}
        fields={[{ id: 'gemini_api_key', label: 'API Key', placeholder: 'Your Gemini API
        Key' }]}
        onConnect={(creds) => handleConnect('Google Gemini', creds)}
        onDisconnect={() => handleDisconnect('Google Gemini', ['gemini_api_key'])}
        status={connectionStatuses['Google Gemini'] || 'Checking...'}
        isLoading={loadingStates['Google Gemini']}
    />
    <ServiceConnectionCard
        serviceName="GitHub"
        icon={<GithubIcon />}
        fields={[{ id: 'github_pat', label: 'Personal Access Token', placeholder:
        'ghp_...' }]}
        onConnect={(creds) => handleConnect('GitHub', creds)}
        onDisconnect={() => handleDisconnect('GitHub', ['github_pat'])}
        status={connectionStatuses.GitHub || 'Checking...'}
        isLoading={loadingStates.GitHub}
    />
     {/* Placeholder cards for Jira and Slack */}
    <ServiceConnectionCard
        serviceName="Jira"
        icon={<div className="w-10 h-10 bg-[#0052CC] rounded flex items-center justify-
        center text-white font-bold text-xl">J</div>}
        fields={[
            { id: 'jira_domain', label: 'Jira Domain', placeholder: 'your-
            company.atlassian.net' },
            { id: 'jira_email', label: 'Your Jira Email', placeholder: 'you@example.com' },
            { id: 'jira_pat', label: 'API Token', placeholder: 'Your API Token' },
        ]}
        onConnect={(creds) => handleConnect('Jira', creds)}
        onDisconnect={() => handleDisconnect('Jira', ['jira_domain', 'jira_email',
        'jira_pat'])}
        status={connectionStatuses.Jira || 'Checking...'}
        isLoading={loadingStates.Jira}
    />
    <ServiceConnectionCard
        serviceName="Slack"
        icon={<div className="w-10 h-10 bg-[#4A154B] rounded flex items-center justify-
        center text-white font-bold text-2xl">#</div>}
        fields={[{ id: 'slack_bot_token', label: 'Bot User OAuth Token', placeholder:
        'xoxb-...' }]}
        onConnect={(creds) => handleConnect('Slack', creds)}
        onDisconnect={() => handleDisconnect('Slack', ['slack_bot_token'])}
        status={connectionStatuses.Slack || 'Checking...'}
        isLoading={loadingStates.Slack}
    />
</div>
<div className="flex flex-col gap-6 bg-surface p-6 border border-border rounded-
lg">
    <h2 className="text-2xl font-bold">Manual Action Runner</h2>
    <div className="space-y-4">
         <div>
            <label className="text-sm font-medium">Action</label>
            <select value={selectedActionId} onChange={e =>
            setSelectedActionId(e.target.value)} className="w-full mt-1 p-2 bg-background
            border rounded">
                {services.map(service => (
                    <optgroup label={service.name} key={service.name}>
                        {service.actions.map((action: any) => (
                            <option key={action.id} value={action.id}>{action.description}</option>
```

```
                                    ))}
                                </optgroup>
                            ))}
                        </select>
                    </div>
                    {Object.entries(actionParameters).map(([key, param]: [string, any]) => (
                        <div key={key}>
                            <label className="text-sm font-medium">{key} {param.required && '*'}</label>
                            <input
                                type={param.type}
                                value={actionParams[key] || ''}
                                onChange={e => setActionParams(p => ({...p, [key]: e.target.value}))}
                                placeholder={param.default || ''}
                                className="w-full mt-1 p-2 bg-background border rounded"
                            />
                        </div>
                    ))}
                    <button onClick={handleExecuteAction} disabled={isExecuting} className="btn-
                    primary w-full py-2 flex items-center justify-center gap-2">
                       {isExecuting ? <LoadingSpinner/> : <><SparklesIcon /> Execute Action</>}
                    </button>
                </div>
                <div>
                    <label className="text-sm font-medium">Result</label>
                    <pre className="w-full h-48 mt-1 p-2 bg-background border rounded overflow-auto
                    text-xs">{actionResult || 'Action results will appear here.'}</pre>
                </div>
            </div>
        </div>
    </div>
  );
};

// ===== AiImageGenerator_61.tsx =====

import React, { useState, useCallback, useRef } from 'react';
import { generateImage, generateImageFromImageAndText, fileToBase64,
blobToDataURL } from '../../services/index.ts';
import { ImageGeneratorIcon, SparklesIcon, ArrowDownTrayIcon, XMarkIcon } from
'../icons.tsx';
import { LoadingSpinner } from '../shared/index.tsx';

const surprisePrompts = [
    'A majestic lion wearing a crown, painted in the style of Van Gogh.',
    'A futuristic cityscape on another planet with two moons in the sky.',
    'A cozy, magical library inside a giant tree.',
    'A surreal image of a ship sailing on a sea of clouds.',
    'An astronaut riding a space-themed bicycle on the moon.',
];

interface UploadedImage {
    base64: string;
    dataUrl: string;
    mimeType: string;
}

export const AiImageGenerator: React.FC = () => {
    const [prompt, setPrompt] = useState<string>('A photorealistic image of a
    futuristic city at sunset, with flying cars.');
    const [uploadedImage, setUploadedImage] = useState<UploadedImage | null>(null);
    const [generatedImageUrl, setGeneratedImageUrl] = useState<string | null>(null);
    const [isLoading, setIsLoading] = useState<boolean>(false);
```

```
const [error, setError] = useState<string>('');
const fileInputRef = useRef<HTMLInputElement>(null);

const handleGenerate = useCallback(async () => {
    if (!prompt.trim()) {
        setError('Please enter a prompt to generate an image.');
        return;
    }
    setIsLoading(true);
    setError('');
    setGeneratedImageUrl(null);
    try {
        let resultUrl: string;
        if (uploadedImage) {
            resultUrl = await generateImageFromImageAndText(prompt, uploadedImage.base64,
            uploadedImage.mimeType);
        } else {
            resultUrl = await generateImage(prompt);
        }
        setGeneratedImageUrl(resultUrl);
    } catch (err) {
        const errorMessage = err instanceof Error ? err.message : 'An unknown error
        occurred.';
        setError(`Failed to generate image: ${errorMessage}`);
    } finally {
        setIsLoading(false);
    }
}, [prompt, uploadedImage]);

const handleSurpriseMe = () => {
    const randomPrompt = surprisePrompts[Math.floor(Math.random() *
    surprisePrompts.length)];
    setPrompt(randomPrompt);
};

const processImageBlob = async (blob: Blob) => {
    try {
        const [dataUrl, base64] = await Promise.all([
            blobToDataURL(blob),
            fileToBase64(blob as File)
        ]);
        setUploadedImage({ dataUrl, base64, mimeType: blob.type });
    } catch (e) {
        setError('Could not process the image.');
    }
};

const handlePaste = useCallback(async (event: React.ClipboardEvent) => {
    const items = event.clipboardData.items;
    for (const item of items) {
        if (item.type.indexOf('image') !== -1) {
            const blob = item.getAsFile();
            if (blob) {
                await processImageBlob(blob);
                return;
            }
        }
    }
}, []);

const handleFileChange = async (event: React.ChangeEvent<HTMLInputElement>) => {
    const file = event.target.files?.[0];
```

```
    if (file) {
        await processImageBlob(file);
    }
};

const handleDownload = () => {
    if (!generatedImageUrl) return;
    const link = document.createElement('a');
    link.href = generatedImageUrl;
    link.download = `${prompt.slice(0, 30).replace(/\s/g, '_')}.png`;
    document.body.appendChild(link);
    link.click();
    document.body.removeChild(link);
}

return (
    <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
        <header className="mb-6">
            <h1 className="text-3xl font-bold flex items-center">
                <ImageGeneratorIcon />
                <span className="ml-3">AI Image Generator</span>
            </h1>
            <p className="text-text-secondary mt-1">Generate images from text, or provide an
            image for inspiration.</p>
        </header>

        <div className="flex-grow grid grid-cols-1 lg:grid-cols-2 gap-6 min-h-0">
            {/* Left Column: Inputs */}
            <div className="flex flex-col gap-4">
                <div>
                    <label htmlFor="prompt-input" className="text-sm font-medium text-text-
                    secondary">Your Prompt</label>
                    <textarea
                        id="prompt-input"
                        value={prompt}
                        onChange={(e) => setPrompt(e.target.value)}
                        placeholder="e.g., A cute cat wearing a wizard hat"
                        className="w-full p-3 mt-1 rounded-md bg-surface border border-border
                        focus:ring-2 focus:ring-primary focus:outline-none resize-y"
                        rows={3}
                    />
                </div>

                <div className="flex flex-col flex-grow min-h-[200px]">
                     <label className="text-sm font-medium text-text-secondary mb-1">Inspiration
                     Image (Optional)</label>
                     <div onPaste={handlePaste} className="relative flex-grow flex flex-col items-
                     center justify-center bg-surface p-4 rounded-lg border-2 border-dashed border-
                     border focus:outline-none focus:border-primary" tabIndex={0}>
                        {uploadedImage ? (
                            <>
                                <img src={uploadedImage.dataUrl} alt="Uploaded content" className="max-w-full
                                max-h-full object-contain rounded-md shadow-lg" />
                                <button onClick={() => setUploadedImage(null)} className="absolute top-2 right-2
                                p-1 bg-black/30 text-white rounded-full hover:bg-black/50"><XMarkIcon
                                /></button>
                            </>
                        ) : (
                            <div className="text-center text-text-secondary">
                                <h2 className="text-lg font-bold text-text-primary">Paste an image here</h2>
                                <p className="text-sm">(Cmd/Ctrl + V)</p>
                                <p className="text-xs my-1">or</p>
```

```
                                <button onClick={() => fileInputRef.current?.click()} className="text-sm font-
                                semibold text-primary hover:underline">Upload File</button>
                                <input type="file" ref={fileInputRef} onChange={handleFileChange}
                                accept="image/*" className="hidden"/>
                            </div>
                        )}
                     </div>
                </div>

                <div className="flex gap-2">
                    <button
                        onClick={handleGenerate}
                        disabled={isLoading}
                        className="btn-primary w-full flex items-center justify-center px-6 py-3"
                    >
                        {isLoading ? <LoadingSpinner /> : 'Generate Image'}
                    </button>
                    <button
                        onClick={handleSurpriseMe}
                        disabled={isLoading}
                        className="px-4 py-3 bg-surface border border-border rounded-md hover:bg-
                        gray-100 transition-colors"
                        title="Surprise Me!"
                    >
                        <SparklesIcon />
                    </button>
                </div>
            </div>

            {/* Right Column: Output */}
            <div className="flex flex-col h-full">
                <label className="text-sm font-medium text-text-secondary mb-2">Generated
                Image</label>
                <div className="flex-grow flex items-center justify-center bg-background
                border-2 border-dashed border-border rounded-lg p-4 relative overflow-auto">
                    {isLoading && <LoadingSpinner />}
                    {error && <p className="text-red-500 text-center">{error}</p>}
                    {generatedImageUrl && !isLoading && (
                        <>
                            <img src={generatedImageUrl} alt={prompt || "Generated by AI"} className="max-w-
                            full max-h-full object-contain rounded-md shadow-lg" />
                            <button
                              onClick={handleDownload}
                              className="absolute top-4 right-4 p-2 bg-black/30 text-white rounded-full
                              hover:bg-black/50 backdrop-blur-sm"
                              title="Download Image"
                            >
                                <ArrowDownTrayIcon />
                            </button>
                        </>
                    )}
                    {!isLoading && !generatedImageUrl && !error && (
                        <div className="text-center text-text-secondary">
                            <p>Your generated image will appear here.</p>
                        </div>
                    )}
                </div>
            </div>
        </div>
    </div>
  );
};
```

```tsx
// ===== AsyncCallTreeViewer_63.tsx =====



import React, { useState, useMemo } from 'react';
import { ChartBarIcon } from '../icons.tsx';

type CallNode = {
    name: string;
    duration: number;
    children?: CallNode[];
}

const exampleJson = `{
    "name": "startApp",
    "duration": 500,
    "children": [
        {
            "name": "fetchUserData",
            "duration": 300,
            "children": [
                { "name": "authenticate", "duration": 100 },
                { "name": "fetchProfile", "duration": 150 }
            ]
        },
        {
            "name": "loadInitialAssets",
            "duration": 450,
            "children": [
                { "name": "loadImage.png", "duration": 200 },
                { "name": "loadScript.js", "duration": 250 }
            ]
        }
    ]
}`;


const TreeNode: React.FC<{ node: CallNode, level: number, maxDuration: number }>
= ({ node, level, maxDuration }) => {
    const [isOpen, setIsOpen] = React.useState(true);
    const hasChildren = node.children && node.children.length > 0;

    return (
        <div className="my-1">
            <div
                className="flex items-center p-2 rounded-md hover:bg-gray-100"
                style={{ paddingLeft: `${level * 20 + 8}px` }}
            >
                {hasChildren && (
                    <button onClick={() => setIsOpen(!isOpen)} className={`mr-2 text-text-secondary
                    w-4 h-4 flex-shrink-0 transform transition-transform ${isOpen ? 'rotate-90' :
                    ''}`}>
                        ■
                    </button>
                )}
                 {!hasChildren && <div className="w-6 mr-2 flex-shrink-0" />}
                 <div className="flex-grow flex items-center justify-between gap-4">
                    <span className="truncate">{node.name}</span>
                    <div className="flex items-center gap-2 flex-shrink-0">
                         <div className="w-24 h-4 bg-gray-200 rounded-full overflow-hidden">
                            <div className="h-4 bg-primary" style={{ width: `${(node.duration / maxDuration)
```

```
                            * 100}%` }}/>
                        </div>
                        <span className="text-primary w-16 text-
                        right">{node.duration.toFixed(0)}ms</span>
                    </div>
                </div>
            </div>
            {isOpen && hasChildren && (
                <div>
                    {node.children!.map((child, index) => (
                        <TreeNode key={index} node={child} level={level + 1} maxDuration={maxDuration}
                        />
                    ))}
                </div>
            )}
        </div>
    );
};


export const AsyncCallTreeViewer: React.FC = () => {
    const [jsonInput, setJsonInput] = useState(exampleJson);
    const [error, setError] = useState('');

    const { treeData, maxDuration } = useMemo(() => {
        try {
            const data: CallNode = JSON.parse(jsonInput);
             let max = 0;
            const findMax = (node: CallNode) => {
                if (node.duration > max) max = node.duration;
                if (node.children) node.children.forEach(findMax);
            };
            findMax(data);
            setError('');
            return { treeData: data, maxDuration: max };
        } catch (e) {
            setError('Invalid JSON format.');
            return { treeData: null, maxDuration: 0 };
        }
    }, [jsonInput]);

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
            <header className="mb-6">
                <h1 className="text-3xl flex items-center">
                    <ChartBarIcon />
                    <span className="ml-3">Async Call Tree Viewer</span>
                </h1>
                <p className="text-text-secondary mt-1">Paste a JSON structure to visualize an
                asynchronous function call tree.</p>
            </header>
            <div className="flex-grow flex flex-col gap-4 min-h-0">
                <div className="flex flex-col h-2/5 min-h-[200px]">
                    <label htmlFor="json-input" className="text-sm font-medium text-text-secondary
                    mb-2">JSON Input</label>
                    <textarea
                        id="json-input"
                        value={jsonInput}
                        onChange={e => setJsonInput(e.target.value)}
                        className={`flex-grow p-4 bg-surface border ${error ? 'border-red-500' :
                        'border-border'} rounded-md resize-y font-mono text-sm`}
                        spellCheck="false"
```

```
                    />
                    {error && <p className="text-red-500 text-xs mt-1">{error}</p>}
                </div>
                <div className="flex flex-col flex-grow min-h-0">
                    <label className="text-sm font-medium text-text-secondary mb-2">Visual
                    Tree</label>
                    <div className="flex-grow bg-surface p-4 rounded-lg text-sm overflow-y-auto
                    border border-border">
                        {treeData ? <TreeNode node={treeData} level={0} maxDuration={maxDuration} /> :
                        <div className="text-text-secondary">{error || 'Enter valid JSON to see the
                        tree.'}</div>}
                    </div>
                </div>
            </div>
        </div>
    );
};

// ===== ColorPaletteGenerator_63.tsx =====


import React, { useState, useCallback } from 'react';
import { HexColorPicker } from 'react-colorful';
import { generateColorPalette, downloadFile } from '../../services/index.ts';
import { SparklesIcon, ArrowDownTrayIcon } from '../icons.tsx';
import { LoadingSpinner } from '../shared/index.tsx';

type PreviewColors = {
    cardBg: string;
    pillBg: string;
    pillText: string;
    buttonBg: string;
}

const PreviewCard: React.FC<{ palette: string[], colors: PreviewColors,
setColors: React.Dispatch<React.SetStateAction<PreviewColors>> }> = ({ palette,
colors, setColors }) => {

    const ColorSelector: React.FC<{ label: string, value: string, onChange: (val:
    string) => void }> = ({ label, value, onChange }) => (
        <div className="flex items-center justify-between text-sm">
            <label className="text-text-primary">{label}</label>
            <div className="flex items-center gap-2">
                {palette.map(color => (
                     <button
                        key={color}
                        onClick={() => onChange(color)}
                        className={`w-5 h-5 rounded-full border border-gray-300 ${value === color ?
                        'ring-2 ring-primary ring-offset-1' : ''}`}
                        style={{ backgroundColor: color }}
                        title={color}
                     />
                ))}
            </div>
        </div>
    );

    return (
        <div className="bg-surface p-4 rounded-lg border border-border w-full max-w-sm">
            <h3 className="text-lg font-bold mb-4 text-text-primary">Live Preview</h3>
            <div className="p-8 rounded-xl mb-4" style={{ backgroundColor: colors.cardBg }}>
                <div className="px-4 py-1 rounded-full text-center text-sm inline-block"
```

```
                style={{ backgroundColor: colors.pillBg, color: colors.pillText }}>
                    New Feature
                </div>
                <div className="mt-8 text-center">
                     <button className="px-6 py-2 rounded-lg font-bold" style={{ backgroundColor:
                     colors.buttonBg, color: colors.cardBg }}>
                        Get Started
                    </button>
                </div>
            </div>
            <div className="space-y-3">
                <ColorSelector label="Card Background" value={colors.cardBg} onChange={val =>
                setColors(c => ({...c, cardBg: val}))} />
                <ColorSelector label="Pill Background" value={colors.pillBg} onChange={val =>
                setColors(c => ({...c, pillBg: val}))} />
                <ColorSelector label="Pill Text" value={colors.pillText} onChange={val =>
                setColors(c => ({...c, pillText: val}))} />
                <ColorSelector label="Button Background" value={colors.buttonBg} onChange={val
                => setColors(c => ({...c, buttonBg: val}))} />
            </div>
        </div>
    );
};

export const ColorPaletteGenerator: React.FC = () => {
    const [baseColor, setBaseColor] = useState("#0047AB");
    const [palette, setPalette] = useState<string[]>(['#F0F2F5', '#CCD3E8',
    '#99AADD', '#6688D1', '#3366CC', '#0047AB']);
    const [isLoading, setIsLoading] = useState<boolean>(false);
    const [error, setError] = useState<string>('');
    const [previewColors, setPreviewColors] = useState<PreviewColors>({
        cardBg: '#F0F2F5', pillBg: '#CCD3E8', pillText: '#0047AB', buttonBg: '#0047AB'
    });

    const handleGenerate = useCallback(async () => {
        setIsLoading(true);
        setError('');
        try {
            const result = await generateColorPalette(baseColor);
            setPalette(result.colors);
            setPreviewColors({
                cardBg: result.colors[0],
                pillBg: result.colors[2],
                pillText: result.colors[5],
                buttonBg: result.colors[5],
            })
        } catch (err) {
            const errorMessage = err instanceof Error ? err.message : 'An unknown error
            occurred.';
            setError(`Failed to generate palette: ${errorMessage}`);
        } finally {
            setIsLoading(false);
        }
    }, [baseColor]);

    const downloadColors = () => {
        const cssContent = `:root {\n${palette.map((c, i) => `  --color-palette-${i+1}:
        ${c};`).join('\n')}\n}`;
        downloadFile(cssContent, 'palette.css', 'text/css');
    };

    const downloadCard = () => {
```

```
        const htmlContent = `
<div class="card">
  <div class="pill">New Feature</div>
  <button class="button">Get Started</button>
</div>
        `;
        const cssContent = `
.card {
  background-color: ${previewColors.cardBg};
  padding: 2rem;
  border-radius: 1rem;
  text-align: center;
}
.pill {
  background-color: ${previewColors.pillBg};
  color: ${previewColors.pillText};
  display: inline-block;
  padding: 0.25rem 1rem;
  border-radius: 9999px;
  text-align: center;
  font-size: 0.875rem;
}
.button {
  margin-top: 2rem;
  background-color: ${previewColors.buttonBg};
  color: ${previewColors.cardBg};
  padding: 0.5rem 1.5rem;
  border-radius: 0.5rem;
  font-weight: bold;
  border: none;
  cursor: pointer;
}
        `;
        const combined = `<!-- HTML -->\n${htmlContent}\n\n<!-- CSS
        -->\n<style>\n${cssContent}\n</style>`;
        downloadFile(combined, 'preview-card.html', 'text/html');
    }

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
            <header className="mb-6 text-center">
                <h1 className="text-3xl font-bold flex items-center justify-center">
                    <SparklesIcon />
                    <span className="ml-3">AI Color Palette Generator</span>
                </h1>
                <p className="text-text-secondary mt-1">Pick a base color, let Gemini design a
                palette, and preview it on a UI card.</p>
            </header>
            <div className="flex-grow flex flex-col lg:flex-row items-center justify-center
            gap-8">
                <div className="flex flex-col items-center gap-4">
                     <HexColorPicker color={baseColor} onChange={setBaseColor} className="!w-64
                     !h-64"/>
                     <div className="p-2 bg-surface rounded-md font-mono text-lg border border-
                     border" style={{color: baseColor}}>{baseColor}</div>
                      <button onClick={handleGenerate} disabled={isLoading} className="btn-primary
                      w-full flex items-center justify-center px-6 py-3">
                        {isLoading ? <LoadingSpinner /> : 'Generate Palette'}
                    </button>
                    {error && <p className="text-red-500 text-sm mt-2">{error}</p>}
                </div>
                <div className="flex flex-col gap-2 w-full max-w-sm">
```

```
                    <label className="text-sm font-medium text-text-secondary mb-2">Generated
                    Palette:</label>
                    {isLoading ? (
                         <div className="flex items-center justify-center h-48"><LoadingSpinner /></div>
                    ) : (
                        palette.map((color) => (
                            <div key={color} className="group flex items-center justify-between p-4 rounded-
                            md shadow-sm border border-border" style={{ backgroundColor: color }}>
                                <span className="font-mono font-bold text-black/70 mix-blend-
                                overlay">{color}</span>
                                <button onClick={() => navigator.clipboard.writeText(color)}
                                className="opacity-0 group-hover:opacity-100 transition-opacity bg-white/30
                                hover:bg-white/50 px-3 py-1 rounded text-xs text-black font-semibold backdrop-
                                blur-sm">Copy</button>
                            </div>
                        ))
                    )}
                    <div className="flex gap-2 mt-2">
                        <button onClick={downloadColors} className="flex-1 flex items-center justify-
                        center gap-2 text-sm py-2 bg-gray-100 border border-border rounded-md hover:bg-
                        gray-200"><ArrowDownTrayIcon className="w-4 h-4"/> Download Colors</button>
                        <button onClick={downloadCard} className="flex-1 flex items-center justify-
                        center gap-2 text-sm py-2 bg-gray-100 border border-border rounded-md hover:bg-
                        gray-200"><ArrowDownTrayIcon className="w-4 h-4"/> Download Card</button>
                    </div>
                </div>
                {!isLoading && <PreviewCard palette={palette} colors={previewColors}
                setColors={setPreviewColors} />}
            </div>
        </div>
    );
};

// ===== DevNotesStickyPanel_63.tsx =====


import React, { useState, useCallback } from 'react';
import { FileCodeIcon, SparklesIcon } from '../icons.tsx';
import { useLocalStorage } from '../../hooks/useLocalStorage.ts';
import { summarizeNotesStream } from '../../services/index.ts';
import { LoadingSpinner } from '../shared/index.tsx';
import { MarkdownRenderer } from '../shared/index.tsx';

type Note = {
    id: number;
    text: string;
    x: number;
    y: number;
    color: string;
}

const colors = ['bg-yellow-400 text-yellow-900', 'bg-green-400 text-green-900',
'bg-blue-400 text-blue-900', 'bg-pink-400 text-pink-900', 'bg-purple-400 text-
purple-900'];

export const DevNotesStickyPanel: React.FC = () => {
    const [notes, setNotes] = useLocalStorage<Note[]>('devcore_notes', []);
    const [dragging, setDragging] = useState<{ id: number; offsetX: number; offsetY:
    number } | null>(null);
    const [isSummarizing, setIsSummarizing] = useState(false);
    const [summary, setSummary] = useState('');
    const handleSummarize = useCallback(async () => {
```

```
    if (notes.length === 0) return;
    setIsSummarizing(true);
    setSummary('');
    try {
        const allNotesText = notes.map((n: Note) => `- ${n.text}`).join('\n');
        const stream = summarizeNotesStream(allNotesText);
        let fullResponse = '';
        for await (const chunk of stream) {
            fullResponse += chunk;
            setSummary(fullResponse);
        }
    } catch (error) {
        console.error(error);
        setSummary('Sorry, an error occurred while summarizing.');
    } finally {
        setIsSummarizing(false);
    }
}, [notes]);


const addNote = () => {
    const newNote: Note = {
        id: Date.now(),
        text: 'New note...',
        x: 50 + (notes.length % 10) * 20,
        y: 50 + (notes.length % 10) * 20,
        color: colors[notes.length % colors.length],
    };
    setNotes([...notes, newNote]);
};

const updateText = (id: number, text: string) => {
    setNotes(notes.map((n: Note) => n.id === id ? { ...n, text } : n));
};

const deleteNote = (id: number, e: React.MouseEvent) => {
    e.stopPropagation();
    setNotes(notes.filter((n: Note) => n.id !== id));
};

const onMouseDown = (e: React.MouseEvent<HTMLDivElement>, id: number) => {
    if((e.target as HTMLElement).tagName === 'TEXTAREA' || (e.target as
    HTMLElement).tagName === 'BUTTON') return;
    const noteElement = e.currentTarget;
    const rect = noteElement.getBoundingClientRect();
    setDragging({ id, offsetX: e.clientX - rect.left, offsetY: e.clientY - rect.top
    });
};

const onMouseMove = (e: React.MouseEvent<HTMLDivElement>) => {
    if (!dragging) return;
    const boardRect = e.currentTarget.getBoundingClientRect();
    setNotes(
        notes.map((n: Note) =>
            n.id === dragging.id
                ? { ...n, x: e.clientX - dragging.offsetX - boardRect.left, y: e.clientY -
                dragging.offsetY - boardRect.top }
                : n
        )
    );
};
const onMouseUp = () => setDragging(null);
```

```
    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
            <header className="mb-6 flex justify-between items-center">
                 <div>
                    <h1 className="text-3xl font-bold flex items-center"><FileCodeIcon /><span
                    className="ml-3">Dev Notes Sticky Panel</span></h1>
                    <p className="text-text-secondary mt-1">A place for your thoughts, todos, and
                    random ideas.</p>
                </div>
                <div className="flex gap-2">
                    <button onClick={handleSummarize} disabled={isSummarizing || notes.length === 0}
                    className="btn-primary flex items-center gap-2 px-4 py-2">
                        <SparklesIcon/> {isSummarizing ? 'Summarizing...' : 'AI Summarize'}
                    </button>
                    <button onClick={addNote} className="btn-primary px-6 py-2">Add Note</button>
                </div>
            </header>
            <div
                className="relative flex-grow bg-background border-2 border-dashed border-border
                rounded-lg overflow-hidden"
                onMouseMove={onMouseMove}
                onMouseUp={onMouseUp}
                onMouseLeave={onMouseUp}
            >
                {notes.map((note: Note) => (
                    <div
                        key={note.id}
                        className={`group absolute w-48 h-48 p-2 flex flex-col shadow-lg cursor-grab
                        active:cursor-grabbing rounded-md transition-transform duration-100 border
                        border-black/40 ${note.color}`}
                        style={{ top: note.y, left: note.x, transform: dragging?.id === note.id ?
                        'scale(1.05)' : 'scale(1)' }}
                        onMouseDown={e => onMouseDown(e, note.id)}
                    >
                         <button onClick={(e) => deleteNote(note.id, e)} className="absolute -top-2
                         -right-2 w-6 h-6 rounded-full bg-gray-700 text-white font-bold text-xs flex
                         items-center justify-center opacity-0 group-hover:opacity-100 hover:bg-red-500
                         transition-all">&times;</button>
                        <textarea
                            value={note.text}
                            onChange={(e) => updateText(note.id, e.target.value)}
                            className="w-full h-full bg-transparent resize-none focus:outline-none font-
                            medium p-1 placeholder:text-inherit/50"
                        />
                    </div>
                ))}
            </div>
            {(isSummarizing || summary) && (
                 <div className="fixed inset-0 bg-gray-900/80 backdrop-blur-sm z-50 flex items-
                 center justify-center" onClick={() => setSummary('')}>
                    <div className="w-full max-w-2xl bg-surface border border-border rounded-lg
                    shadow-2xl p-6" onClick={e => e.stopPropagation()}>
                        <h2 className="text-xl font-bold mb-4">AI Summary of Notes</h2>
                        {isSummarizing && !summary ? <LoadingSpinner /> : <MarkdownRenderer
                        content={summary} />}
                    </div>
                </div>
            )}
        </div>
    );
};
// ===== DigitalWhiteboard_61.tsx =====
```

```
import React, { useState, useCallback } from 'react';
import { SparklesIcon, DigitalWhiteboardIcon } from '../icons.tsx';
import { useLocalStorage } from '../../hooks/useLocalStorage.ts';
import { summarizeNotesStream } from '../../services/index.ts';
import { LoadingSpinner } from '../shared/index.tsx';
import { MarkdownRenderer } from '../shared/index.tsx';

type Note = {
    id: number;
    text: string;
    x: number;
    y: number;
    color: string;
}

const colors = ['bg-yellow-400', 'bg-green-400', 'bg-blue-400', 'bg-pink-400',
'bg-purple-400', 'bg-orange-400'];
const textColors = ['text-yellow-900', 'text-green-900', 'text-blue-900', 'text-
pink-900', 'text-purple-900', 'text-orange-900'];

export const DigitalWhiteboard: React.FC = () => {
    const [notes, setNotes] = useLocalStorage<Note[]>('devcore_whiteboard_notes',
    []);
    const [dragging, setDragging] = useState<{ id: number; offsetX: number; offsetY:
    number } | null>(null);
    const [isSummarizing, setIsSummarizing] = useState(false);
    const [summary, setSummary] = useState('');

    const handleSummarize = useCallback(async () => {
        if (notes.length === 0) return;
        setIsSummarizing(true);
        setSummary('');
        try {
            const allNotesText = notes.map((n: Note) => `- ${n.text}`).join('\n');
            const stream = summarizeNotesStream(allNotesText);
            let fullResponse = '';
            for await (const chunk of stream) {
                fullResponse += chunk;
                setSummary(fullResponse);
            }
        } catch (error) {
            console.error(error);
            setSummary('Sorry, an error occurred while summarizing.');
        } finally {
            setIsSummarizing(false);
        }
    }, [notes]);

    const addNote = () => {
        const newNote: Note = {
            id: Date.now(),
            text: 'New idea...',
            x: 50,
            y: 50,
            color: colors[notes.length % colors.length],
        };
        setNotes([...notes, newNote]);
    };

    const deleteNote = (id: number, e: React.MouseEvent) => {
        e.stopPropagation();
        setNotes(notes.filter((n) => n.id !== id));
```

```
};

const updateNote = (id: number, updates: Partial<Note>) => {
    setNotes(notes.map((n) => n.id === id ? { ...n, ...updates } : n));
};

const onMouseDown = (e: React.MouseEvent<HTMLDivElement>, id: number) => {
    const target = e.target as HTMLElement;
    if (target.tagName === 'TEXTAREA' || target.dataset.role === 'button') return;

    const noteElement = e.currentTarget;
    const rect = noteElement.getBoundingClientRect();
    setDragging({ id, offsetX: e.clientX - rect.left, offsetY: e.clientY - rect.top
    });
};

const onMouseMove = (e: React.MouseEvent<HTMLDivElement>) => {
    if (!dragging) return;
    const boardRect = e.currentTarget.getBoundingClientRect();
    updateNote(dragging.id, {
        x: e.clientX - dragging.offsetX - boardRect.left,
        y: e.clientY - dragging.offsetY - boardRect.top
    });
};

const onMouseUp = () => setDragging(null);

return (
    <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
        <header className="mb-6 flex justify-between items-center">
             <div>
                <h1 className="text-3xl font-bold flex items-center"><DigitalWhiteboardIcon
                /><span className="ml-3">Digital Whiteboard</span></h1>
                <p className="text-text-secondary mt-1">Organize your ideas with interactive
                sticky notes and AI summaries.</p>
            </div>
            <div className="flex gap-2">
                <button onClick={handleSummarize} disabled={isSummarizing || notes.length === 0}
                className="btn-primary flex items-center gap-2 px-4 py-2">
                    <SparklesIcon/> {isSummarizing ? 'Summarizing...' : 'AI Summarize'}
                </button>
                <button onClick={addNote} className="btn-primary px-6 py-2">Add Note</button>
            </div>
        </header>
        <div
            className="relative flex-grow bg-background border-2 border-dashed border-border
            rounded-lg overflow-hidden"
            onMouseMove={onMouseMove} onMouseUp={onMouseUp} onMouseLeave={onMouseUp}
        >
            {notes.map((note) => (
                <div
                    key={note.id}
                    className={`group absolute w-56 h-56 p-2 flex flex-col shadow-lg cursor-grab
                    active:cursor-grabbing rounded-md transition-transform duration-100 border
                    border-black/40 ${note.color} ${textColors[colors.indexOf(note.color)]}`}
                    style={{ top: note.y, left: note.x, transform: dragging?.id === note.id ?
                    'scale(1.05)' : 'scale(1)' }}
                    onMouseDown={e => onMouseDown(e, note.id)}
                >
                    <button data-role="button" onClick={(e) => deleteNote(note.id, e)}
                    className="absolute -top-2 -right-2 w-6 h-6 rounded-full bg-gray-700 text-white
                    font-bold text-xs flex items-center justify-center opacity-0 group-
```

```
                        hover:opacity-100 hover:bg-red-500 transition-all">&times;</button>
                        <textarea
                            value={note.text}
                            onChange={(e) => updateNote(note.id, { text: e.target.value })}
                            className="w-full h-full bg-transparent resize-none focus:outline-none font-
                            medium p-1"
                        />
                        <div data-role="button" className="flex-shrink-0 flex justify-center gap-1 p-1
                        opacity-0 group-hover:opacity-100 transition-opacity">
                            {colors.map((c, i) => <button key={c} onClick={() => updateNote(note.id, {
                            color: c })} className={`w-4 h-4 rounded-full ${c} border border-black/20
                            ${note.color === c ? 'ring-2 ring-offset-1 ring-black/50' : ''}`}/>)}
                        </div>
                    </div>
                ))}
            </div>
             {(isSummarizing || summary) && (
                 <div className="fixed inset-0 bg-gray-900/80 backdrop-blur-sm z-50 flex items-
                 center justify-center" onClick={() => setSummary('')}>
                    <div className="w-full max-w-2xl bg-surface border border-border rounded-lg
                    shadow-2xl p-6" onClick={e => e.stopPropagation()}>
                        <h2 className="text-xl font-bold mb-4">AI Summary of Notes</h2>
                        {isSummarizing && !summary ? <LoadingSpinner /> : <MarkdownRenderer
                        content={summary} />}
                    </div>
                </div>
             )}
        </div>
    );
};

// ===== JsonTreeNavigator_63.tsx =====


import React, { useState } from 'react';
import { FileCodeIcon } from '../icons.tsx';

type JsonNodeProps = {
    data: any;
    nodeKey: string;
    isRoot?: boolean;
}

const JsonNode: React.FC<JsonNodeProps> = ({ data, nodeKey, isRoot = false }) =>
{
    const [isOpen, setIsOpen] = useState(isRoot);
    const isObject = typeof data === 'object' && data !== null;

    const toggleOpen = () => setIsOpen(!isOpen);

    if (!isObject) {
        return (
            <div className="ml-4 pl-4 border-l border-border">
                <span className="text-purple-700">{nodeKey}: </span>
                <span className={typeof data === 'string' ? 'text-green-700' : 'text-
                orange-700'}>
                    {typeof data === 'string' ? `"${data}"` : String(data)}
                </span>
            </div>
        );
    }
    const entries = Object.entries(data);
```

```
    const bracket = Array.isArray(data) ? '[]' : '{}';

    return (
        <div className={`ml-4 ${!isRoot ? 'pl-4 border-l border-border' : ''}`}>
            <button onClick={toggleOpen} className="flex items-center cursor-pointer
            hover:bg-gray-100 rounded px-1">
                <span className={`transform transition-transform ${isOpen ? 'rotate-90' :
                'rotate-0'}`}>■</span>
                <span className="ml-1 text-purple-700">{nodeKey}:</span>
                <span className="ml-2 text-text-secondary">{bracket[0]}</span>
                {!isOpen && <span className="text-text-secondary">...{bracket[1]}</span>}
            </button>
            {isOpen && (
                <div>
                    {entries.map(([key, value]) => (
                        <JsonNode key={key} nodeKey={key} data={value} />
                    ))}
                    <div className="text-text-secondary ml-4">{bracket[1]}</div>
                </div>
            )}
        </div>
    );
};

export const JsonTreeNavigator: React.FC<{ initialData?: object }> = ({
initialData }) => {
    const defaultJson = '{\n  "id": "devcore-001",\n  "active": true,\n  "features":
    [\n    "ai-explainer",\n    "api-tester"\n  ],\n  "config": {\n    "theme":
    "dark",\n    "version": 1\n  }\n}';
    const [jsonInput, setJsonInput] = useState(initialData ?
    JSON.stringify(initialData, null, 2) : defaultJson);
    const [parsedData, setParsedData] = useState<any>(() => {
        try {
            return JSON.parse(jsonInput);
        } catch {
            return null;
        }
    });
    const [error, setError] = useState('');

    const parseJson = (input: string) => {
        try {
            const parsed = JSON.parse(input);
            setParsedData(parsed);
            setError('');
        } catch (e) {
            if (e instanceof Error) setError(e.message);
            setParsedData(null);
        }
    };

    const handleInputChange = (e: React.ChangeEvent<HTMLTextAreaElement>) => {
        setJsonInput(e.target.value);
        parseJson(e.target.value);
    }

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
            <header className="mb-6">
                <h1 className="text-3xl font-bold flex items-center">
                    <FileCodeIcon />
                    <span className="ml-3">JSON Tree Navigator</span>
```

```
                </h1>
                <p className="text-text-secondary mt-1">Paste your JSON data to visualize it as
                a collapsible tree.</p>
            </header>
            <div className="flex-grow flex flex-col gap-4 min-h-0">
                <div className="flex flex-col h-2/5 min-h-[200px]">
                    <label htmlFor="json-input" className="text-sm font-medium text-text-secondary
                    mb-2">JSON Input</label>
                    <textarea
                        id="json-input"
                        value={jsonInput}
                        onChange={handleInputChange}
                        className={`flex-grow p-4 bg-surface border ${error ? 'border-red-500' :
                        'border-border'} rounded-md resize-y font-mono text-sm focus:ring-2 focus:ring-
                        primary focus:outline-none`}
                    />
                    {error && <p className="text-red-500 text-xs mt-1">{error}</p>}
                </div>
                 <div className="flex flex-col flex-grow min-h-0">
                    <label className="text-sm font-medium text-text-secondary mb-2">Tree
                    View</label>
                    <div className="flex-grow p-4 bg-surface border border-border rounded-md
                    overflow-y-auto font-mono text-sm">
                        {parsedData ? <JsonNode data={parsedData} nodeKey="root" isRoot /> : <div
                        className="text-text-secondary">Enter valid JSON to view</div>}
                    </div>
                </div>
            </div>
        </div>
    );
};

// ===== LogicFlowBuilder_63.tsx =====


import React, { useState, useRef, useMemo, useCallback } from 'react';
import { ALL_FEATURES } from './index.ts';
import { FEATURE_TAXONOMY, generatePipelineCode } from
'../../services/index.ts';
import type { Feature } from '../../types.ts';
import { MapIcon, SparklesIcon, XMarkIcon } from '../icons.tsx';
import { LoadingSpinner, MarkdownRenderer } from '../shared/index.tsx';

type Node = {
    id: number;
    featureId: string;
    x: number;
    y: number;
};

type Link = {
    from: number;
    to: number;
};

const featuresMap = new Map(ALL_FEATURES.map(f => [f.id, f]));
const taxonomyMap = new Map(FEATURE_TAXONOMY.map(f => [f.id, f]));

const FeaturePaletteItem: React.FC<{ feature: Feature, onDragStart: (e:
React.DragEvent, featureId: string) => void }> = ({ feature, onDragStart }) => (
    <div
        draggable
```

```
        onDragStart={e => onDragStart(e, feature.id)}
        className="p-3 rounded-md bg-gray-50 border border-border flex items-center
        gap-3 cursor-grab hover:bg-gray-100 transition-colors"
    >
        <div className="text-primary flex-shrink-0">{feature.icon}</div>
        <div>
            <h4 className="font-bold text-sm text-text-primary">{feature.name}</h4>
            <p className="text-xs text-text-secondary">{feature.category}</p>
        </div>
    </div>
);

const NodeComponent: React.FC<{
    node: Node;
    feature: Feature;
    isTarget: boolean;
    onMouseDown: (e: React.MouseEvent, id: number) => void;
    onLinkStart: (e: React.MouseEvent, id: number) => void;
    onLinkEnd: (e: React.MouseEvent, id: number) => void;
    onNodeHover: (id: number | null) => void;
}> = ({ node, feature, isTarget, onMouseDown, onLinkStart, onLinkEnd,
onNodeHover }) => (
    <div
        className={`absolute w-52 bg-surface rounded-lg shadow-md border-2 cursor-grab
        active:cursor-grabbing flex flex-col transition-all duration-150 ${isTarget ?
        'border-green-500 ring-2 ring-green-300' : 'border-border'}`}
        style={{ left: node.x, top: node.y }}
        onMouseDown={e => onMouseDown(e, node.id)}
        onMouseUp={e => onLinkEnd(e, node.id)}
        onMouseEnter={() => onNodeHover(node.id)}
        onMouseLeave={() => onNodeHover(null)}
    >
        <div className="p-2 flex items-center gap-2 border-b border-border">
            <div className="w-5 h-5 text-primary">{feature.icon}</div>
            <span className="text-sm font-semibold truncate text-text-
            primary">{feature.name}</span>
        </div>
        <div className="relative p-3 text-xs text-text-secondary min-h-[40px] flex
        items-center justify-center">
            Workflow Node
            <div
                onMouseDown={e => onLinkStart(e, node.id)}
                className="absolute right-[-9px] top-1/2 -translate-y-1/2 w-4 h-4 bg-primary
                rounded-full border-2 border-surface cursor-crosshair hover:scale-125
                transition-transform"
                title="Drag to connect"
            />
        </div>
    </div>
);

const SVGGrid: React.FC = React.memo(() => (
    <svg width="100%" height="100%" className="absolute inset-0">
        <defs>
            <pattern id="smallGrid" width="10" height="10" patternUnits="userSpaceOnUse">
                <path d="M 10 0 L 0 0 0 10" fill="none" stroke="rgba(0, 0, 0, 0.05)"
                strokeWidth="0.5"/>
            </pattern>
            <pattern id="grid" width="50" height="50" patternUnits="userSpaceOnUse">
                <rect width="50" height="50" fill="url(#smallGrid)"/>
                <path d="M 50 0 L 0 0 0 50" fill="none" stroke="rgba(0, 0, 0, 0.1)"
                strokeWidth="1"/>
```

```
            </pattern>
        </defs>
        <rect width="100%" height="100%" fill="url(#grid)" />
    </svg>
));

export const LogicFlowBuilder: React.FC = () => {
    const [nodes, setNodes] = useState<Node[]>([]);
    const [links, setLinks] = useState<Link[]>([]);
    const [draggingNode, setDraggingNode] = useState<{ id: number; offsetX: number;
    offsetY: number } | null>(null);
    const [linking, setLinking] = useState<{ from: number; fromPos: { x: number; y:
    number }; toPos: { x: number; y: number } } | null>(null);
    const [hoveredNodeId, setHoveredNodeId] = useState<number | null>(null);
    const [generatedCode, setGeneratedCode] = useState('');
    const [isGenerating, setIsGenerating] = useState(false);
    const canvasRef = useRef<HTMLDivElement>(null);

    const handleGenerateCode = useCallback(async () => {
        setIsGenerating(true);
        setGeneratedCode('');

        const sortedNodeIds: number[] = [];
        const inDegree = new Map<number, number>();
        nodes.forEach(node => inDegree.set(node.id, 0));
        links.forEach(link => inDegree.set(link.to, (inDegree.get(link.to) || 0) + 1));

        const queue = nodes.filter(node => inDegree.get(node.id) === 0).map(n => n.id);

        while(queue.length > 0) {
            const u = queue.shift()!;
            sortedNodeIds.push(u);
            links.filter(l => l.from === u).forEach(l => {
                inDegree.set(l.to, (inDegree.get(l.to) || 0) - 1);
                if(inDegree.get(l.to) === 0) queue.push(l.to);
            })
        }

        const flowDescription = sortedNodeIds.map((id, index) => {
            const node = nodes.find(n => n.id === id)!;
            const featureInfo = taxonomyMap.get(node.featureId);
            return `Step ${index + 1}: Execute the '${featureInfo?.name}' tool. Description:
            ${featureInfo?.description}. Inputs: ${featureInfo?.inputs}.`;
        }).join('\n');

        try {
            const code = await generatePipelineCode(flowDescription);
            setGeneratedCode(code);
        } catch (e) {
            setGeneratedCode(`// Error generating code: ${e instanceof Error ? e.message :
            'Unknown error'}`);
        } finally {
            setIsGenerating(false);
        }

    }, [nodes, links]);

    const handleDragStart = (e: React.DragEvent, featureId: string) => {
        e.dataTransfer.setData('application/json', JSON.stringify({ featureId }));
    };

    const handleDrop = (e: React.DragEvent) => {
```

```
    e.preventDefault();
    if (!canvasRef.current) return;
    try {
        const data = e.dataTransfer.getData('application/json');
        if (!data) return;

        const { featureId } = JSON.parse(data);
        const canvasRect = canvasRef.current.getBoundingClientRect();
        const newNode: Node = {
            id: Date.now(),
            featureId,
            x: e.clientX - canvasRect.left,
            y: e.clientY - canvasRect.top,
        };
        setNodes(prev => [...prev, newNode]);
    } catch (error) {
        console.error("Failed to handle drop:", error);
    }
};

const handleNodeMouseDown = (e: React.MouseEvent, id: number) => {
    const node = nodes.find(n => n.id === id);
    if (!node || (e.target as HTMLElement).title === 'Drag to connect') return;
    setDraggingNode({ id, offsetX: e.nativeEvent.offsetX, offsetY:
    e.nativeEvent.offsetY });
};

const handleNodeHover = useCallback((id: number | null) => {
    setHoveredNodeId(id);
}, []);

const handleCanvasMouseMove = (e: React.MouseEvent) => {
    if (!canvasRef.current) return;
    const canvasRect = canvasRef.current.getBoundingClientRect();
    const mouseX = e.clientX - canvasRect.left;
    const mouseY = e.clientY - canvasRect.top;

    if (draggingNode) {
        setNodes(nodes.map(n => n.id === draggingNode.id ? { ...n, x: mouseX -
        draggingNode.offsetX, y: mouseY - draggingNode.offsetY } : n));
    }
    if (linking) {
         const hoveredNode = hoveredNodeId ? nodes.find(n => n.id === hoveredNodeId) :
         null;
        if (hoveredNode && hoveredNode.id !== linking.from) {
             setLinking({ ...linking, toPos: { x: hoveredNode.x, y: hoveredNode.y } });
        } else {
            setLinking({ ...linking, toPos: { x: mouseX, y: mouseY } });
        }
    }
};

const handleCanvasMouseUp = () => {
    setDraggingNode(null);
    setLinking(null);
};

const handleLinkStart = (e: React.MouseEvent, id: number) => {
    e.stopPropagation();
    const fromNode = nodes.find(n => n.id === id);
    if (!fromNode) return;
    setLinking({ from: id, fromPos: { x: fromNode.x, y: fromNode.y }, toPos: { x:
```

```
    fromNode.x, y: fromNode.y } });
};

const handleLinkEnd = (e: React.MouseEvent, id: number) => {
    e.stopPropagation();
    if (linking && linking.from !== id) {
        setLinks(prev => [...prev.filter(l => !(l.from === linking.from && l.to ===
        id)), { from: linking.from, to: id }]);
    }
    setLinking(null);
};

const nodePositions = useMemo(() => new Map(nodes.map(n => [n.id, { x: n.x, y:
n.y }])), [nodes]);
const isLinking = !!linking;

return (
    <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
        <header className="mb-6 flex justify-between items-start">
            <div>
                <h1 className="text-3xl font-bold flex items-center"><MapIcon /><span
                className="ml-3">Logic Flow Builder</span></h1>
                <p className="text-text-secondary mt-1">Visually build application logic flows
                and generate pipeline code.</p>
            </div>
            <button onClick={handleGenerateCode} disabled={isGenerating || nodes.length ===
            0} className="btn-primary flex items-center gap-2 px-4 py-2">
                <SparklesIcon /> {isGenerating ? 'Generating...' : 'Generate Code'}
            </button>
        </header>
        <div className="flex-grow flex gap-6 min-h-0">
            <aside className="w-72 flex-shrink-0 bg-surface border border-border p-4
            rounded-lg flex flex-col">
                <h3 className="font-bold mb-3 text-lg">Features</h3>
                <div className="flex-grow overflow-y-auto space-y-3 pr-2">
                    {ALL_FEATURES.map(feature => <FeaturePaletteItem key={feature.id}
                    feature={feature} onDragStart={handleDragStart} />)}
                </div>
            </aside>
            <main
                ref={canvasRef}
                className="flex-grow relative bg-background border-2 border-dashed border-border
                rounded-lg overflow-hidden"
                onDrop={handleDrop}
                onDragOver={e => e.preventDefault()}
                onMouseMove={handleCanvasMouseMove}
                onMouseUp={handleCanvasMouseUp}
                onMouseLeave={handleCanvasMouseUp}
            >
                <SVGGrid />
                <svg width="100%" height="100%" className="absolute inset-0 pointer-events-
                none">
                    {links.map((link, i) => {
                        const fromNode = nodePositions.get(link.from);
                        const toNode = nodePositions.get(link.to);
                        if (!fromNode || !toNode) return null;
                        return <line key={i} x1={fromNode.x} y1={fromNode.y} x2={toNode.x} y2={toNode.y}
                        stroke="var(--color-primary)" strokeWidth="2" markerEnd="url(#arrow)" />;
                    })}
                    {linking && <line x1={linking.fromPos.x} y1={linking.fromPos.y}
                    x2={linking.toPos.x} y2={linking.toPos.y} stroke="var(--color-primary)"
                    strokeWidth="2" strokeDasharray="5,5" />}
```

```
                        <defs><marker id="arrow" viewBox="0 0 10 10" refX="8" refY="5" markerWidth="6"
                        markerHeight="6" orient="auto-start-reverse"><path d="M 0 0 L 10 5 L 0 10 z"
                        fill="var(--color-primary)" /></marker></defs>
                    </svg>
                    {nodes.map(node => {
                        const feature = featuresMap.get(node.featureId);
                        return feature ? <NodeComponent key={node.id} node={node} feature={feature}
                        isTarget={isLinking && hoveredNodeId === node.id && linking.from !== node.id}
                        onMouseDown={handleNodeMouseDown} onLinkStart={handleLinkStart}
                        onLinkEnd={handleLinkEnd} onNodeHover={handleNodeHover} /> : null;
                    })}
                </main>
            </div>
            {(isGenerating || generatedCode) && (
                <div className="fixed inset-0 bg-gray-900/80 backdrop-blur-sm z-50 flex items-
                center justify-center" onClick={() => setGeneratedCode('')}>
                    <div className="w-full max-w-3xl h-3/4 bg-surface border border-border rounded-
                    lg shadow-2xl p-6 flex flex-col" onClick={e => e.stopPropagation()}>
                        <div className="flex justify-between items-center mb-4">
                            <h2 className="text-xl font-bold">Generated Pipeline Code</h2>
                            <button onClick={() => setGeneratedCode('')}><XMarkIcon/></button>
                        </div>
                        <div className="flex-grow bg-background border border-border rounded-md
                        overflow-auto">
                            {isGenerating && !generatedCode ? <div className="flex justify-center items-
                            center h-full"><LoadingSpinner /></div> : <MarkdownRenderer
                            content={'```javascript\n' + generatedCode + '\n```'} />}
                        </div>
                    </div>
                </div>
            )}
        </div>
    );
};

// ===== MetaTagEditor_63.tsx =====


import React, { useState, useMemo } from 'react';
import { CodeBracketSquareIcon } from '../icons.tsx';

type MetaData = {
    title: string;
    description: string;
    image: string;
    url: string;
}

const SocialCardPreview: React.FC<{ meta: MetaData }> = ({ meta }) => (
    <div className="w-full max-w-md mx-auto bg-surface border border-border
    rounded-2xl overflow-hidden shadow-lg">
        <div className="h-52 bg-gray-100 flex items-center justify-center">
            {meta.image ? <img src={meta.image} alt="Preview" className="w-full h-full
            object-cover" onError={(e) => e.currentTarget.style.display='none'}/> : <span
            className="text-text-secondary">Image Preview</span>}
        </div>
        <div className="p-4">
            <p className="text-xs text-text-secondary truncate">{new URL(meta.url ||
            'https://example.com').hostname}</p>
            <h3 className="font-bold text-text-primary truncate mt-1">{meta.title || 'Your
            Title Here'}</h3>
            <p className="text-sm text-text-secondary mt-1 line-clamp-2">{meta.description
```

```
            || 'A concise description of your content will appear here.'}</p>
        </div>
    </div>
);

export const MetaTagEditor: React.FC = () => {
    const [meta, setMeta] = useState<MetaData>({
        title: 'DevCore AI Toolkit', description: 'The ultimate toolkit for modern
        developers, powered by Gemini.',
        image: 'https://storage.googleapis.com/maker-studio-project-images-
        prod/programming_power_on_a_laptop_3a8f0bb1_39a9_4c2b_81f0_a74551480f2c.png',
        url: 'https://devcore.example.com'
    });

    const handleChange = (e: React.ChangeEvent<HTMLInputElement>) => {
        setMeta({ ...meta, [e.target.name]: e.target.value });
    };

    const generatedHtml = useMemo(() => {
        return `<!-- Primary Meta Tags -->
<title>${meta.title}</title>
<meta name="title" content="${meta.title}" />
<meta name="description" content="${meta.description}" />
<!-- Open Graph / Facebook -->
<meta property="og:type" content="website" />
<meta property="og:url" content="${meta.url}" />
<meta property="og:title" content="${meta.title}" />
<meta property="og:description" content="${meta.description}" />
<meta property="og:image" content="${meta.image}" />
<!-- Twitter -->
<meta property="twitter:card" content="summary_large_image" />
<meta property="twitter:url" content="${meta.url}" />
<meta property="twitter:title" content="${meta.title}" />
<meta property="twitter:description" content="${meta.description}" />
<meta property="twitter:image" content="${meta.image}" />`;
    }, [meta]);

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
            <header className="mb-6"><h1 className="text-3xl font-bold flex items-
            center"><CodeBracketSquareIcon /><span className="ml-3">Meta Tag
            Editor</span></h1><p className="text-text-secondary mt-1">Generate SEO & social
            media meta tags with a live preview.</p></header>
            <div className="flex-grow grid grid-cols-1 lg:grid-cols-2 xl:grid-cols-3 gap-6
            min-h-0">
                <div className="xl:col-span-1 flex flex-col gap-4 bg-surface border border-
                border p-6 rounded-lg overflow-y-auto">
                    <h3 className="text-xl font-bold">Metadata</h3>
                    <div><label className="block text-sm">Title</label><input type="text"
                    name="title" value={meta.title} onChange={handleChange} className="w-full mt-1
                    p-2 rounded bg-background border border-border"/></div>
                    <div><label className="block text-sm">Description</label><input type="text"
                    name="description" value={meta.description} onChange={handleChange}
                    className="w-full mt-1 p-2 rounded bg-background border border-border"/></div>
                    <div><label className="block text-sm">Canonical URL</label><input type="text"
                    name="url" value={meta.url} onChange={handleChange} className="w-full mt-1 p-2
                    rounded bg-background border border-border"/></div>
                    <div><label className="block text-sm">Social Image URL</label><input type="text"
                    name="image" value={meta.image} onChange={handleChange} className="w-full mt-1
                    p-2 rounded bg-background border border-border"/></div>
                </div>
                <div className="xl:col-span-1 flex flex-col">
```

```
                    <label className="text-sm font-medium text-text-secondary mb-2">Generated
                    HTML</label>
                    <div className="relative flex-grow"><pre className="w-full h-full bg-background
                    p-4 rounded-md text-primary text-sm overflow-auto">{generatedHtml}</pre><button
                    onClick={() => navigator.clipboard.writeText(generatedHtml)} className="absolute
                    top-2 right-2 px-2 py-1 bg-gray-100 hover:bg-gray-200 rounded-md text-
                    xs">Copy</button></div>
                </div>
                 <div className="hidden xl:flex flex-col items-center justify-center">
                    <label className="text-sm font-medium text-text-secondary mb-2">Live
                    Preview</label>
                    <SocialCardPreview meta={meta} />
                </div>
            </div>
        </div>
    );
};

// ===== ProjectMoodboard_63.tsx =====




import React, { useState } from 'react';
import { PhotoIcon } from '../icons.tsx';
import { useLocalStorage } from '../../hooks/useLocalStorage.ts';

type Note = {
    id: number;
    text: string;
    x: number;
    y: number;
    color: string;
}

const colors = ['bg-yellow-400', 'bg-green-400', 'bg-blue-400', 'bg-pink-400',
'bg-purple-400', 'bg-orange-400'];
const textColors = ['text-yellow-900', 'text-green-900', 'text-blue-900', 'text-
pink-900', 'text-purple-900', 'text-orange-900'];

export const ProjectMoodboard: React.FC = () => {
    const [notes, setNotes] = useLocalStorage<Note[]>('devcore_moodboard', []);
    const [dragging, setDragging] = useState<{ id: number; offsetX: number; offsetY:
    number } | null>(null);

    const addNote = () => {
        const newNote: Note = {
            id: Date.now(),
            text: 'New idea...',
            x: 50,
            y: 50,
            color: colors[notes.length % colors.length],
        };
        setNotes([...notes, newNote]);
    };

    const deleteNote = (id: number, e: React.MouseEvent) => {
        e.stopPropagation();
        setNotes(notes.filter((n) => n.id !== id));
    };

    const updateNote = (id: number, updates: Partial<Note>) => {
```

```
    setNotes(notes.map((n) => n.id === id ? { ...n, ...updates } : n));
};

const onMouseDown = (e: React.MouseEvent<HTMLDivElement>, id: number) => {
    const target = e.target as HTMLElement;
    if (target.tagName === 'TEXTAREA' || target.dataset.role === 'button') return;

    const noteElement = e.currentTarget;
    const rect = noteElement.getBoundingClientRect();
    setDragging({ id, offsetX: e.clientX - rect.left, offsetY: e.clientY - rect.top
    });
};

const onMouseMove = (e: React.MouseEvent<HTMLDivElement>) => {
    if (!dragging) return;
    const boardRect = e.currentTarget.getBoundingClientRect();
    updateNote(dragging.id, {
        x: e.clientX - dragging.offsetX - boardRect.left,
        y: e.clientY - dragging.offsetY - boardRect.top
    });
};

const onMouseUp = () => setDragging(null);

return (
    <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
        <header className="mb-6 flex justify-between items-center">
             <div>
                <h1 className="text-3xl font-bold flex items-center"><PhotoIcon /><span
                className="ml-3">Project Moodboard</span></h1>
                <p className="text-text-secondary mt-1">Organize your ideas with interactive
                sticky notes.</p>
            </div>
            <button onClick={addNote} className="btn-primary px-6 py-2">Add Note</button>
        </header>
        <div
            className="relative flex-grow bg-background border-2 border-dashed border-border
            rounded-lg overflow-hidden"
            onMouseMove={onMouseMove} onMouseUp={onMouseUp} onMouseLeave={onMouseUp}
        >
            {notes.map((note) => (
                <div
                    key={note.id}
                    className={`group absolute w-56 h-56 p-2 flex flex-col shadow-lg cursor-grab
                    active:cursor-grabbing rounded-md transition-transform duration-100 border
                    border-black/40 ${note.color} ${textColors[colors.indexOf(note.color)]}`}
                    style={{ top: note.y, left: note.x, transform: dragging?.id === note.id ?
                    'scale(1.05)' : 'scale(1)' }}
                    onMouseDown={e => onMouseDown(e, note.id)}
                >
                    <button data-role="button" onClick={(e) => deleteNote(note.id, e)}
                    className="absolute -top-2 -right-2 w-6 h-6 rounded-full bg-gray-700 text-white
                    font-bold text-xs flex items-center justify-center opacity-0 group-
                    hover:opacity-100 hover:bg-red-500 transition-all">&times;</button>
                    <textarea
                        value={note.text}
                        onChange={(e) => updateNote(note.id, { text: e.target.value })}
                        className="w-full h-full bg-transparent resize-none focus:outline-none font-
                        medium p-1"
                    />
                    <div data-role="button" className="flex-shrink-0 flex justify-center gap-1 p-1
                    opacity-0 group-hover:opacity-100 transition-opacity">
```

```
                            {colors.map((c, i) => <button key={c} onClick={() => updateNote(note.id, {
                            color: c })} className={`w-4 h-4 rounded-full ${c} border border-black/20
                            ${note.color === c ? 'ring-2 ring-offset-1 ring-black/50' : ''}`}/>)}
                        </div>
                    </div>
                ))}
            </div>
        </div>
    );
};

// ===== PromptCraftPad_63.tsx =====


import React, { useState, useEffect, useMemo } from 'react';
import { SparklesIcon } from '../icons.tsx';
import { useLocalStorage } from '../../hooks/useLocalStorage.ts';

type Prompt = {
    id: number;
    name: string;
    text: string;
}

export const PromptCraftPad: React.FC = () => {
    const [prompts, setPrompts] = useLocalStorage<Prompt[]>('devcore_prompts', [
        { id: 1, name: 'React Component Generator', text: 'Generate a React component
        named {name} that {description}. Style it with Tailwind CSS.'}
    ]);
    const [activePrompt, setActivePrompt] = useState<Prompt | null>(prompts[0] ||
    null);
    const [editingId, setEditingId] = useState<number | null>(null);
    const [tempName, setTempName] = useState('');
    const [variables, setVariables] = useState<Record<string, string>>({});

    const variableNames = useMemo(() => {
        if (!activePrompt) return [];
        return [...activePrompt.text.matchAll(/\{(\w+)\}/g)].map(match => match[1]);
    }, [activePrompt]);

    const renderedPrompt = useMemo(() => {
        if (!activePrompt) return '';
        return variableNames.reduce((acc, varName) => {
            return acc.replace(new RegExp(`\\{${varName}\\}`, 'g'), variables[varName] ||
            `{${varName}}`);
        }, activePrompt.text);
    }, [activePrompt, variables, variableNames]);

    useEffect(() => {
        if(!activePrompt && prompts.length > 0) setActivePrompt(prompts[0]);
        if (activePrompt) setActivePrompt(prompts.find((p: Prompt) => p.id ===
        activePrompt.id) || null);
    }, [prompts, activePrompt]);

    const handleTextChange = (e: React.ChangeEvent<HTMLTextAreaElement>) => {
        if (!activePrompt) return;
        const updatedPrompt = { ...activePrompt, text: e.target.value };
        setPrompts(prompts.map((p: Prompt) => p.id === updatedPrompt.id ? updatedPrompt
        : p));
    };

    const handleNameUpdate = (id: number, newName: string) => {
```

```
    setPrompts(prompts.map((p: Prompt) => p.id === id ? {...p, name: newName} : p));
    setEditingId(null);
};

const handleAddNew = () => {
    const newPrompt = { id: Date.now(), name: 'New Untitled Prompt', text: '' };
    setPrompts([...prompts, newPrompt]);
    setActivePrompt(newPrompt);
};

const handleDelete = (id: number) => {
    setPrompts(prompts.filter((p: Prompt) => p.id !== id));
    if(activePrompt?.id === id) setActivePrompt(prompts.length > 1 ? prompts[0] :
    null);
}

return (
    <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
        <header className="mb-6"><h1 className="text-3xl font-bold flex items-
        center"><SparklesIcon /><span className="ml-3">Prompt Craft Pad</span></h1><p
        className="text-text-secondary mt-1">Create, save, and manage your favorite AI
        prompts.</p></header>
        <div className="flex-grow flex gap-6 min-h-0">
            <aside className="w-1/3 bg-surface border border-border p-4 rounded-lg flex
            flex-col">
                <h3 className="font-bold mb-2">My Prompts</h3>
                <ul className="space-y-2 flex-grow overflow-y-auto">{prompts.map((p: Prompt) =>
                (<li key={p.id} className="group flex items-center justify-between"><div
                className={`w-full text-left rounded-md ${activePrompt?.id === p.id ? 'bg-
                primary/10' : ''}`}><button onClick={() => setActivePrompt(p)} onDoubleClick={()
                => {setEditingId(p.id); setTempName(p.name);}} className={`w-full text-left px-3
                py-2 ${activePrompt?.id === p.id ? 'text-primary' : 'hover:bg-gray-100'}`}>
                {editingId === p.id ? <input autoFocus value={tempName} onChange={e =>
                setTempName(e.target.value)} onBlur={() => handleNameUpdate(p.id, tempName)}
                onKeyDown={e => e.key === 'Enter' && handleNameUpdate(p.id, tempName)}
                className="bg-gray-100 text-text-primary w-full"/> : p.name}
                </button></div><button onClick={() => handleDelete(p.id)} className="ml-2 p-1
                text-text-secondary hover:text-red-500 opacity-0 group-
                hover:opacity-100">&times;</button></li>))}</ul>
                <div className="mt-4 pt-4 border-t border-border"><button onClick={handleAddNew}
                className="btn-primary w-full text-sm py-2">Add New Prompt</button></div>
            </aside>
            <main className="w-2/3 flex flex-col gap-4">
                {activePrompt ? (<>
                    <textarea value={activePrompt.text} onChange={handleTextChange} className="flex-
                    grow p-4 bg-surface border border-border rounded-md resize-none font-mono text-
                    sm focus:ring-2 focus:ring-primary focus:outline-none"/>
                    {variableNames.length > 0 && <div className="flex-shrink-0 bg-surface border
                    border-border p-4 rounded-lg"><h4 className="font-bold mb-2">Test
                    Variables</h4><div className="grid grid-cols-2 gap-2">{variableNames.map(v =>
                    (<div key={v}><label className="text-xs">{v}</label><input type="text"
                    value={variables[v] || ''} onChange={e => setVariables({...variables, [v]:
                    e.target.value})} className="w-full bg-background border border-border px-2 py-1
                    rounded text-sm"/></div>))}</div><h4 className="font-bold mt-4 mb-2">Live
                    Preview</h4><p className="text-sm p-2 bg-background rounded border border-
                    border">{renderedPrompt}</p></div>}
                </>) : (<div className="flex-grow flex items-center justify-center bg-background
                rounded-lg text-text-secondary border border-border">Select a prompt or create a
                new one.</div>)}
            </main>
        </div>
    </div>
```

```
    );
};

// ===== PwaManifestEditor_63.tsx =====


import React, { useState, useMemo } from 'react';
import { CodeBracketSquareIcon, ArrowDownTrayIcon } from '../icons.tsx';
import { downloadFile } from '../../services/index.ts';

type ManifestData = {
    name: string;
    short_name: string;
    start_url: string;
    scope: string;
    display: 'standalone' | 'fullscreen' | 'minimal-ui' | 'browser';
    orientation: 'any' | 'natural' | 'landscape' | 'portrait';
    background_color: string;
    theme_color: string;
}

const HomeScreenPreview: React.FC<{ manifest: ManifestData }> = ({ manifest })
=> (
    <div className="w-full max-w-xs mx-auto flex flex-col items-center">
        <div className="w-72 h-[550px] bg-gray-800 rounded-[40px] border-[10px] border-
        black shadow-2xl p-4 flex flex-col">
            <div className="flex-shrink-0 h-6 flex justify-between items-center px-4">
                <span className="text-xs font-bold" style={{color:
                manifest.theme_color}}>9:41</span>
                <div className="w-16 h-4 bg-black rounded-full" />
                <span className="text-xs font-bold" style={{color:
                manifest.theme_color}}>100%</span>
            </div>
            <div className="flex-grow grid grid-cols-4 gap-4 p-4">
                <div className="flex flex-col items-center gap-1">
                    <div className="w-14 h-14 bg-white rounded-xl flex items-center justify-center
                    text-3xl" style={{backgroundColor: manifest.background_color}}>
                        <span style={{color: manifest.theme_color}}>{manifest.short_name[0]}</span>
                    </div>
                    <p className="text-xs text-center text-white truncate
                    w-full">{manifest.short_name}</p>
                </div>
            </div>
        </div>
         <p className="text-xs text-text-secondary mt-2 text-center">Home Screen
         Preview</p>
    </div>
);


export const PwaManifestEditor: React.FC = () => {
    const [manifest, setManifest] = useState<ManifestData>({
        name: 'DevCore Progressive Web App', short_name: 'DevCore', start_url: '/',
        scope: '/',
        display: 'standalone', orientation: 'any', background_color: '#F5F7FA',
        theme_color: '#0047AB',
    });

    const handleChange = (e: React.ChangeEvent<HTMLInputElement |
    HTMLSelectElement>) => {
        setManifest({ ...manifest, [e.target.name]: e.target.value });
    };
```

```
const generatedJson = useMemo(() => {
    const fullManifest = { ...manifest, icons: [{"src": "icon-192.png", "type":
    "image/png", "sizes": "192x192"}, {"src": "icon-512.png", "type": "image/png",
    "sizes": "512x512"}] };
    return JSON.stringify(fullManifest, null, 2);
}, [manifest]);

return (
    <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
        <header className="mb-6"><h1 className="text-3xl font-bold flex items-
        center"><CodeBracketSquareIcon /><span className="ml-3">PWA Manifest
        Editor</span></h1><p className="text-text-secondary mt-1">Configure and generate
        the `manifest.json` file for your PWA.</p></header>
        <div className="flex-grow grid grid-cols-1 lg:grid-cols-2 xl:grid-cols-3 gap-6
        min-h-0">
            <div className="xl:col-span-1 flex flex-col gap-4 bg-surface border border-
            border p-6 rounded-lg overflow-y-auto">
                <h3 className="text-xl font-bold">Configuration</h3>
                <div><label className="block text-sm">App Name</label><input type="text"
                name="name" value={manifest.name} onChange={handleChange} className="w-full mt-1
                p-2 rounded bg-background border border-border"/></div>
                <div><label className="block text-sm">Short Name</label><input type="text"
                name="short_name" value={manifest.short_name} onChange={handleChange}
                className="w-full mt-1 p-2 rounded bg-background border border-border"/></div>
                <div><label className="block text-sm">Start URL</label><input type="text"
                name="start_url" value={manifest.start_url} onChange={handleChange}
                className="w-full mt-1 p-2 rounded bg-background border border-border"/></div>
                <div><label className="block text-sm">Scope</label><input type="text"
                name="scope" value={manifest.scope} onChange={handleChange} className="w-full
                mt-1 p-2 rounded bg-background border border-border"/></div>
                <div><label className="block text-sm">Display Mode</label><select name="display"
                value={manifest.display} onChange={handleChange} className="w-full mt-1 p-2
                rounded bg-background border border-
                border"><option>standalone</option><option>fullscreen</option><option>minimal-
                ui</option><option>browser</option></select></div>
                <div><label className="block text-sm">Orientation</label><select
                name="orientation" value={manifest.orientation} onChange={handleChange}
                className="w-full mt-1 p-2 rounded bg-background border border-border"><option>a
                ny</option><option>natural</option><option>landscape</option><option>portrait</o
                ption></select></div>
                 <div className="flex gap-4">
                    <div className="w-1/2"><label className="block text-sm">Background
                    Color</label><input type="color" name="background_color"
                    value={manifest.background_color} onChange={handleChange} className="w-full mt-1
                    h-10 rounded bg-background border border-border"/></div>
                    <div className="w-1/2"><label className="block text-sm">Theme
                    Color</label><input type="color" name="theme_color" value={manifest.theme_color}
                    onChange={handleChange} className="w-full mt-1 h-10 rounded bg-background border
                    border-border"/></div>
                 </div>
            </div>
            <div className="xl:col-span-1 flex flex-col">
                <div className="flex justify-between items-center mb-2">
                     <label className="text-sm font-medium text-text-secondary">Generated
                     manifest.json</label>
                     <button onClick={() => downloadFile(generatedJson, 'manifest.json',
                     'application/json')} className="flex items-center gap-1 px-3 py-1 bg-gray-100
                     text-xs rounded-md hover:bg-gray-200">
                        <ArrowDownTrayIcon className="w-4 h-4"/> Download
                    </button>
                </div>
                 <div className="relative flex-grow"><pre className="w-full h-full bg-background
```

```
                    p-4 rounded-md text-primary text-sm overflow-auto">{generatedJson}</pre></div>
                </div>
                <div className="hidden xl:flex flex-col items-center justify-center">
                    <label className="text-sm font-medium text-text-secondary mb-2">Live
                    Preview</label>
                    <HomeScreenPreview manifest={manifest} />
                </div>
            </div>
        </div>
    );
};

// ===== SchemaDesigner_63.tsx =====


import React, { useState, useRef, useMemo } from 'react';
import { MapIcon, ArrowDownTrayIcon, TrashIcon, PlusIcon } from '../icons.tsx';
import { downloadFile } from '../../services/index.ts';

type Column = { id: number; name: string; type: string; };
type Table = { id: number; name: string; columns: Column[]; x: number; y:
number; };

const classicBankSchema: Table[] = [
    { id: 1, name: 'customers', columns: [{ id: 1, name: 'customer_id', type:
    'INTEGER PRIMARY KEY' }, {id: 2, name: 'first_name', type: 'VARCHAR(100)'}, {id:
    3, name: 'last_name', type: 'VARCHAR(100)'}, {id: 4, name: 'address', type:
    'VARCHAR(255)'}, {id: 5, name: 'created_at', type: 'TIMESTAMP'}], x: 50, y: 50
    },
    { id: 2, name: 'accounts', columns: [{ id: 1, name: 'account_id', type: 'INTEGER
    PRIMARY KEY' }, {id: 2, name: 'customer_id', type: 'INTEGER'}, {id: 3, name:
    'account_type', type: 'VARCHAR(50)'}, {id: 4, name: 'balance', type:
    'DECIMAL(10, 2)'}, {id: 5, name: 'opened_at', type: 'TIMESTAMP'}], x: 400, y: 50
    },
    { id: 3, name: 'transactions', columns: [{ id: 1, name: 'transaction_id', type:
    'INTEGER PRIMARY KEY' }, {id: 2, name: 'account_id', type: 'INTEGER'}, {id: 3,
    name: 'amount', type: 'DECIMAL(10, 2)'}, {id: 4, name: 'transaction_type', type:
    'VARCHAR(50)'}, {id: 5, name: 'timestamp', type: 'TIMESTAMP'}], x: 225, y: 350
    },
];
const fintechSchema: Table[] = [
    { id: 1, name: 'users', columns: [{ id: 1, name: 'user_uuid', type: 'UUID
    PRIMARY KEY' }, {id: 2, name: 'email', type: 'VARCHAR(255) UNIQUE'}, {id: 3,
    name: 'phone_hash', type: 'VARCHAR(255)'}, {id: 4, name: 'kyc_status', type:
    'VARCHAR(50)'}], x: 50, y: 50 },
    { id: 2, name: 'wallets', columns: [{ id: 1, name: 'wallet_uuid', type: 'UUID
    PRIMARY KEY' }, {id: 2, name: 'user_uuid', type: 'UUID'}, {id: 3, name:
    'asset_code', type: 'VARCHAR(10)'}, {id: 4, name: 'balance', type: 'NUMERIC'}],
    x: 400, y: 50 },
    { id: 3, name: 'transfers', columns: [{ id: 1, name: 'transfer_uuid', type:
    'UUID PRIMARY KEY' }, {id: 2, name: 'from_wallet', type: 'UUID'}, {id: 3, name:
    'to_wallet', type: 'UUID'}, {id: 4, name: 'amount', type: 'NUMERIC'}, {id: 5,
    name: 'status', type: 'VARCHAR(50)'}], x: 225, y: 300 },
];

const exampleSchemas = [
    { name: 'Classic Bank', schema: classicBankSchema },
    { name: 'Fintech Startup', schema: fintechSchema }
];

const exportToSQL = (tables: Table[]) => {
    return tables.map(table => {
```

```
        const columnsSQL = table.columns.map(col => `  "${col.name}"
        ${col.type.toUpperCase()}`).join(',\n');
        return `CREATE TABLE "${table.name}" (\n${columnsSQL}\n);`;
    }).join('\n\n');
};

export const SchemaDesigner: React.FC = () => {
    const [tables, setTables] = useState<Table[]>(classicBankSchema);
    const [selectedTableId, setSelectedTableId] = useState<number | null>(null);
    const [dragging, setDragging] = useState<{ id: number; offsetX: number; offsetY:
    number } | null>(null);
    const canvasRef = useRef<HTMLDivElement>(null);

    const onMouseDown = (e: React.MouseEvent<HTMLDivElement>, id: number) => {
        const target = e.target as HTMLElement;
        if (target.tagName === 'INPUT' || target.tagName === 'SELECT' || target.tagName
        === 'BUTTON') return;
        setSelectedTableId(id);
        const tableElement = e.currentTarget;
        const rect = tableElement.getBoundingClientRect();
        const canvasRect = canvasRef.current!.getBoundingClientRect();
        setDragging({ id, offsetX: e.clientX - rect.left, offsetY: e.clientY - rect.top
        });
    };

    const onMouseMove = (e: React.MouseEvent<HTMLDivElement>) => {
        if (!dragging || !canvasRef.current) return;
        const canvasRect = canvasRef.current.getBoundingClientRect();
        const newX = e.clientX - canvasRect.left - dragging.offsetX;
        const newY = e.clientY - canvasRect.top - dragging.offsetY;
        setTables(tables.map(t => t.id === dragging.id ? { ...t, x: newX, y: newY } :
        t));
    };

    const onMouseUp = () => setDragging(null);

    const updateTable = (id: number, updates: Partial<Table>) =>
    setTables(tables.map(t => t.id === id ? { ...t, ...updates } : t));
    const addTable = () => {
        const newTable: Table = { id: Date.now(), name: 'new_table', columns: [{ id: 1,
        name: 'id', type: 'INTEGER PRIMARY KEY' }], x: 20, y: 20 };
        setTables([...tables, newTable]);
        setSelectedTableId(newTable.id);
    };
    const deleteTable = (id: number) => {
        setTables(tables.filter(t => t.id !== id));
        if (selectedTableId === id) setSelectedTableId(null);
    };
    const addColumn = (tableId: number) => {
        const newColumn = { id: Date.now(), name: 'new_column', type: 'VARCHAR' };
        updateTable(tableId, { columns: [...(tables.find(t => t.id === tableId)?.columns
        || []), newColumn] });
    };
    const updateColumn = (tableId: number, colId: number, field: 'name' | 'type',
    value: string) => {
        const table = tables.find(t => t.id === tableId);
        if (!table) return;
        const newColumns = table.columns.map(c => c.id === colId ? { ...c, [field]:
        value } : c);
        updateTable(tableId, { columns: newColumns });
    };
    const deleteColumn = (tableId: number, colId: number) => {
```

```
     const table = tables.find(t => t.id === tableId);
    if (!table) return;
    updateTable(tableId, { columns: table.columns.filter(c => c.id !== colId) });
};

const selectedTable = useMemo(() => tables.find(t => t.id === selectedTableId),
[tables, selectedTableId]);

return (
    <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
        <header className="mb-6"><h1 className="text-3xl font-bold flex items-
        center"><MapIcon /><span className="ml-3">Schema Designer</span></h1><p
        className="text-text-secondary mt-1">Visually design your database schema with
        drag-and-drop.</p></header>
        <div className="flex-grow flex gap-6 min-h-0">
            <main ref={canvasRef} className="flex-grow relative bg-background rounded-lg
            border-2 border-dashed border-border overflow-auto" onMouseMove={onMouseMove}
            onMouseUp={onMouseUp} onMouseLeave={onMouseUp}>
                {tables.map(table => (
                    <div key={table.id} className={`absolute w-64 bg-surface rounded-lg shadow-xl
                    border cursor-grab active:cursor-grabbing ${selectedTableId === table.id ?
                    'border-primary ring-2 ring-primary/50' : 'border-border'}`} style={{ top:
                    table.y, left: table.x }} onMouseDown={e => onMouseDown(e, table.id)}>
                        <h3 className="font-bold text-primary text-lg p-2 bg-gray-50 rounded-t-lg
                        border-b border-border">{table.name}</h3>
                        <div className="p-2 space-y-1 font-mono text-xs">
                            {table.columns.map(col => (<div key={col.id} className="flex justify-between
                            items-center"><span className="text-text-primary">{col.name}</span><span
                            className="text-text-secondary">{col.type}</span></div>))}
                        </div>
                    </div>
                ))}
            </main>
            <aside className="w-96 flex-shrink-0 flex flex-col gap-4">
                <div className="flex flex-col gap-2">
                     <button onClick={() => downloadFile(JSON.stringify(tables, null, 2),
                     'schema.json', 'application/json')} className="w-full text-sm py-2 bg-gray-100
                     border border-border rounded-md flex items-center justify-center gap-2 hover:bg-
                     gray-200"><ArrowDownTrayIcon className="w-4 h-4"/> Download JSON</button>
                     <button onClick={() => downloadFile(exportToSQL(tables), 'schema.sql',
                     'application/sql')} className="w-full btn-primary text-sm py-2 flex items-center
                     justify-center gap-2"><ArrowDownTrayIcon className="w-4 h-4"/> Download
                     SQL</button>
                </div>
                <div className="flex-grow bg-surface border border-border p-4 rounded-lg
                overflow-y-auto flex flex-col">
                    <h3 className="font-bold mb-2 text-lg">Editor</h3>
                    <div className="mb-2">
                         <select onChange={e => {
                            if (e.target.value) setTables(JSON.parse(e.target.value))
                         }} className="w-full p-2 bg-background border border-border rounded-md text-sm">
                            <option value="">Load Example Schema...</option>
                            {exampleSchemas.map(s => <option key={s.name}
                            value={JSON.stringify(s.schema)}>{s.name}</option>)}
                         </select>
                    </div>
                    {selectedTable ? (
                        <div className="flex-grow flex flex-col min-h-0">
                            <label className="text-xs font-semibold">Table Name</label>
                            <input value={selectedTable.name} onChange={e => updateTable(selectedTable.id, {
                            name: e.target.value })} className="w-full p-1 bg-background border border-
                            border rounded text-sm mb-2"/>
```

```
                                <label className="text-xs font-semibold">Columns</label>
                                <div className="flex-grow space-y-1 overflow-y-auto pr-1">
                                    {selectedTable.columns.map(col => (
                                        <div key={col.id} className="flex items-center gap-1">
                                            <input value={col.name} onChange={e => updateColumn(selectedTable.id, col.id,
                                            'name', e.target.value)} className="w-1/2 p-1 bg-background border border-border
                                            rounded text-xs"/>
                                            <input value={col.type} onChange={e => updateColumn(selectedTable.id, col.id,
                                            'type', e.target.value)} className="w-1/2 p-1 bg-background border border-border
                                            rounded text-xs"/>
                                            <button onClick={() => deleteColumn(selectedTable.id, col.id)} className="p-1
                                            text-red-500 hover:bg-red-100 rounded"><TrashIcon/></button>
                                        </div>
                                    ))}
                                </div>
                                <div className="mt-2 pt-2 border-t border-border flex gap-2">
                                     <button onClick={() => addColumn(selectedTable.id)} className="flex-1 text-sm
                                     py-1 bg-gray-100 border border-border rounded-md hover:bg-gray-200 flex items-
                                     center justify-center gap-1"><PlusIcon/>Add Column</button>
                                     <button onClick={() => deleteTable(selectedTable.id)} className="flex-1 text-sm
                                     py-1 bg-red-500/10 text-red-600 border border-red-200 rounded-md hover:bg-
                                     red-500/20">Delete Table</button>
                                </div>
                            </div>
                        ) : (<p className="text-sm text-text-secondary text-center flex-grow flex items-
                        center justify-center">Select a table to edit</p>)}
                        <button onClick={addTable} className="w-full text-sm py-2 bg-gray-100 border
                        border-border rounded-md hover:bg-gray-200 mt-auto flex items-center justify-
                        center gap-1"><PlusIcon/> Add Table</button>
                    </div>
                </aside>
            </div>
        </div>
    );
};

// ===== ScreenshotToComponent_63.tsx =====


import React, { useState, useCallback, useRef } from 'react';
import { generateComponentFromImageStream } from
'../../services/geminiService.ts';
import { PhotoIcon, ArrowDownTrayIcon } from '../icons.tsx';
import { LoadingSpinner, MarkdownRenderer } from '../shared/index.tsx';
import { fileToBase64, blobToDataURL, downloadFile } from
'../../services/index.ts';

type UploadedImage = {
    base64: string;
    dataUrl: string;
    mimeType: string;
}

export const ScreenshotToComponent: React.FC = () => {
    const [previewImage, setPreviewImage] = useState<string | null>(null);
    const [rawCode, setRawCode] = useState('');
    const [isLoading, setIsLoading] = useState(false);
    const [error, setError] = useState('');
    const fileInputRef = useRef<HTMLInputElement>(null);

    const handleGenerate = async (base64Image: string) => {
        setIsLoading(true);
```

```
    setError('');
    setRawCode('');
    try {
        const stream = generateComponentFromImageStream(base64Image);
        let fullResponse = '';
        for await (const chunk of stream) {
            fullResponse += chunk;
            setRawCode(fullResponse.replace(/^```(?:\w+\n)?/, '').replace(/```$/, ''));
        }
    } catch (err) {
        setError(err instanceof Error ? err.message : 'An unknown error occurred.');
    } finally {
        setIsLoading(false);
    }
};

const processImageBlob = async (blob: Blob) => {
    try {
        const [dataUrl, base64Image] = await Promise.all([blobToDataURL(blob),
        fileToBase64(blob as File)]);
        setPreviewImage(dataUrl);
        handleGenerate(base64Image);
    } catch (e) {
        setError('Could not process the image.');
    }
};

const handlePaste = useCallback(async (event: React.ClipboardEvent) => {
    const items = event.clipboardData.items;
    for (const item of items) {
        if (item.type.indexOf('image') !== -1) {
            const blob = item.getAsFile();
            if (blob) await processImageBlob(blob);
            return;
        }
    }
}, []);

const handleFileChange = async (event: React.ChangeEvent<HTMLInputElement>) => {
    const file = event.target.files?.[0];
    if (file) await processImageBlob(file);
};

return (
    <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
        <header className="mb-6"><h1 className="text-3xl font-bold flex items-
        center"><PhotoIcon /><span className="ml-3">AI Screenshot-to-
        Component</span></h1><p className="text-text-secondary mt-1">Paste or upload a
        screenshot of a UI element to generate React/Tailwind code.</p></header>
        <div className="flex-grow grid grid-cols-1 lg:grid-cols-2 gap-6 min-h-0">
            <div onPaste={handlePaste} className="flex flex-col items-center justify-center
            bg-surface p-6 rounded-lg border-2 border-dashed border-border focus:outline-
            none focus:border-primary overflow-y-auto" tabIndex={0}>
                {previewImage ? (<img src={previewImage} alt="Pasted content" className="max-w-
                full max-h-full object-contain rounded-md shadow-lg" />) : (<div
                className="text-center text-text-secondary">
                        <h2 className="text-xl font-bold text-text-primary">Paste an image here</h2>
                        <p className="mb-2">(Cmd/Ctrl + V)</p>
                        <p className="text-sm">or</p>
                        <button onClick={() => fileInputRef.current?.click()} className="mt-2 btn-
                        primary px-4 py-2 text-sm">Upload File</button>
                        <input type="file" ref={fileInputRef} onChange={handleFileChange}
```

```
                        accept="image/*" className="hidden"/>
                    </div>)}
                </div>
                <div className="flex flex-col h-full">
                    <div className="flex justify-between items-center mb-2">
                        <label className="text-sm font-medium text-text-secondary">Generated
                        Code</label>
                        {rawCode && !isLoading && (
                            <div className="flex items-center gap-2">
                                <button onClick={() => navigator.clipboard.writeText(rawCode)} className="px-3
                                py-1 bg-gray-100 text-xs rounded-md hover:bg-gray-200">Copy Code</button>
                                <button onClick={() => downloadFile(rawCode, 'Component.tsx',
                                'text/typescript')} className="flex items-center gap-1 px-3 py-1 bg-gray-100
                                text-xs rounded-md hover:bg-gray-200">
                                    <ArrowDownTrayIcon className="w-4 h-4" /> Download
                                </button>
                            </div>
                        )}
                    </div>
                    <div className="flex-grow bg-background border border-border rounded-md
                    overflow-y-auto">
                        {isLoading && (<div className="flex items-center justify-center
                        h-full"><LoadingSpinner /></div>)}
                        {error && <p className="p-4 text-red-500">{error}</p>}
                        {rawCode && !isLoading && <MarkdownRenderer
                        content={`\`\`\`tsx\n${rawCode}\n\`\`\`} />}
                        {!isLoading && !rawCode && !error && (<div className="text-text-secondary h-full
                        flex items-center justify-center">Generated component code will appear
                        here.</div>)}
                    </div>
                </div>
            </div>
        </div>
    );
};

// ===== SnippetVault_63.tsx =====


import React, { useState, useEffect, useMemo } from 'react';
import { LockClosedIcon, SparklesIcon, TrashIcon, ClipboardDocumentIcon,
ArrowDownTrayIcon } from '../icons.tsx';
import { useLocalStorage } from '../../hooks/useLocalStorage.ts';
import { enhanceSnippetStream } from '../../services/geminiService.ts';
import { LoadingSpinner } from '../shared/index.tsx';
import { downloadFile } from '../../services/index.ts';

type Snippet = {
    id: number; name: string; code: string; language: string; tags: string[];
}

const langToExt: Record<string, string> = {
    javascript: 'js',
    typescript: 'ts',
    python: 'py',
    css: 'css',
    html: 'html',
    json: 'json',
    markdown: 'md',
    plaintext: 'txt',
};
export const SnippetVault: React.FC = () => {
```

```
const [snippets, setSnippets] = useLocalStorage<Snippet[]>('devcore_snippets',
[{ id: 1, name: 'React Hook Boilerplate', language: 'javascript', code: `import
{ useState } from 'react';\n\nconst useCustomHook = () => {\n  const [value,
setValue] = useState(null);\n  return { value, setValue };\n};`, tags: ['react',
'hook'] }]);
const [activeSnippet, setActiveSnippet] = useState<Snippet | null>(null);
const [isEnhancing, setIsEnhancing] = useState(false);
const [searchTerm, setSearchTerm] = useState('');
const [isEditingName, setIsEditingName] = useState(false);

const filteredSnippets = useMemo(() => {
    if (!searchTerm) return snippets;
    const lowerSearch = searchTerm.toLowerCase();
    return snippets.filter((s: Snippet) =>
        s.name.toLowerCase().includes(lowerSearch) ||
        s.code.toLowerCase().includes(lowerSearch) ||
        (s.tags && s.tags.some(t => t.toLowerCase().includes(lowerSearch)))
    );
}, [snippets, searchTerm]);

useEffect(() => {
    if (!activeSnippet && filteredSnippets.length > 0)
    setActiveSnippet(filteredSnippets[0]);
    if (activeSnippet) setActiveSnippet(snippets.find((s: Snippet) => s.id ===
    activeSnippet.id) || null);
}, [snippets, activeSnippet, filteredSnippets]);

const updateSnippet = (snippet: Snippet) => {
    setSnippets(snippets.map((s: Snippet) => s.id === snippet.id ? snippet : s));
    setActiveSnippet(snippet);
};

const handleEnhance = async () => {
    if (!activeSnippet) return;
    setIsEnhancing(true);
    try {
        const stream = enhanceSnippetStream(activeSnippet.code);
        let fullResponse = '';
        for await (const chunk of stream) {
            fullResponse += chunk;
            updateSnippet({ ...activeSnippet, code: fullResponse.replace(/^```(?:\w+\n)?/,
            '').replace(/```$/, '') });
        }
    } finally { setIsEnhancing(false); }
};

const handleAddNew = () => {
    const newSnippet: Snippet = { id: Date.now(), name: 'New Snippet', language:
    'plaintext', code: '', tags: [] };
    setSnippets([...snippets, newSnippet]);
    setActiveSnippet(newSnippet);
};

const handleDelete = (id: number) => {
    setSnippets(snippets.filter((s: Snippet) => s.id !== id));
    if(activeSnippet?.id === id) setActiveSnippet(filteredSnippets.length > 1 ?
    filteredSnippets[0] : null);
};

const handleDownload = () => {
    if(!activeSnippet) return;
    const extension = langToExt[activeSnippet.language] || 'txt';
```

```
    const filename = `${activeSnippet.name.replace(/\s/g, '_')}.${extension}`;
    downloadFile(activeSnippet.code, filename);
}

const handleNameChange = (e: React.ChangeEvent<HTMLInputElement>) => {
    if (activeSnippet) updateSnippet({...activeSnippet, name: e.target.value});
};

const handleTagsChange = (e: React.KeyboardEvent<HTMLInputElement>) => {
    if (e.key === 'Enter' && activeSnippet) {
        const newTag = e.currentTarget.value.trim();
        if (newTag && !activeSnippet.tags.includes(newTag)) {
            updateSnippet({...activeSnippet, tags: [...(activeSnippet.tags ?? []),
            newTag]});
        }
        e.currentTarget.value = '';
    }
};

return (
    <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
        <header className="mb-6"><h1 className="text-3xl font-bold flex items-
        center"><LockClosedIcon /><span className="ml-3">Snippet Vault</span></h1><p
        className="text-text-secondary mt-1">Store, search, tag, and enhance your
        reusable code snippets with AI.</p></header>
        <div className="flex-grow flex gap-6 min-h-0">
            <aside className="w-1/3 bg-surface border border-border p-4 rounded-lg flex
            flex-col">
                <input type="text" placeholder="Search snippets..." value={searchTerm}
                onChange={e => setSearchTerm(e.target.value)} className="w-full px-3 py-1.5 mb-3
                rounded-md bg-background border border-border text-sm"/>
                <ul className="space-y-2 flex-grow overflow-y-auto
                pr-2">{filteredSnippets.map((s: Snippet) => (<li key={s.id} className="group
                flex items-center justify-between"><button onClick={() => setActiveSnippet(s)}
                className={`w-full text-left px-3 py-2 rounded-md ${activeSnippet?.id === s.id ?
                'bg-primary/10 text-primary' : 'hover:bg-gray-100'}`}>{s.name}</button><div
                className="flex opacity-0 group-hover:opacity-100 transition-opacity"><button
                onClick={() => navigator.clipboard.writeText(s.code)} className="ml-2 p-1 text-
                text-secondary hover:text-primary" title="Copy"><ClipboardDocumentIcon
                /></button><button onClick={() => handleDelete(s.id)} className="ml-2 p-1 text-
                text-secondary hover:text-red-500"
                title="Delete"><TrashIcon/></button></div></li>))}</ul>
                <div className="mt-4 pt-4 border-t border-border"><button onClick={handleAddNew}
                className="btn-primary w-full text-sm py-2">Add New Snippet</button></div>
            </aside>
            <main className="w-2/3 flex flex-col">
                {activeSnippet ? (<>
                    <div className="flex justify-between items-center mb-2">
                        {isEditingName ? <input type="text" value={activeSnippet.name}
                        onChange={handleNameChange} onBlur={() => setIsEditingName(false)} autoFocus
                        className="text-lg font-bold bg-gray-100 rounded px-2"/> : <h3 onDoubleClick={()
                        => setIsEditingName(true)} className="text-lg font-bold cursor-
                        pointer">{activeSnippet.name}</h3>}
                        <div className="flex gap-2">
                            <button onClick={handleEnhance} disabled={isEnhancing} className="flex items-
                            center gap-2 px-3 py-1 bg-purple-500 text-white font-bold text-xs rounded-md
                            disabled:bg-gray-400"><SparklesIcon /> AI Enhance</button>
                            <button onClick={() => navigator.clipboard.writeText(activeSnippet.code)}
                            className="px-3 py-1 bg-gray-100 text-xs rounded-md">Copy</button>
                            <button onClick={handleDownload} className="flex items-center gap-1 px-3 py-1
                            bg-gray-100 text-xs rounded-md"><ArrowDownTrayIcon className="w-4 h-4"/>
                            Download</button>
```

```
                        </div>
                    </div>
                    <textarea value={activeSnippet.code} onChange={e =>
                    updateSnippet({...activeSnippet, code: e.target.value})} className="flex-grow
                    p-4 bg-surface border border-border rounded-md resize-none font-mono text-sm
                    focus:ring-2 focus:ring-primary focus:outline-none"/>
                    <div className="mt-2 text-xs text-text-secondary">
                       <div className="flex items-center gap-2 flex-wrap">
                         <span className="font-bold">Tags:</span> {(activeSnippet.tags ?? []).map(t =>
                         <span key={t} className="bg-gray-200 px-2 py-0.5 rounded-full">{t}</span>)}
                         <input type="text" placeholder="+ Add tag" onKeyDown={handleTagsChange}
                         className="bg-transparent border-b border-border focus:outline-none
                         focus:border-primary w-24 text-xs px-1"/>
                       </div>
                    </div>
                </>) : (<div className="flex-grow flex items-center justify-center bg-background
                border border-border rounded-lg text-text-secondary">Select a snippet or create
                a new one.</div>)}
            </main>
        </div>
    </div>
  );
};

// ===== SvgPathEditor_63.tsx =====

import React, { useState, useRef } from 'react';
import { CodeBracketSquareIcon, ArrowDownTrayIcon } from '../icons.tsx';
import { downloadFile } from '../../services/index.ts';

const initialPath = "M 20 80 Q 100 20 180 80 T 340 80";

const parsePath = (d: string) => {
    const commands = d.match(/[a-df-z][^a-df-z]*/ig) || [];
    return commands.map((cmdStr, i) => {
        const command = cmdStr[0];
        const args = cmdStr.slice(1).trim().split(/[\s,]+/).map(parseFloat).filter(n =>
        !isNaN(n));
        const points = [];
        for (let j = 0; j < args.length; j += 2) {
            points.push({ x: args[j], y: args[j + 1] });
        }
        return { id: i, command, points };
    });
};

const buildPath = (parsed: any[]) => {
    return parsed.map(cmd => `${cmd.command} ${cmd.points.map((p:any) => `${p.x}
    ${p.y}`).join(' ')}`).join(' ');
};

export const SvgPathEditor: React.FC = () => {
    const [pathData, setPathData] = useState(initialPath);
    const svgRef = useRef<SVGSVGElement>(null);
    const [draggingPoint, setDraggingPoint] = useState<any>(null);
    const parsedPath = parsePath(pathData);

    const handleMouseDown = (e: React.MouseEvent, cmdIndex: number, pointIndex:
    number) => {
        e.stopPropagation();
        setDraggingPoint({ cmdIndex, pointIndex });
    };
```

```
    const handleMouseMove = (e: React.MouseEvent) => {
        if (!draggingPoint || !svgRef.current) return;
        const pt = new DOMPoint(e.clientX, e.clientY);
        const svgPoint = pt.matrixTransform(svgRef.current.getScreenCTM()?.inverse());

        const newParsedPath = parsedPath.map((cmd, cIdx) => {
            if (cIdx === draggingPoint.cmdIndex) {
                const newPoints = cmd.points.map((p, pIdx) => {
                    if (pIdx === draggingPoint.pointIndex) {
                        return { x: Math.round(svgPoint.x), y: Math.round(svgPoint.y) };
                    }
                    return p;
                });
                return { ...cmd, points: newPoints };
            }
            return cmd;
        });
        setPathData(buildPath(newParsedPath));
    };

    const handleMouseUp = () => setDraggingPoint(null);

    const handleAddPoint = (e: React.MouseEvent) => {
        if (!svgRef.current) return;
        const pt = new DOMPoint(e.clientX, e.clientY);
        const svgPoint = pt.matrixTransform(svgRef.current.getScreenCTM()?.inverse());
        const newPathData = `${pathData} L ${Math.round(svgPoint.x)}
        ${Math.round(svgPoint.y)}`;
        setPathData(newPathData);
    };

    const handleDownload = () => {
        const svgContent = `<svg viewBox="0 0 400 160"
        xmlns="http://www.w3.org/2000/svg">
  <path d="${pathData}" stroke="black" fill="transparent" stroke-width="2"/>
</svg>`;
        downloadFile(svgContent, 'path.svg', 'image/svg+xml');
    };

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
            <header className="mb-6"><h1 className="text-3xl font-bold flex items-
            center"><CodeBracketSquareIcon /><span className="ml-3">SVG Path
            Editor</span></h1><p className="text-text-secondary mt-1">Visually create and
            manipulate SVG path data by dragging points.</p></header>
            <div className="flex-grow grid grid-cols-1 lg:grid-cols-2 gap-6 h-full overflow-
            hidden">
                <div className="flex flex-col h-full overflow-y-auto">
                    <div className="flex justify-between items-center mb-2">
                        <label htmlFor="path-input" className="text-sm font-medium text-text-
                        secondary">Path Data (d attribute)</label>
                         <button onClick={handleDownload} className="flex items-center gap-1 px-3 py-1
                         bg-gray-100 text-xs rounded-md hover:bg-gray-200">
                            <ArrowDownTrayIcon className="w-4 h-4"/> Download SVG
                        </button>
                    </div>
                    <textarea id="path-input" value={pathData} onChange={(e) =>
                    setPathData(e.target.value)} className="h-24 p-4 bg-surface border border-border
                    rounded-md resize-y font-mono text-sm text-primary" />
                     <div className="flex-grow mt-4 p-4 bg-surface border-2 border-dashed border-
                     border rounded-md overflow-hidden flex items-center justify-center
                     min-h-[200px]">
```

```
                        <svg ref={svgRef} viewBox="0 0 400 160" className="w-full h-full cursor-
                        crosshair" onMouseMove={handleMouseMove} onMouseUp={handleMouseUp}
                        onMouseLeave={handleMouseUp} onDoubleClick={handleAddPoint}>
                           <rect width="400" height="160" fill="var(--color-background)" />
                            <path d={pathData} stroke="var(--color-primary)" fill="transparent"
                            strokeWidth="2" />
                            {parsedPath.flatMap((cmd, cmdIndex) =>
                                cmd.points.map((p, pointIndex) => (
                                    <circle
                                        key={`${cmd.id}-${pointIndex}`}
                                        cx={p.x}
                                        cy={p.y}
                                        r="5"
                                        fill={cmd.command.toLowerCase() === 'c' || cmd.command.toLowerCase() === 'q' ||
                                        cmd.command.toLowerCase() === 's' || cmd.command.toLowerCase() === 't' ?
                                        '#fde047' : '#f87171'}
                                        stroke="var(--color-surface)"
                                        strokeWidth="2"
                                        className="cursor-move hover:stroke-primary"
                                        onMouseDown={(e) => handleMouseDown(e, cmdIndex, pointIndex)}
                                    />
                                ))
                            )}
                        </svg>
                    </div>
                    <p className="text-xs text-center text-text-secondary mt-2">Double-click on the
                    canvas to add a new point.</p>
                </div>
                <div className="flex flex-col h-full">
                    <label className="text-sm font-medium text-text-secondary mb-2">Parsed
                    Commands</label>
                    <div className="flex-grow p-2 bg-background border border-border rounded-md
                    overflow-y-auto font-mono text-xs space-y-2">
                        {parsedPath.map(cmd => (
                            <div key={cmd.id} className="p-2 bg-surface rounded">
                                <span className="font-bold text-amber-600">{cmd.command}</span>
                                <span className="text-text-secondary"> {cmd.points.map(p =>
                                `(${p.x},${p.y})`).join(' ')}</span>
                            </div>
                        ))}
                    </div>
                </div>
            </div>
        </div>
    );
};

// ===== ThemeDesigner_63.tsx =====


import React, { useState, useCallback, useEffect } from 'react';
import { SparklesIcon, ArrowDownTrayIcon } from '../icons.tsx';
import { generateThemeFromDescription } from '../../services/geminiService.ts';
import type { ColorTheme } from '../../types.ts';
import { LoadingSpinner } from '../shared/index.tsx';
import { HexColorPicker } from 'react-colorful';
import { downloadFile } from '../../services/index.ts';

type ColorInputProps = {
    label: string;
    color: string;
    onChange: (color: string) => void;
```

```
}

const ColorInput: React.FC<ColorInputProps> = ({ label, color, onChange }) => {
    const [isOpen, setIsOpen] = useState(false);
    return (
         <div className="relative">
             <div onClick={() => setIsOpen(!isOpen)} className="flex items-center justify-
             between p-3 bg-background rounded-md cursor-pointer border-2 border-border
             hover:border-gray-300">
                <div className="flex items-center gap-3">
                     <div className="w-6 h-6 rounded border border-border" style={{ backgroundColor:
                     color }} />
                     <span className="text-sm font-medium text-text-primary">{label}</span>
                </div>
                <span className="font-mono text-sm text-text-secondary">{color}</span>
            </div>
            {isOpen && (
                <div className="absolute z-10 top-full mt-2 right-0">
                     <div className="fixed inset-0" onClick={() => setIsOpen(false)} />
                    <HexColorPicker color={color} onChange={onChange} />
                </div>
             )}
        </div>
    );
};

const randomPrompts = [
    'A serene forest at dawn',
    'A retro 8-bit video game',
    'A cyberpunk cityscape at night',
    'A warm, cozy coffee shop',
    'An arctic expedition with icy blues',
    'A vibrant coral reef',
];

export const ThemeDesigner: React.FC<{ initialPrompt?: string }> = ({
initialPrompt }) => {
    const [theme, setTheme] = useState<ColorTheme>({
        primary: '#0047AB', background: '#F5F7FA', surface: '#FFFFFF', textPrimary:
        '#111827', textSecondary: '#6B7280',
    });
    const [prompt, setPrompt] = useState(initialPrompt || 'A professional and clean
    corporate blue theme.');
    const [isLoading, setIsLoading] = useState(false);
    const [error, setError] = useState('');

    const handleGenerate = useCallback(async (p: string) => {
        if (!p.trim()) { setError('Please enter a description.'); return; }
        setIsLoading(true); setError('');
        try {
            const newTheme = await generateThemeFromDescription(p);
            setTheme(newTheme);
        } catch (err) {
            setError(err instanceof Error ? err.message : "An unknown error occurred.");
        } finally {
            setIsLoading(false);
        }
    }, []);

    const handleRandom = () => {
        const randomPrompt = randomPrompts[Math.floor(Math.random() *
        randomPrompts.length)];
```

```
        setPrompt(randomPrompt);
        handleGenerate(randomPrompt);
    };

    const handleColorChange = (key: keyof ColorTheme, color: string) => {
        setTheme(prev => ({...prev, [key]: color}));
    };

    const getExportContent = (type: 'css' | 'tailwind') => {
        if (type === 'css') {
            return `:root {\n` +
                   `  --primary: ${theme.primary};\n` +
                   `  --background: ${theme.background};\n` +
                   `  --surface: ${theme.surface};\n` +
                   `  --text-primary: ${theme.textPrimary};\n` +
                   `  --text-secondary: ${theme.textSecondary};\n` +
                   `}`;
        }
        return `// tailwind.config.js
module.exports = {
  theme: {
    extend: {
      colors: {
        primary: '${theme.primary}',
        background: '${theme.background}',
        surface: '${theme.surface}',
        'text-primary': '${theme.textPrimary}',
        'text-secondary': '${theme.textSecondary}',
      },
    },
  },
};`;
    };

    useEffect(() => {
        if (initialPrompt) handleGenerate(initialPrompt);
    }, [initialPrompt, handleGenerate]);

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
            <header className="mb-6">
                <h1 className="text-3xl font-bold flex items-center"><SparklesIcon /><span
                className="ml-3">AI Theme Designer</span></h1>
                <p className="text-text-secondary mt-1">Describe, generate, fine-tune, and
                export a color theme for your application.</p>
            </header>
            <div className="flex-grow grid grid-cols-1 md:grid-cols-2 gap-6 min-h-0">
                <div className="md:col-span-1 flex flex-col gap-4 bg-surface border border-
                border p-6 rounded-lg overflow-y-auto">
                    <h3 className="text-xl font-bold">Describe your theme</h3>
                    <textarea value={prompt} onChange={e => setPrompt(e.target.value)}
                    className="p-2 bg-background border border-border rounded-md resize-none text-sm
                    h-24" placeholder="e.g., A light, airy theme for a blog" />
                     <div className="flex gap-2">
                         <button onClick={() => handleGenerate(prompt)} disabled={isLoading}
                         className="btn-primary w-full flex items-center justify-center gap-2 px-4 py-2">
                            {isLoading ? <LoadingSpinner /> : 'Generate Theme'}
                        </button>
                         <button onClick={handleRandom} disabled={isLoading} className="px-4 py-2 bg-
                         gray-100 border border-border rounded-md hover:bg-gray-200" title="Random
                         Theme">■</button>
                     </div>
```

```
    {error && <p className="text-red-500 text-xs text-center">{error}</p>}

    <div className="mt-4 border-t border-border pt-4 space-y-2 flex-grow overflow-y-
    auto pr-2">
         <h3 className="text-lg font-bold">Fine-Tune Palette</h3>
         <ColorInput label="Primary" color={theme.primary} onChange={c =>
         handleColorChange('primary', c)} />
         <ColorInput label="Background" color={theme.background} onChange={c =>
         handleColorChange('background', c)} />
         <ColorInput label="Surface" color={theme.surface} onChange={c =>
         handleColorChange('surface', c)} />
         <ColorInput label="Text Primary" color={theme.textPrimary} onChange={c =>
         handleColorChange('textPrimary', c)} />
         <ColorInput label="Text Secondary" color={theme.textSecondary} onChange={c =>
         handleColorChange('textSecondary', c)} />
    </div>
    <div className="flex-shrink-0 pt-4 border-t border-border flex flex-col gap-2">
        <button onClick={() => downloadFile(getExportContent('css'), 'theme.css',
        'text/css')} className="flex-1 text-sm py-2 bg-gray-100 border border-border
        rounded-md flex items-center justify-center gap-2 hover:bg-gray-200">
            <ArrowDownTrayIcon className="w-4 h-4"/> Download CSS
        </button>
        <button onClick={() => downloadFile(getExportContent('tailwind'),
        'tailwind.config.js', 'application/javascript')} className="flex-1 text-sm py-2
        bg-gray-100 border border-border rounded-md flex items-center justify-center
        gap-2 hover:bg-gray-200">
             <ArrowDownTrayIcon className="w-4 h-4"/> Download Tailwind Config
        </button>
    </div>
</div>
<div className="md:col-span-1 rounded-lg p-8 overflow-y-auto border border-
border" style={{ backgroundColor: theme.background, color: theme.textPrimary }}>
     <h3 className="text-2xl font-bold mb-6" style={{ color: theme.textPrimary
     }}>Live Preview</h3>
     <div className="p-6 rounded-lg grid grid-cols-1 md:grid-cols-2 gap-6" style={{
     backgroundColor: theme.surface }}>
        <div className="space-y-4">
            <h4 className="text-lg font-bold">Sample Card</h4>
            <p className="text-sm" style={{color: theme.textSecondary}}>This is a sample
            card to demonstrate the theme colors. It contains a primary button and some
            secondary text.</p>
            <button className="px-4 py-2 rounded-md font-bold transition-colors" style={{
            backgroundColor: theme.primary, color: theme.textSecondary.includes('#F') ?
            '#000' : '#FFF' }}>Primary Button</button>
        </div>
         <div className="space-y-4">
            <input type="text" placeholder="Text input" className="w-full px-3 py-2 rounded-
            md border" style={{backgroundColor: theme.background, borderColor:
            theme.primary, color: theme.textPrimary}} />
            <div className="p-3 border rounded" style={{borderColor: theme.primary, color:
            theme.textSecondary}}>
                <p>A bordered container.</p>
            </div>
         </div>
     </div>
     <div className="mt-6">
        <h4 className="text-lg font-bold mb-2">Typography</h4>
        <h1>Heading 1</h1>
        <h2>Heading 2</h2>
        <p style={{color: theme.textPrimary}}>This is a paragraph of primary text. Lorem
        ipsum dolor sit amet, consectetur adipiscing elit.</p>
        <p style={{color: theme.textSecondary}}>This is secondary text, for less
```

```
                        important information.</p>
                    </div>
                </div>
            </div>
        </div>
    );
};

// ===== VisualGitTree_63.tsx =====


import React, { useState, useCallback, useEffect, useMemo } from 'react';
import { GitBranchIcon, ArrowDownTrayIcon } from '../icons.tsx';
import { generateChangelogFromLogStream } from
'../../services/geminiService.ts';
import { LoadingSpinner, MarkdownRenderer } from '../shared/index.tsx';
import { downloadFile } from '../../services/index.ts';

const exampleLog = `* commit 3a4b5c6d7e8f9g0h1i2j3k4l5m6n7o8p9q0r (HEAD -> main,
origin/main)
|\\  Merge: 1a2b3c4 2d3e4f5
| | Author: Dev One <dev.one@example.com>
| | Date:   Mon Jul 15 11:30:00 2024 -0400
| |
| |     feat: Implement collapsible sidebar navigation
| |
* | commit 2d3e4f5g6h7i8j9k0l1m2n3o4p5q6r7s8t9u (feature/new-sidebar)
| | Author: Dev Two <dev.two@example.com>
| | Date:   Mon Jul 15 10:00:00 2024 -0400
| |
| |     feat: Add icons to sidebar items
| |
* | commit 1a2b3c4d5e6f7g8h9i0j1k2l3m4n5o6p7q8r
|/  Author: Dev One <dev.one@example.com>
|   Date:   Fri Jul 12 16:45:00 2024 -0400
|
|       fix: Correct user authentication bug`;

const CommitGraph = ({ logInput }: { logInput: string }) => {
    const commits = useMemo(() => {
        const lines = logInput.split('\n');
        const parsedCommits: any[] = [];
        let currentCommit: any = null;

        lines.forEach(line => {
            const commitMatch = line.match(/^.?[\\|/ ]*\* commit (\w+)(.*)/);
            if (commitMatch) {
                if (currentCommit) parsedCommits.push(currentCommit);
                currentCommit = {
                    hash: commitMatch[1],
                    shortHash: commitMatch[1].substring(0, 7),
                    refs: commitMatch[2].trim(),
                    message: '',
                    author: '',
                };
            } else if (currentCommit) {
                 if (line.includes('Author:')) currentCommit.author =
                 line.split('Author:')[1].trim();
                 else if (line.trim().length > 0 && !line.match(/^[\\|/ ]*[\\|/ ]/)) {
                     currentCommit.message += line.trim() + ' ';
                 }
            }
```

```
        });
        if (currentCommit) parsedCommits.push(currentCommit);

        return parsedCommits.map((c, i) => ({ ...c, x: 50, y: 50 + i * 60 }));
    }, [logInput]);

    return (
         <svg width="100%" height={50 + commits.length * 60} className="min-h-[200px]">
            {commits.map((commit, i) => {
                const parent = commits[i + 1];
                return (
                    <g key={commit.hash}>
                        {parent && <line x1={commit.x} y1={commit.y} x2={parent.x} y2={parent.y}
                        stroke="var(--color-border)" strokeWidth="2" />}
                        <g className="group cursor-pointer">
                            <circle cx={commit.x} cy={commit.y} r="8" fill="var(--color-primary)"
                            stroke="var(--color-surface)" strokeWidth="3" />
                            <foreignObject x={commit.x + 20} y={commit.y - 25} width="350" height="50">
                                <div className="text-sm p-1">
                                    <p className="font-bold truncate text-text-primary">{commit.message}</p>
                                    <p className="text-xs text-text-secondary font-mono">{commit.shortHash} <span
                                    className="text-amber-600">{commit.refs}</span></p>
                                </div>
                            </foreignObject>
                            <title>{`Commit: ${commit.hash}\nAuthor:
                            ${commit.author}\n\n${commit.message}`}</title>
                        </g>
                    </g>
                );
            })}
        </svg>
    );
};

export const VisualGitTree: React.FC<{ logInput?: string }> = ({ logInput:
initialLogInput }) => {
    const [logInput, setLogInput] = useState(initialLogInput || exampleLog);
    const [analysis, setAnalysis] = useState('');
    const [isLoading, setIsLoading] = useState(false);
    const [error, setError] = useState('');

    const handleAnalyze = useCallback(async (logToAnalyze: string) => {
        if (!logToAnalyze.trim()) {
            setError('Please paste git log output.');
            return;
        }
        setIsLoading(true);
        setError('');
        setAnalysis('');
        try {
            const stream = generateChangelogFromLogStream(logToAnalyze);
            let fullResponse = '';
            for await (const chunk of stream) {
                fullResponse += chunk;
                setAnalysis(fullResponse);
            }
        } catch (err) {
            const errorMessage = err instanceof Error ? err.message : 'An unknown error
            occurred.';
            setError(`Failed to analyze log: ${errorMessage}`);
        } finally {
            setIsLoading(false);
```

```
    }
}, []);

useEffect(() => {
    if (initialLogInput) {
        setLogInput(initialLogInput);
        handleAnalyze(initialLogInput);
    }
}, [initialLogInput, handleAnalyze]);

return (
    <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
        <header className="mb-6">
            <h1 className="text-3xl font-bold flex items-center">
                <GitBranchIcon />
                <span className="ml-3">Visual Git Tree</span>
            </h1>
            <p className="text-text-secondary mt-1">Paste your `git log --graph` output to
            visualize the history and get an AI summary.</p>
        </header>
        <div className="flex-grow grid grid-cols-1 lg:grid-cols-2 gap-6 h-full overflow-
        hidden">
            <div className="flex flex-col h-full">
                <label htmlFor="log-input" className="text-sm font-medium text-text-secondary
                mb-2">Git Log Output</label>
                <textarea
                    id="log-input"
                    value={logInput}
                    onChange={(e) => setLogInput(e.target.value)}
                    placeholder="Paste your git log output here..."
                    className="flex-grow p-4 bg-surface border border-border rounded-md resize-none
                    font-mono text-sm"
                />
                <button
                    onClick={() => handleAnalyze(logInput)}
                    disabled={isLoading}
                    className="btn-primary mt-4 w-full flex items-center justify-center px-6 py-3"
                >
                    {isLoading ? <LoadingSpinner /> : 'Analyze & Summarize'}
                </button>
            </div>
            <div className="flex flex-col h-full gap-4">
                <div className="flex flex-col h-1/2">
                    <label className="text-sm font-medium text-text-secondary mb-2">Commit
                    Graph</label>
                    <div className="flex-grow p-2 bg-surface border border-border rounded-md
                    overflow-auto">
                        <CommitGraph logInput={logInput} />
                    </div>
                </div>
                 <div className="flex flex-col h-1/2">
                    <div className="flex justify-between items-center mb-2">
                        <label className="text-sm font-medium text-text-secondary">AI Summary</label>
                        {analysis && !isLoading && (
                            <button onClick={() => downloadFile(analysis, 'summary.md', 'text/markdown')}
                            className="flex items-center gap-1 px-3 py-1 bg-gray-100 text-xs rounded-md
                            hover:bg-gray-200">
                                <ArrowDownTrayIcon className="w-4 h-4"/> Download Summary
                            </button>
                        )}
                    </div>
                    <div className="flex-grow p-4 bg-background border border-border rounded-md
```

```
                        overflow-y-auto">
                            {isLoading && <div className="flex items-center justify-center
                            h-full"><LoadingSpinner /></div>}
                            {error && <p className="text-red-500">{error}</p>}
                            {analysis && !isLoading && <MarkdownRenderer content={analysis} />}
                            {!isLoading && !analysis && !error && <div className="text-text-secondary h-full
                            flex items-center justify-center">AI summary will appear here.</div>}
                        </div>
                    </div>
                </div>
            </div>
        </div>
    );
};

// ===== WorkerThreadDebugger_63.tsx =====


import React, { useState, useCallback, useEffect } from 'react';
import { BugAntIcon, ArrowDownTrayIcon } from '../icons.tsx';
import { analyzeConcurrencyStream } from '../../services/geminiService.ts';
import { LoadingSpinner, MarkdownRenderer } from '../shared/index.tsx';
import { downloadFile } from '../../services/index.ts';

const exampleCode = `// main.js
const worker = new Worker('worker.js');

// This object is sent back and forth.
// A race condition can occur because both threads
// read the counter, increment it, and send it back.
// The final value depends on which thread's message
// is processed last.
const data = { counter: 0 };

worker.onmessage = function(e) {
  // Main thread reads and updates
  data.counter = e.data.counter;
  console.log('Main received:', data.counter);
  data.counter++;
  worker.postMessage(data);
};

// Start the process
console.log('Main starting with:', data.counter);
data.counter++;
worker.postMessage(data);


// worker.js
// onmessage = function(e) {
//   // Worker reads and updates
//   let receivedCounter = e.data.counter;
//   console.log('Worker received:', receivedCounter);
//   receivedCounter++;
//   postMessage({ counter: receivedCounter });
// }
`;

export const WorkerThreadDebugger: React.FC<{ codeInput?: string }> = ({
codeInput: initialCode }) => {
    const [codeInput, setCodeInput] = useState(initialCode || exampleCode);
    const [analysis, setAnalysis] = useState('');
```

```
const [isLoading, setIsLoading] = useState(false);
const [error, setError] = useState('');

const handleAnalyze = useCallback(async (codeToAnalyze: string) => {
    if (!codeToAnalyze.trim()) {
        setError('Please paste some code to analyze.');
        return;
    }
    setIsLoading(true);
    setError('');
    setAnalysis('');
    try {
        const stream = analyzeConcurrencyStream(codeToAnalyze);
        let fullResponse = '';
        for await (const chunk of stream) {
            fullResponse += chunk;
            setAnalysis(fullResponse);
        }
    } catch (err) {
        const errorMessage = err instanceof Error ? err.message : 'An unknown error
        occurred.';
        setError(`Failed to analyze code: ${errorMessage}`);
    } finally {
        setIsLoading(false);
    }
}, []);

useEffect(() => {
    if (initialCode) {
        setCodeInput(initialCode);
        handleAnalyze(initialCode);
    }
}, [initialCode, handleAnalyze]);

return (
    <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
        <header className="mb-6">
            <h1 className="text-3xl font-bold flex items-center">
                <BugAntIcon />
                <span className="ml-3">AI Concurrency Analyzer</span>
            </h1>
            <p className="text-text-secondary mt-1">Analyze JavaScript code for potential
            Web Worker concurrency issues.</p>
        </header>
        <div className="flex-grow flex flex-col gap-4 min-h-0">
            <div className="flex flex-col flex-1 min-h-0">
                <label htmlFor="code-input" className="text-sm font-medium text-text-secondary
                mb-2">JavaScript Code</label>
                <textarea
                    id="code-input"
                    value={codeInput}
                    onChange={(e) => setCodeInput(e.target.value)}
                    placeholder="Paste your worker-related JS code here..."
                    className="flex-grow p-4 bg-surface border border-border rounded-md resize-none
                    font-mono text-sm"
                />
            </div>
             <div className="flex-shrink-0">
                <button
                    onClick={() => handleAnalyze(codeInput)}
                    disabled={isLoading}
                    className="btn-primary w-full max-w-xs mx-auto flex items-center justify-center
```

```
                        px-6 py-3"
                    >
                        {isLoading ? <LoadingSpinner /> : 'Analyze Code'}
                    </button>
                </div>
                <div className="flex flex-col flex-1 min-h-0">
                    <div className="flex justify-between items-center mb-2">
                        <label className="text-sm font-medium text-text-secondary">AI Analysis</label>
                        {analysis && !isLoading && (
                             <button onClick={() => downloadFile(analysis, 'analysis.md', 'text/markdown')}
                             className="flex items-center gap-1 px-3 py-1 bg-gray-100 text-xs rounded-md
                             hover:bg-gray-200">
                                <ArrowDownTrayIcon className="w-4 h-4"/> Download
                            </button>
                        )}
                    </div>
                    <div className="flex-grow p-4 bg-background border border-border rounded-md
                    overflow-y-auto">
                        {isLoading && <div className="flex items-center justify-center
                        h-full"><LoadingSpinner /></div>}
                        {error && <p className="text-red-500">{error}</p>}
                        {analysis && !isLoading && <MarkdownRenderer content={analysis} />}
                        {!isLoading && !analysis && !error && <div className="text-text-secondary h-full
                        flex items-center justify-center">Analysis will appear here.</div>}
                    </div>
                </div>
            </div>
        </div>
    );
};

// ===== LogicFlowBuilder_64.tsx =====


import React, { useState, useCallback, useRef } from 'react';
import { ALL_FEATURES } from './index.ts';
import { FEATURE_TAXONOMY, generatePipelineCode } from
'../../services/index.ts';
import { featureToFunctionMap } from '../../services/pipelineTools.ts';
import type { Feature } from '../../types.ts';
import { MapIcon, SparklesIcon, XMarkIcon, Bars3Icon } from '../icons.tsx';
import { LoadingSpinner, MarkdownRenderer } from '../shared/index.tsx';

interface PipelineNode {
    id: number;
    featureId: string;
}

const featuresMap = new Map(ALL_FEATURES.map(f => [f.id, f]));
const taxonomyMap = new Map(FEATURE_TAXONOMY.map(f => [f.id, f]));

const FeaturePaletteItem: React.FC<{ feature: Feature, onDragStart: (e:
React.DragEvent, featureId: string) => void }> = ({ feature, onDragStart }) => (
    <div
        draggable
        onDragStart={e => onDragStart(e, feature.id)}
        className="p-3 rounded-md bg-gray-50 border border-border flex items-center
        gap-3 cursor-grab hover:bg-gray-100 transition-colors"
    >
        <div className="text-primary flex-shrink-0">{feature.icon}</div>
        <div>
            <h4 className="font-bold text-sm text-text-primary">{feature.name}</h4>
```

```
            <p className="text-xs text-text-secondary">{feature.category}</p>
        </div>
    </div>
);

const PipelineNodeComponent: React.FC<{
    node: PipelineNode;
    feature: Feature;
    index: number;
    onDragStart: () => void;
    onDragEnter: () => void;
    onDragEnd: () => void;
    onRemove: () => void;
}> = ({ node, feature, index, onDragStart, onDragEnter, onDragEnd, onRemove })
=> (
    <div
        draggable
        onDragStart={onDragStart}
        onDragEnter={onDragEnter}
        onDragEnd={onDragEnd}
        onDragOver={(e) => e.preventDefault()}
        className="w-full bg-surface rounded-lg shadow-md border-2 border-border cursor-
        grab active:cursor-grabbing flex items-center p-3 gap-4"
    >
        <div className="flex-shrink-0 w-8 h-8 rounded-full bg-primary text-white flex
        items-center justify-center font-bold text-lg">{index + 1}</div>
        <div className="w-6 h-6 text-text-secondary">{feature.icon}</div>
        <span className="flex-grow font-semibold truncate text-text-
        primary">{feature.name}</span>
        <button onClick={onRemove} className="p-1 text-text-secondary hover:text-
        red-500"><XMarkIcon /></button>
        <div className="p-1 text-text-secondary cursor-grab"><Bars3Icon /></div>
    </div>
);

export const LogicFlowBuilder: React.FC = () => {
    const [pipeline, setPipeline] = useState<PipelineNode[]>([]);
    const [generatedCode, setGeneratedCode] = useState('');
    const [isGenerating, setIsGenerating] = useState(false);
    const [isDraggingOver, setIsDraggingOver] = useState(false);

    const draggedItem = useRef<PipelineNode | null>(null);
    const dragOverItem = useRef<PipelineNode | null>(null);

    const handleGenerateCode = useCallback(async () => {
        setIsGenerating(true);
        setGeneratedCode('');

        const flowDescription = pipeline.map((node, index) => {
            const featureInfo = taxonomyMap.get(node.featureId);
            const functionName = featureToFunctionMap[node.featureId] || 'unknownTool';
            return `Step ${index + 1}: Execute the '${featureInfo?.name}' tool (function:
            ${functionName}). Description: ${featureInfo?.description}. Inputs:
            ${featureInfo?.inputs}.`;
        }).join('\n');


        try {
            const code = await generatePipelineCode(flowDescription);
            setGeneratedCode(code);
        } catch (e) {
            setGeneratedCode(`// Error generating code: ${e instanceof Error ? e.message :
```

```
        'Unknown error'}`);
    } finally {
        setIsGenerating(false);
    }
}, [pipeline]);

const handlePaletteDragStart = (e: React.DragEvent, featureId: string) => {
    e.dataTransfer.setData('application/json', JSON.stringify({ featureId }));
};

const handleDrop = (e: React.DragEvent) => {
    e.preventDefault();
    setIsDraggingOver(false);
    const data = e.dataTransfer.getData('application/json');
    if (!data) return; // Not a drop from the palette

    const { featureId } = JSON.parse(data);
    const newNode: PipelineNode = {
        id: Date.now(),
        featureId,
    };
    setPipeline(prev => [...prev, newNode]);
};

const handleNodeDragStart = (node: PipelineNode) => {
    draggedItem.current = node;
};

const handleNodeDragEnter = (node: PipelineNode) => {
    dragOverItem.current = node;
};

const handleNodeDragEnd = () => {
    if (draggedItem.current && dragOverItem.current && draggedItem.current.id !==
    dragOverItem.current.id) {
        const newPipeline = [...pipeline];
        const draggedIndex = pipeline.findIndex(p => p.id === draggedItem.current!.id);
        const targetIndex = pipeline.findIndex(p => p.id === dragOverItem.current!.id);

        const [removed] = newPipeline.splice(draggedIndex, 1);
        newPipeline.splice(targetIndex, 0, removed);
        setPipeline(newPipeline);
    }
    draggedItem.current = null;
    dragOverItem.current = null;
};

const handleRemoveNode = (id: number) => {
    setPipeline(prev => prev.filter(node => node.id !== id));
};

return (
    <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
        <header className="mb-6 flex justify-between items-start">
            <div>
                <h1 className="text-3xl font-bold flex items-center"><MapIcon /><span
                className="ml-3">Logic Flow Builder</span></h1>
                <p className="text-text-secondary mt-1">Visually build application logic flows
                and generate pipeline code.</p>
            </div>
            <button onClick={handleGenerateCode} disabled={isGenerating || pipeline.length
            === 0} className="btn-primary flex items-center gap-2 px-4 py-2">
```

```
        <SparklesIcon /> {isGenerating ? 'Generating...' : 'Generate Code'}
    </button>
</header>
<div className="flex-grow flex gap-6 min-h-0">
    <aside className="w-72 flex-shrink-0 bg-surface border border-border p-4
    rounded-lg flex flex-col">
        <h3 className="font-bold mb-3 text-lg">Features</h3>
        <div className="flex-grow overflow-y-auto space-y-3 pr-2">
            {ALL_FEATURES.map(feature => <FeaturePaletteItem key={feature.id}
            feature={feature} onDragStart={handlePaletteDragStart} />)}
        </div>
    </aside>
    <main
        className={`flex-grow relative bg-background border-2 ${isDraggingOver ?
        'border-primary' : 'border-dashed border-border'} rounded-lg overflow-y-auto p-4
        transition-colors`}
        onDrop={handleDrop}
        onDragOver={e => { e.preventDefault(); setIsDraggingOver(true); }}
        onDragLeave={() => setIsDraggingOver(false)}
    >
        {pipeline.length === 0 ? (
            <div className="w-full h-full flex items-center justify-center text-text-
            secondary">
                <p>Drag features here to build your pipeline</p>
            </div>
        ) : (
            <div className="space-y-2 max-w-lg mx-auto">
               {pipeline.map((node, index) => {
                    const feature = featuresMap.get(node.featureId);
                    if (!feature) return null;
                    return (
                        <div key={node.id} className="flex flex-col items-center">
                            <PipelineNodeComponent
                               node={node}
                               feature={feature}
                               index={index}
                               onDragStart={() => handleNodeDragStart(node)}
                               onDragEnter={() => handleNodeDragEnter(node)}
                               onDragEnd={handleNodeDragEnd}
                               onRemove={() => handleRemoveNode(node.id)}
                            />
                            {index < pipeline.length - 1 && (
                               <div className="w-0.5 h-6 bg-border my-1" />
                            )}
                        </div>
                    );
               })}
            </div>
        )}
    </main>
</div>
 {(isGenerating || generatedCode) && (
    <div className="fixed inset-0 bg-gray-900/80 backdrop-blur-sm z-50 flex items-
    center justify-center" onClick={() => setGeneratedCode('')}>
        <div className="w-full max-w-3xl h-3/4 bg-surface border border-border rounded-
        lg shadow-2xl p-6 flex flex-col" onClick={e => e.stopPropagation()}>
            <div className="flex justify-between items-center mb-4">
                <h2 className="text-xl font-bold">Generated Pipeline Code</h2>
                 <button onClick={() => setGeneratedCode('')} className="p-1 text-text-secondary
                 hover:text-text-primary"><XMarkIcon/></button>
            </div>
            <div className="flex-grow bg-background border border-border rounded-md
```

```
                        overflow-auto">
                            {isGenerating && !generatedCode ? <div className="flex justify-center items-
                            center h-full"><LoadingSpinner /></div> : <MarkdownRenderer
                            content={'```javascript\n' + generatedCode.replace(/^```(javascript)?|```$/g,
                            '') + '\n```'} />}
                        </div>
                    </div>
                </div>
            )}
        </div>
    );
};


// ===== AiCommandCenter_64.tsx =====

import React, { useState, useCallback } from 'react';
import { Type, FunctionDeclaration } from "@google/genai";
import { getInferenceFunction, CommandResponse, FEATURE_TAXONOMY, logError }
from '../../services/index.ts';
import { useGlobalState } from '../../contexts/GlobalStateContext.tsx';
import { CommandLineIcon } from '../icons.tsx';
import { LoadingSpinner } from '../shared/index.tsx';
import { RAW_FEATURES } from '../../constants.ts';

const functionDeclarations: FunctionDeclaration[] = [
    {
        name: 'navigateTo',
        description: 'Navigates to a specific feature page.',
        parameters: {
            type: Type.OBJECT,
            properties: {
                featureId: {
                    type: Type.STRING,
                    description: 'The ID of the feature to navigate to.',
                    enum: RAW_FEATURES.map(f => f.id)
                },
            },
            required: ['featureId'],
        },
    },
    {
        name: 'runFeatureWithInput',
        description: 'Navigates to a feature and passes initial data to it.',
        parameters: {
            type: Type.OBJECT,
            properties: {
                 featureId: {
                    type: Type.STRING,
                    description: 'The ID of the feature to run.',
                    enum: RAW_FEATURES.map(f => f.id)
                },
                props: {
                    type: Type.OBJECT,
                    description: 'An object containing the initial properties for the feature, based
                    on its required inputs.',
                    properties: {
                        initialCode: { type: Type.STRING },
                        initialPrompt: { type: Type.STRING },
                        beforeCode: { type: Type.STRING },
                        afterCode: { type: Type.STRING },
                        logInput: { type: Type.STRING },
```

```
                        diff: { type: Type.STRING },
                        codeInput: { type: Type.STRING },
                        jsonInput: { type: Type.STRING },
                    }
                }
            },
            required: ['featureId', 'props']
        }
    }
];

const knowledgeBase = FEATURE_TAXONOMY.map(f => `- ${f.name} (${f.id}):
${f.description} Inputs: ${f.inputs}`).join('\n');

const ExamplePromptButton: React.FC<{ text: string, onClick: (text: string) =>
void }> = ({ text, onClick }) => (
    <button
        onClick={() => onClick(text)}
        className="px-3 py-1.5 bg-surface border border-border rounded-full text-xs
        hover:bg-gray-100 transition-colors"
    >
        {text}
    </button>
)

export const AiCommandCenter: React.FC = () => {
    const { dispatch } = useGlobalState();
    const [prompt, setPrompt] = useState('');
    const [isLoading, setIsLoading] = useState(false);
    const [lastResponse, setLastResponse] = useState('');

    const handleCommand = useCallback(async () => {
        if (!prompt.trim()) return;

        setIsLoading(true);
        setLastResponse('');

        try {
            const response: CommandResponse = await getInferenceFunction(prompt,
            functionDeclarations, knowledgeBase);

            if (response.functionCalls && response.functionCalls.length > 0) {
                const call = response.functionCalls[0];
                const { name, args } = call;

                setLastResponse(`Understood! Executing command: ${name}`);

                switch (name) {
                    case 'navigateTo':
                        dispatch({ type: 'SET_VIEW', payload: { view: args.featureId }});
                        break;
                    case 'runFeatureWithInput':
                         dispatch({ type: 'SET_VIEW', payload: { view: args.featureId, props: args.props
                         } });
                        break;
                    default:
                        setLastResponse(`Unknown command: ${name}`);
                }
                 setPrompt('');
            } else {
                 setLastResponse(response.text);
            }
```

```
    } catch (err) {
        logError(err as Error, { prompt });
        setLastResponse(err instanceof Error ? err.message : 'An unknown error
        occurred.');
    } finally {
        setIsLoading(false);
    }
}, [prompt, dispatch]);

const handleKeyDown = (e: React.KeyboardEvent) => {
    if (e.key === 'Enter' && !e.shiftKey) {
        e.preventDefault();
        handleCommand();
    }
};

const handleExampleClick = (text: string) => {
    setPrompt(text);
}

return (
    <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
        <header className="mb-6 text-center">
            <h1 className="text-4xl font-extrabold tracking-tight flex items-center justify-
            center">
                <CommandLineIcon />
                <span className="ml-3">AI Command Center</span>
            </h1>
            <p className="mt-2 text-lg text-text-secondary">What would you like to do?</p>
        </header>

        <div className="flex-grow flex flex-col justify-end max-w-3xl w-full mx-auto">
            {lastResponse && (
                <div className="mb-4 p-4 bg-surface rounded-lg text-text-primary border border-
                border">
                    <p><strong>AI:</strong> {lastResponse}</p>
                </div>
            )}
             <div className="relative">
                <textarea
                    value={prompt}
                    onChange={e => setPrompt(e.target.value)}
                    onKeyDown={handleKeyDown}
                    disabled={isLoading}
                    placeholder='Try "explain this code: const a = 1;" or "open the theme designer"'
                    className="w-full p-4 pr-28 rounded-lg bg-surface border border-border
                    focus:ring-2 focus:ring-primary focus:outline-none resize-none shadow-sm"
                    rows={2}
                />
                <button
                    onClick={handleCommand}
                    disabled={isLoading}
                    className="btn-primary absolute right-3 top-1/2 -translate-y-1/2 px-4 py-2"
                >
                   {isLoading ? <LoadingSpinner/> : 'Send'}
                </button>
            </div>
             <div className="flex flex-wrap items-center justify-center gap-2 mt-4">
                <ExamplePromptButton text="Open Theme Designer" onClick={handleExampleClick} />
                <ExamplePromptButton text="Generate a commit for a bug fix"
                onClick={handleExampleClick} />
                <ExamplePromptButton text="Create a regex for email validation"
```

```
                    onClick={handleExampleClick} />
                </div>
                 <p className="text-xs text-text-secondary text-center mt-2">Press Enter to send,
                 Shift+Enter for new line.</p>
            </div>
        </div>
    );
};

// ===== LogicFlowBuilder_66.tsx =====

import React, { useState, useCallback, useRef } from 'react';
import { ALL_FEATURES } from './index.ts';
// FIX: Import FEATURE_TAXONOMY from services/index.ts where it is now exported.
import { FEATURE_TAXONOMY, generatePipelineCode } from
'../../services/index.ts';
import { featureToFunctionMap } from '../../services/pipelineTools.ts';
import type { Feature } from '../../types.ts';
import { MapIcon, SparklesIcon, XMarkIcon, Bars3Icon } from '../icons.tsx';
import { LoadingSpinner, MarkdownRenderer } from '../shared/index.tsx';

interface PipelineNode {
    id: number;
    featureId: string;
}

const featuresMap = new Map(ALL_FEATURES.map(f => [f.id, f]));
const taxonomyMap = new Map(FEATURE_TAXONOMY.map(f => [f.id, f]));

const FeaturePaletteItem: React.FC<{ feature: Feature, onDragStart: (e:
React.DragEvent, featureId: string) => void }> = ({ feature, onDragStart }) => (
    <div
        draggable
        onDragStart={e => onDragStart(e, feature.id)}
        className="p-3 rounded-md bg-gray-50 border border-border flex items-center
        gap-3 cursor-grab hover:bg-gray-100 transition-colors"
    >
        <div className="text-primary flex-shrink-0">{feature.icon}</div>
        <div>
            <h4 className="font-bold text-sm text-text-primary">{feature.name}</h4>
            <p className="text-xs text-text-secondary">{feature.category}</p>
        </div>
    </div>
);

const PipelineNodeComponent: React.FC<{
    node: PipelineNode;
    feature: Feature;
    index: number;
    onDragStart: () => void;
    onDragEnter: () => void;
    onDragEnd: () => void;
    onRemove: () => void;
}> = ({ node, feature, index, onDragStart, onDragEnter, onDragEnd, onRemove })
=> (
    <div
        draggable
        onDragStart={onDragStart}
        onDragEnter={onDragEnter}
        onDragEnd={onDragEnd}
        onDragOver={(e) => e.preventDefault()}
        className="w-full bg-surface rounded-lg shadow-md border-2 border-border cursor-
```

```
        grab active:cursor-grabbing flex items-center p-3 gap-4"
    >
        <div className="flex-shrink-0 w-8 h-8 rounded-full bg-primary text-white flex
        items-center justify-center font-bold text-lg">{index + 1}</div>
        <div className="w-6 h-6 text-text-secondary">{feature.icon}</div>
        <span className="flex-grow font-semibold truncate text-text-
        primary">{feature.name}</span>
        <button onClick={onRemove} className="p-1 text-text-secondary hover:text-
        red-500"><XMarkIcon /></button>
        <div className="p-1 text-text-secondary cursor-grab"><Bars3Icon /></div>
    </div>
);

export const LogicFlowBuilder: React.FC = () => {
    const [pipeline, setPipeline] = useState<PipelineNode[]>([]);
    const [generatedCode, setGeneratedCode] = useState('');
    const [isGenerating, setIsGenerating] = useState(false);
    const [isDraggingOver, setIsDraggingOver] = useState(false);

    const draggedItem = useRef<PipelineNode | null>(null);
    const dragOverItem = useRef<PipelineNode | null>(null);

    const handleGenerateCode = useCallback(async () => {
        setIsGenerating(true);
        setGeneratedCode('');

        const flowDescription = pipeline.map((node, index) => {
            const featureInfo = taxonomyMap.get(node.featureId);
            if (!featureInfo) {
                return `Step ${index + 1}: Unknown tool with ID ${node.featureId}`;
            }
            const functionName = featureToFunctionMap[node.featureId] || 'unknownTool';
            return `Step ${index + 1}: Execute the '${featureInfo.name}' tool (function:
            ${functionName}). Description: ${featureInfo.description}. Inputs:
            ${featureInfo.inputs}.`;
        }).join('\n');


        try {
            const code = await generatePipelineCode(flowDescription);
            setGeneratedCode(code);
        } catch (e) {
            setGeneratedCode(`// Error generating code: ${e instanceof Error ? e.message :
            'Unknown error'}`);
        } finally {
            setIsGenerating(false);
        }
    }, [pipeline]);

    const handlePaletteDragStart = (e: React.DragEvent, featureId: string) => {
        e.dataTransfer.setData('application/json', JSON.stringify({ featureId }));
    };

    const handleDrop = (e: React.DragEvent) => {
        e.preventDefault();
        setIsDraggingOver(false);
        const data = e.dataTransfer.getData('application/json');
        if (!data) return; // Not a drop from the palette

        const { featureId } = JSON.parse(data);
        const newNode: PipelineNode = {
            id: Date.now(),
```

```
        featureId,
    };
    setPipeline(prev => [...prev, newNode]);
};

const handleNodeDragStart = (node: PipelineNode) => {
    draggedItem.current = node;
};

const handleNodeDragEnter = (node: PipelineNode) => {
    dragOverItem.current = node;
};

const handleNodeDragEnd = () => {
    if (draggedItem.current && dragOverItem.current && draggedItem.current.id !==
    dragOverItem.current.id) {
        const newPipeline = [...pipeline];
        const draggedIndex = pipeline.findIndex(p => p.id === draggedItem.current!.id);
        const targetIndex = pipeline.findIndex(p => p.id === dragOverItem.current!.id);

        const [removed] = newPipeline.splice(draggedIndex, 1);
        newPipeline.splice(targetIndex, 0, removed);
        setPipeline(newPipeline);
    }
    draggedItem.current = null;
    dragOverItem.current = null;
};

const handleRemoveNode = (id: number) => {
    setPipeline(prev => prev.filter(node => node.id !== id));
};

return (
    <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
        <header className="mb-6 flex justify-between items-start">
            <div>
                <h1 className="text-3xl font-bold flex items-center"><MapIcon /><span
                className="ml-3">Logic Flow Builder</span></h1>
                <p className="text-text-secondary mt-1">Visually build application logic flows
                and generate pipeline code.</p>
            </div>
            <button onClick={handleGenerateCode} disabled={isGenerating || pipeline.length
            === 0} className="btn-primary flex items-center gap-2 px-4 py-2">
                <SparklesIcon /> {isGenerating ? 'Generating...' : 'Generate Code'}
            </button>
        </header>
        <div className="flex-grow flex gap-6 min-h-0">
            <aside className="w-72 flex-shrink-0 bg-surface border border-border p-4
            rounded-lg flex flex-col">
                <h3 className="font-bold mb-3 text-lg">Features</h3>
                <div className="flex-grow overflow-y-auto space-y-3 pr-2">
                    {ALL_FEATURES.map(feature => <FeaturePaletteItem key={feature.id}
                    feature={feature} onDragStart={handlePaletteDragStart} />)}
                </div>
            </aside>
            <main
                className={`flex-grow relative bg-background border-2 ${isDraggingOver ?
                'border-primary' : 'border-dashed border-border'} rounded-lg overflow-y-auto p-4
                transition-colors`}
                onDrop={handleDrop}
                onDragOver={e => { e.preventDefault(); setIsDraggingOver(true); }}
                onDragLeave={() => setIsDraggingOver(false)}
```

```
                >
                    {pipeline.length === 0 ? (
                        <div className="w-full h-full flex items-center justify-center text-text-
                        secondary">
                            <p>Drag features here to build your pipeline</p>
                        </div>
                    ) : (
                        <div className="space-y-2 max-w-lg mx-auto">
                           {pipeline.map((node, index) => {
                                const feature = featuresMap.get(node.featureId);
                                if (!feature) return null;
                                return (
                                    <div key={node.id} className="flex flex-col items-center">
                                        <PipelineNodeComponent
                                           node={node}
                                           feature={feature}
                                           index={index}
                                           onDragStart={() => handleNodeDragStart(node)}
                                           onDragEnter={() => handleNodeDragEnter(node)}
                                           onDragEnd={handleNodeDragEnd}
                                           onRemove={() => handleRemoveNode(node.id)}
                                        />
                                        {index < pipeline.length - 1 && (
                                            <div className="w-0.5 h-6 bg-border my-1" />
                                        )}
                                    </div>
                                );
                           })}
                        </div>
                    )}
                </main>
            </div>
             {(isGenerating || generatedCode) && (
                <div className="fixed inset-0 bg-gray-900/80 backdrop-blur-sm z-50 flex items-
                center justify-center" onClick={() => setGeneratedCode('')}>
                    <div className="w-full max-w-3xl h-3/4 bg-surface border border-border rounded-
                    lg shadow-2xl p-6 flex flex-col" onClick={e => e.stopPropagation()}>
                        <div className="flex justify-between items-center mb-4">
                            <h2 className="text-xl font-bold">Generated Pipeline Code</h2>
                             <button onClick={() => setGeneratedCode('')} className="p-1 text-text-secondary
                             hover:text-text-primary"><XMarkIcon/></button>
                        </div>
                        <div className="flex-grow bg-background border border-border rounded-md
                        overflow-auto">
                            {isGenerating && !generatedCode ? <div className="flex justify-center items-
                            center h-full"><LoadingSpinner /></div> : <MarkdownRenderer
                            content={'```javascript\n' + generatedCode.replace(/^```(javascript)?|```$/g,
                            '') + '\n```'} />}
                        </div>
                    </div>
                </div>
            )}
        </div>
    );
};

// ===== ProjectExplorer_64.tsx =====



import React, { useState, useEffect } from 'react';
import { useGlobalState } from '../../contexts/GlobalStateContext.tsx';
```

```
import { getRepos, getRepoTree, getFileContent } from '../../services/index.ts';
import type { Repo, FileNode } from '../../types.ts';
import { FolderIcon, DocumentIcon } from '../icons.tsx';
import { LoadingSpinner } from '../shared/index.tsx';

const FileTree: React.FC<{ node: FileNode, onFileSelect: (path: string) => void
}> = ({ node, onFileSelect }) => {
    const [isOpen, setIsOpen] = useState(true);

    // FIX: Use isDirectory boolean property instead of non-existent type property.
    if (!node.isDirectory) {
        return (
            <div
                className="flex items-center space-x-2 pl-4 py-1 cursor-pointer hover:bg-
                gray-100 rounded"
                onClick={() => onFileSelect(node.path)}
            >
                <DocumentIcon />
                <span>{node.name}</span>
            </div>
        );
    }

    return (
        <div>
            <div
                className="flex items-center space-x-2 py-1 cursor-pointer hover:bg-gray-100
                rounded"
                onClick={() => setIsOpen(!isOpen)}
            >
                <div className={`transform transition-transform ${isOpen ? 'rotate-90' :
                ''}`}>■</div>
                <FolderIcon />
                <span className="font-semibold">{node.name}</span>
            </div>
            {/* FIX: Check for children array before attempting to map over it. */}
            {isOpen && node.children && (
                <div className="pl-4 border-l border-border ml-3">
                    {node.children.map(child => <FileTree key={child.path} node={child}
                    onFileSelect={onFileSelect} />)}
                </div>
            )}
        </div>
    );
};

export const ProjectExplorer: React.FC = () => {
    const { state, dispatch } = useGlobalState();
    const { githubToken, selectedRepo, projectFiles } = state;
    const [repos, setRepos] = useState<Repo[]>([]);
    const [isLoading, setIsLoading] = useState<'repos' | 'tree' | null>(null);
    const [error, setError] = useState('');
    const [activeFileContent, setActiveFileContent] = useState<string | null>(null);

    useEffect(() => {
        if (githubToken) {
            setIsLoading('repos');
            setError('');
            getRepos()
                .then(setRepos)
                .catch(err => setError(err.message))
                .finally(() => setIsLoading(null));
```

```
    } else {
        setRepos([]);
    }
}, [githubToken]);

useEffect(() => {
    if (selectedRepo && githubToken) {
        setIsLoading('tree');
        setError('');
        setActiveFileContent(null);
        getRepoTree(selectedRepo.owner, selectedRepo.repo)
            .then(tree => dispatch({ type: 'LOAD_PROJECT_FILES', payload: tree }))
            .catch(err => setError(err.message))
            .finally(() => setIsLoading(null));
    }
}, [selectedRepo, githubToken, dispatch]);

const handleFileSelect = async (path: string) => {
    if (!selectedRepo) return;
    try {
        const content = await getFileContent(selectedRepo.owner, selectedRepo.repo,
        path);
        setActiveFileContent(content);
    } catch (err) {
        setError((err as Error).message);
    }
};

if (!githubToken) {
    return (
        <div className="h-full flex flex-col items-center justify-center text-center
        text-text-secondary p-4">
            <FolderIcon />
            <h2 className="text-lg font-semibold mt-2">Connect to GitHub</h2>
            <p>Please go to the "Connections" tab and provide a Personal Access Token to
            explore your repositories.</p>
        </div>
    );
}

return (
    <div className="h-full flex flex-col text-text-primary">
        <header className="p-4 border-b border-border flex-shrink-0">
            <h1 className="text-xl font-bold flex items-center"><FolderIcon /><span
            className="ml-3">Project Explorer</span></h1>
            <div className="mt-2">
                <select
                    value={selectedRepo ? `${selectedRepo.owner}/${selectedRepo.repo}` : ''}
                    onChange={e => {
                        const [owner, repo] = e.target.value.split('/');
                        dispatch({ type: 'SET_SELECTED_REPO', payload: { owner, repo } });
                    }}
                    className="w-full p-2 bg-surface border border-border rounded-md text-sm"
                >
                    <option value="" disabled>{isLoading === 'repos' ? 'Loading...' : 'Select a
                    repository'}</option>
                    {repos.map(r => <option key={r.id} value={r.full_name}>{r.full_name}</option>)}
                </select>
            </div>
            {error && <p className="text-red-500 text-xs mt-2">{error}</p>}
        </header>
        <div className="flex-grow flex min-h-0">
```

```
                <aside className="w-1/3 bg-background border-r border-border p-4 overflow-y-
                auto">
                    {isLoading === 'tree' && <div className="flex justify-center"><LoadingSpinner
                    /></div>}
                    {projectFiles && <FileTree node={projectFiles} onFileSelect={handleFileSelect}
                    />}
                </aside>
                <main className="flex-1 bg-surface">
                    <pre className="w-full h-full p-4 text-sm overflow-auto whitespace-pre-wrap">
                        <code>{activeFileContent ?? 'Select a file to view its content.'}</code>
                    </pre>
                </main>
            </div>
        </div>
    );
};


// ===== RegexSandbox_66.tsx =====



import React, { useState, useMemo, useCallback, useEffect } from 'react';
// FIX: Import from index.ts where all services are exported
import { generateRegExStream } from '../../services/index.ts';
import { BeakerIcon } from '../icons.tsx';
import { LoadingSpinner } from '../shared/index.tsx';

const commonPatterns = [
    { name: 'Email', pattern: '/[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\\.[a-zA-Z]{2,}/g'
    },
    { name: 'URL', pattern: '/https?:\\/\\/(www\\.)?[-a-zA-Z0-
    9@:%._\\+~#=]{1,256}\\.[a-zA-Z0-9()]{1,6}\\b([-a-zA-Z0-9()@:%_\\+.~#?&//=]*)/g'
    },
    { name: 'IPv4 Address', pattern:
    '/((25[0-5]|(2[0-4]|1\\d|[1-9]|)\\d)\\.?\\b){4}/g' },
    { name: 'Date (YYYY-MM-DD)', pattern: '/\\d{4}-\\d{2}-\\d{2}/g' },
];

const CheatSheet = () => (
    <div className="bg-surface border border-border p-4 rounded-lg">
        <h3 className="text-lg font-bold mb-2">Regex Cheat Sheet</h3>
        <div className="grid grid-cols-2 gap-x-4 gap-y-1 text-xs font-mono">
            <p><span className="text-primary">.</span> - Any character</p>
            <p><span className="text-primary">\d</span> - Any digit</p>
            <p><span className="text-primary">\w</span> - Word character</p>
            <p><span className="text-primary">\s</span> - Whitespace</p>
            <p><span className="text-primary">[abc]</span> - a, b, or c</p>
            <p><span className="text-primary">[^abc]</span> - Not a, b, or c</p>
            <p><span className="text-primary">*</span> - 0 or more</p>
            <p><span className="text-primary">+</span> - 1 or more</p>
            <p><span className="text-primary">?</span> - 0 or one</p>
            <p><span className="text-primary">^</span> - Start of string</p>
            <p><span className="text-primary">$</span> - End of string</p>
            <p><span className="text-primary">\b</span> - Word boundary</p>
        </div>
    </div>
);

export const RegexSandbox: React.FC<{ initialPrompt?: string }> = ({
initialPrompt }) => {
    const [pattern, setPattern] =
```

```
useState<string>('/\\b([A-Z][a-z]+)\\s(\\w+)\\b/g');
const [testString, setTestString] = useState<string>('The quick Brown Fox jumps
over the Lazy Dog.');
const [aiPrompt, setAiPrompt] = useState<string>(initialPrompt || 'find
capitalized words and the word after');
const [isAiLoading, setIsAiLoading] = useState<boolean>(false);

const { matches, error } = useMemo(() => {
    try {
        const patternParts = pattern.match(/^\/(.*)\/([gimyus]*)$/);
        if (!patternParts) return { matches: null, error: 'Invalid regex literal. Use
        /pattern/flags.' };
        const [, regexBody, regexFlags] = patternParts;
        const regex = new RegExp(regexBody, regexFlags);
        return { matches: [...testString.matchAll(regex)], error: null };
    } catch (e) { return { matches: null, error: e instanceof Error ? e.message :
    'Unknown error.' }; }
}, [pattern, testString]);

const handleGenerateRegex = useCallback(async (p: string) => {
    if (!p) return;
    setIsAiLoading(true);
    try {
        const stream = generateRegExStream(p);
        let fullResponse = '';
        for await (const chunk of stream) { fullResponse += chunk; }
        setPattern(fullResponse.trim().replace(/^`+|`+$/g, ''));
    } finally { setIsAiLoading(false); }
}, []);

useEffect(() => { if (initialPrompt) handleGenerateRegex(initialPrompt); },
[initialPrompt, handleGenerateRegex]);

const highlightedString = useMemo(() => {
    if (!matches || matches.length === 0 || error) return testString;
    let lastIndex = 0;
    const parts: (string | JSX.Element)[] = [];
    matches.forEach((match, i) => {
        if (match.index === undefined) return;
        parts.push(testString.substring(lastIndex, match.index));
        parts.push(<mark key={i} className="bg-primary/20 text-primary rounded
        px-1">{match[0]}</mark>);
        lastIndex = match.index + match[0].length;
    });
    parts.push(testString.substring(lastIndex));
    return parts;
}, [matches, testString, error]);

return (
    <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
        <header className="mb-6"><h1 className="text-3xl font-bold flex items-
        center"><BeakerIcon /><span className="ml-3">RegEx Sandbox</span></h1><p
        className="text-text-secondary mt-1">Test your regular expressions and generate
        them with AI.</p></header>
        <div className="flex-grow grid grid-cols-1 lg:grid-cols-3 gap-6 min-h-0">
            <div className="lg:col-span-2 flex flex-col gap-4">
                <div className="flex gap-2"><input type="text" value={aiPrompt} onChange={(e) =>
                setAiPrompt(e.target.value)} placeholder="Describe the pattern to find..."
                className="flex-grow px-3 py-1.5 rounded-md bg-surface border border-border
                text-sm focus:ring-2 focus:ring-primary" /><button onClick={() =>
                handleGenerateRegex(aiPrompt)} disabled={isAiLoading} className="btn-primary
                px-4 py-1.5 flex items-center">{isAiLoading ? <LoadingSpinner/> :
```

```
                    'Generate'}</button></div>
                    <div><label htmlFor="regex-pattern" className="text-sm font-medium text-text-
                    secondary">Regular Expression</label><input id="regex-pattern" type="text"
                    value={pattern} onChange={(e) => setPattern(e.target.value)} className={`w-full
                    mt-1 px-3 py-2 rounded-md bg-surface border ${error ? 'border-red-500' :
                    'border-border'} font-mono text-sm focus:ring-2 focus:ring-primary`} />{error &&
                    <p className="text-red-500 text-xs mt-1">{error}</p>}</div>
                    <div className="flex flex-col flex-grow min-h-0"><label htmlFor="test-string"
                    className="text-sm font-medium text-text-secondary">Test String</label><textarea
                    id="test-string" value={testString} onChange={(e) =>
                    setTestString(e.target.value)} className="w-full mt-1 p-3 rounded-md bg-surface
                    border border-border font-mono text-sm resize-y h-32" /><div className="mt-2 p-3
                    bg-background rounded-md border border-border min-h-[50px] whitespace-pre-
                    wrap">{highlightedString}</div></div>
                    <div className="flex-shrink-0"><h3 className="text-lg font-bold">Match Groups
                    ({matches?.length || 0})</h3><div className="mt-2 p-2 bg-surface rounded-md
                    overflow-y-auto max-h-48 font-mono text-xs border border-border">{matches &&
                    matches.length > 0 ? (matches.map((match, i) => (<details key={i} className="p-2
                    border-b border-border"><summary className="cursor-pointer text-green-700">Match
                    {i + 1}: "{match[0]}"</summary><div className="pl-4
                    mt-1">{Array.from(match).map((group, gIndex) => <p key={gIndex} className="text-
                    text-secondary">Group {gIndex}: <span className="text-
                    amber-700">{String(group)}</span></p>)}</div></details>))) : (<p
                    className="text-text-secondary text-sm p-2">No matches found.</p>)}</div></div>
                </div>
                <div className="lg:col-span-1 space-y-4">
                    <CheatSheet />
                    <div className="bg-surface border border-border p-4 rounded-lg">
                        <h3 className="text-lg font-bold mb-2">Common Patterns</h3>
                        <div className="flex flex-col items-start gap-2">
                            {commonPatterns.map(p => (
                                <button key={p.name} onClick={() => setPattern(p.pattern)} className="text-left
                                text-sm text-primary hover:underline">
                                    {p.name}
                                </button>
                            ))}
                        </div>
                    </div>
                </div>
            </div>
        </div>
    );
};


// ===== ScreenshotToComponent_66.tsx =====



import React, { useState, useCallback, useRef } from 'react';
// FIX: Import from index.ts where all services are exported
import { generateComponentFromImageStream } from '../../services/index.ts';
import { PhotoIcon, ArrowDownTrayIcon } from '../icons.tsx';
import { LoadingSpinner, MarkdownRenderer } from '../shared/index.tsx';
import { fileToBase64, blobToDataURL, downloadFile } from
'../../services/index.ts';

export const ScreenshotToComponent: React.FC = () => {
    const [previewImage, setPreviewImage] = useState<string | null>(null);
    const [rawCode, setRawCode] = useState('');
    const [isLoading, setIsLoading] = useState(false);
    const [error, setError] = useState('');
```

```
const fileInputRef = useRef<HTMLInputElement>(null);

const handleGenerate = async (base64Image: string) => {
    setIsLoading(true);
    setError('');
    setRawCode('');
    try {
        const stream = generateComponentFromImageStream(base64Image);
        let fullResponse = '';
        for await (const chunk of stream) {
            fullResponse += chunk;
            setRawCode(fullResponse.replace(/^```(?:\w+\n)?/, '').replace(/```$/, ''));
        }
    } catch (err) {
        setError(err instanceof Error ? err.message : 'An unknown error occurred.');
    } finally {
        setIsLoading(false);
    }
};

const processImageBlob = async (blob: Blob) => {
    try {
        const [dataUrl, base64Image] = await Promise.all([blobToDataURL(blob),
        fileToBase64(blob as File)]);
        setPreviewImage(dataUrl);
        handleGenerate(base64Image);
    } catch (e) {
        setError('Could not process the image.');
    }
};

const handlePaste = useCallback(async (event: React.ClipboardEvent) => {
    const items = event.clipboardData.items;
    for (const item of items) {
        if (item.type.indexOf('image') !== -1) {
            const blob = item.getAsFile();
            if (blob) await processImageBlob(blob);
            return;
        }
    }
}, []);

const handleFileChange = async (event: React.ChangeEvent<HTMLInputElement>) => {
    const file = event.target.files?.[0];
    if (file) await processImageBlob(file);
};

return (
    <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
        <header className="mb-6"><h1 className="text-3xl font-bold flex items-
        center"><PhotoIcon /><span className="ml-3">AI Screenshot-to-
        Component</span></h1><p className="text-text-secondary mt-1">Paste or upload a
        screenshot of a UI element to generate React/Tailwind code.</p></header>
        <div className="flex-grow grid grid-cols-1 lg:grid-cols-2 gap-6 min-h-0">
            <div onPaste={handlePaste} className="flex flex-col items-center justify-center
            bg-surface p-6 rounded-lg border-2 border-dashed border-border focus:outline-
            none focus:border-primary overflow-y-auto" tabIndex={0}>
                {previewImage ? (<img src={previewImage} alt="Pasted content" className="max-w-
                full max-h-full object-contain rounded-md shadow-lg" />) : (<div
                className="text-center text-text-secondary">
                        <h2 className="text-xl font-bold text-text-primary">Paste an image here</h2>
                        <p className="mb-2">(Cmd/Ctrl + V)</p>
```

```
                            <p className="text-sm">or</p>
                            <button onClick={() => fileInputRef.current?.click()} className="mt-2 btn-
                            primary px-4 py-2 text-sm">Upload File</button>
                            <input type="file" ref={fileInputRef} onChange={handleFileChange}
                            accept="image/*" className="hidden"/>
                        </div>)}
                </div>
                <div className="flex flex-col h-full">
                    <div className="flex justify-between items-center mb-2">
                        <label className="text-sm font-medium text-text-secondary">Generated
                        Code</label>
                        {rawCode && !isLoading && (
                            <div className="flex items-center gap-2">
                                <button onClick={() => navigator.clipboard.writeText(rawCode)} className="px-3
                                py-1 bg-gray-100 text-xs rounded-md hover:bg-gray-200">Copy Code</button>
                                <button onClick={() => downloadFile(rawCode, 'Component.tsx',
                                'text/typescript')} className="flex items-center gap-1 px-3 py-1 bg-gray-100
                                text-xs rounded-md hover:bg-gray-200">
                                    <ArrowDownTrayIcon className="w-4 h-4" /> Download
                                </button>
                            </div>
                        )}
                    </div>
                    <div className="flex-grow bg-background border border-border rounded-md
                    overflow-y-auto">
                        {isLoading && (<div className="flex items-center justify-center
                        h-full"><LoadingSpinner /></div>)}
                        {error && <p className="p-4 text-red-500">{error}</p>}
                        {rawCode && !isLoading && <MarkdownRenderer
                        content={`\`\`\`tsx\n${rawCode}\n\`\`\`} />}
                        {!isLoading && !rawCode && !error && (<div className="text-text-secondary h-full
                        flex items-center justify-center">Generated component code will appear
                        here.</div>)}
                    </div>
                </div>
            </div>
        </div>
    );
};


// ===== SnippetVault_66.tsx =====



import React, { useState, useEffect, useMemo } from 'react';
import { LockClosedIcon, SparklesIcon, TrashIcon, ClipboardDocumentIcon,
ArrowDownTrayIcon } from '../icons.tsx';
import { useLocalStorage } from '../../hooks/useLocalStorage.ts';
// FIX: Import from index.ts where all services are exported
import { enhanceSnippetStream } from '../../services/index.ts';
import { LoadingSpinner } from '../shared/index.tsx';
import { downloadFile } from '../../services/index.ts';

interface Snippet {
    id: number; name: string; code: string; language: string; tags: string[];
}

const langToExt: Record<string, string> = {
    javascript: 'js',
    typescript: 'ts',
    python: 'py',
```

```
    css: 'css',
    html: 'html',
    json: 'json',
    markdown: 'md',
    plaintext: 'txt',
};

export const SnippetVault: React.FC = () => {
    const [snippets, setSnippets] = useLocalStorage<Snippet[]>('devcore_snippets',
    [{ id: 1, name: 'React Hook Boilerplate', language: 'javascript', code: `import
    { useState } from 'react';\n\nconst useCustomHook = () => {\n  const [value,
    setValue] = useState(null);\n  return { value, setValue };\n};`, tags: ['react',
    'hook'] }]);
    const [activeSnippet, setActiveSnippet] = useState<Snippet | null>(null);
    const [isEnhancing, setIsEnhancing] = useState(false);
    const [searchTerm, setSearchTerm] = useState('');
    const [isEditingName, setIsEditingName] = useState(false);

    const filteredSnippets = useMemo(() => {
        if (!searchTerm) return snippets;
        const lowerSearch = searchTerm.toLowerCase();
        return snippets.filter((s: Snippet) =>
            s.name.toLowerCase().includes(lowerSearch) ||
            s.code.toLowerCase().includes(lowerSearch) ||
            (s.tags && s.tags.some(t => t.toLowerCase().includes(lowerSearch)))
        );
    }, [snippets, searchTerm]);

    useEffect(() => {
        if (!activeSnippet && filteredSnippets.length > 0)
        setActiveSnippet(filteredSnippets[0]);
        if (activeSnippet) setActiveSnippet(snippets.find((s: Snippet) => s.id ===
        activeSnippet.id) || null);
    }, [snippets, activeSnippet, filteredSnippets]);

    const updateSnippet = (snippet: Snippet) => {
        setSnippets(snippets.map((s: Snippet) => s.id === snippet.id ? snippet : s));
        setActiveSnippet(snippet);
    };

    const handleEnhance = async () => {
        if (!activeSnippet) return;
        setIsEnhancing(true);
        try {
            const stream = enhanceSnippetStream(activeSnippet.code);
            let fullResponse = '';
            for await (const chunk of stream) {
                fullResponse += chunk;
                updateSnippet({ ...activeSnippet, code: fullResponse.replace(/^```(?:\w+\n)?/,
                '').replace(/```$/, '') });
            }
        } finally { setIsEnhancing(false); }
    };

    const handleAddNew = () => {
        const newSnippet: Snippet = { id: Date.now(), name: 'New Snippet', language:
        'plaintext', code: '', tags: [] };
        setSnippets([...snippets, newSnippet]);
        setActiveSnippet(newSnippet);
    };

    const handleDelete = (id: number) => {
```

```
    setSnippets(snippets.filter((s: Snippet) => s.id !== id));
    if(activeSnippet?.id === id) setActiveSnippet(filteredSnippets.length > 1 ?
    filteredSnippets[0] : null);
};

const handleDownload = () => {
    if(!activeSnippet) return;
    const extension = langToExt[activeSnippet.language] || 'txt';
    const filename = `${activeSnippet.name.replace(/\s/g, '_')}.${extension}`;
    downloadFile(activeSnippet.code, filename);
}

const handleNameChange = (e: React.ChangeEvent<HTMLInputElement>) => {
    if (activeSnippet) updateSnippet({...activeSnippet, name: e.target.value});
};

const handleTagsChange = (e: React.KeyboardEvent<HTMLInputElement>) => {
    if (e.key === 'Enter' && activeSnippet) {
        const newTag = e.currentTarget.value.trim();
        if (newTag && !activeSnippet.tags.includes(newTag)) {
            updateSnippet({...activeSnippet, tags: [...(activeSnippet.tags ?? []),
            newTag]});
        }
        e.currentTarget.value = '';
    }
};

return (
    <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
        <header className="mb-6"><h1 className="text-3xl font-bold flex items-
        center"><LockClosedIcon /><span className="ml-3">Snippet Vault</span></h1><p
        className="text-text-secondary mt-1">Store, search, tag, and enhance your
        reusable code snippets with AI.</p></header>
        <div className="flex-grow flex gap-6 min-h-0">
            <aside className="w-1/3 bg-surface border border-border p-4 rounded-lg flex
            flex-col">
                <input type="text" placeholder="Search snippets..." value={searchTerm}
                onChange={e => setSearchTerm(e.target.value)} className="w-full px-3 py-1.5 mb-3
                rounded-md bg-background border border-border text-sm"/>
                <ul className="space-y-2 flex-grow overflow-y-auto
                pr-2">{filteredSnippets.map((s: Snippet) => (<li key={s.id} className="group
                flex items-center justify-between"><button onClick={() => setActiveSnippet(s)}
                className={`w-full text-left px-3 py-2 rounded-md ${activeSnippet?.id === s.id ?
                'bg-primary/10 text-primary' : 'hover:bg-gray-100'}`}>{s.name}</button><div
                className="flex opacity-0 group-hover:opacity-100 transition-opacity"><button
                onClick={() => navigator.clipboard.writeText(s.code)} className="ml-2 p-1 text-
                text-secondary hover:text-primary" title="Copy"><ClipboardDocumentIcon
                /></button><button onClick={() => handleDelete(s.id)} className="ml-2 p-1 text-
                text-secondary hover:text-red-500"
                title="Delete"><TrashIcon/></button></div></li>))}</ul>
                <div className="mt-4 pt-4 border-t border-border"><button onClick={handleAddNew}
                className="btn-primary w-full text-sm py-2">Add New Snippet</button></div>
            </aside>
            <main className="w-2/3 flex flex-col">
                {activeSnippet ? (<>
                    <div className="flex justify-between items-center mb-2">
                        {isEditingName ? <input type="text" value={activeSnippet.name}
                        onChange={handleNameChange} onBlur={() => setIsEditingName(false)} autoFocus
                        className="text-lg font-bold bg-gray-100 rounded px-2"/> : <h3 onDoubleClick={()
                        => setIsEditingName(true)} className="text-lg font-bold cursor-
                        pointer">{activeSnippet.name}</h3>}
                        <div className="flex gap-2">
```

```
                                <button onClick={handleEnhance} disabled={isEnhancing} className="flex items-
                                center gap-2 px-3 py-1 bg-purple-500 text-white font-bold text-xs rounded-md
                                disabled:bg-gray-400"><SparklesIcon /> AI Enhance</button>
                                <button onClick={() => navigator.clipboard.writeText(activeSnippet.code)}
                                className="px-3 py-1 bg-gray-100 text-xs rounded-md">Copy</button>
                                <button onClick={handleDownload} className="flex items-center gap-1 px-3 py-1
                                bg-gray-100 text-xs rounded-md"><ArrowDownTrayIcon className="w-4 h-4"/>
                                Download</button>
                            </div>
                        </div>
                        <textarea value={activeSnippet.code} onChange={e =>
                        updateSnippet({...activeSnippet, code: e.target.value})} className="flex-grow
                        p-4 bg-surface border border-border rounded-md resize-none font-mono text-sm
                        focus:ring-2 focus:ring-primary focus:outline-none"/>
                        <div className="mt-2 text-xs text-text-secondary">
                           <div className="flex items-center gap-2 flex-wrap">
                             <span className="font-bold">Tags:</span> {(activeSnippet.tags ?? []).map(t =>
                             <span key={t} className="bg-gray-200 px-2 py-0.5 rounded-full">{t}</span>)}
                             <input type="text" placeholder="+ Add tag" onKeyDown={handleTagsChange}
                             className="bg-transparent border-b border-border focus:outline-none
                             focus:border-primary w-24 text-xs px-1"/>
                           </div>
                        </div>
                    </>) : (<div className="flex-grow flex items-center justify-center bg-background
                    border border-border rounded-lg text-text-secondary">Select a snippet or create
                    a new one.</div>)}
                </main>
            </div>
        </div>
    );
};


// ===== ThemeDesigner_66.tsx =====



import React, { useState, useCallback, useEffect } from 'react';
import { SparklesIcon, ArrowDownTrayIcon } from '../icons.tsx';
// FIX: Import from index.ts where all services are exported
import { generateThemeFromDescription } from '../../services/index.ts';
import type { ColorTheme } from '../../types.ts';
import { LoadingSpinner } from '../shared/index.tsx';
import { HexColorPicker } from 'react-colorful';
import { downloadFile } from '../../services/index.ts';

const ColorInput: React.FC<{ label: string; color: string; onChange: (color:
string) => void; }> = ({ label, color, onChange }) => {
    const [isOpen, setIsOpen] = useState(false);
    return (
        <div className="relative">
            <div onClick={() => setIsOpen(!isOpen)} className="flex items-center justify-
            between p-3 bg-background rounded-md cursor-pointer border-2 border-border
            hover:border-gray-300">
               <div className="flex items-center gap-3">
                    <div className="w-6 h-6 rounded border border-border" style={{ backgroundColor:
                    color }} />
                    <span className="text-sm font-medium text-text-primary">{label}</span>
               </div>
               <span className="font-mono text-sm text-text-secondary">{color}</span>
            </div>
            {isOpen && (
```

```
                <div className="absolute z-10 top-full mt-2 right-0">
                     <div className="fixed inset-0" onClick={() => setIsOpen(false)} />
                    <HexColorPicker color={color} onChange={onChange} />
                </div>
            )}
        </div>
    );
};

const randomPrompts = [
    'A serene forest at dawn',
    'A retro 8-bit video game',
    'A cyberpunk cityscape at night',
    'A warm, cozy coffee shop',
    'An arctic expedition with icy blues',
    'A vibrant coral reef',
];

export const ThemeDesigner: React.FC<{ initialPrompt?: string }> = ({
initialPrompt }) => {
    const [theme, setTheme] = useState<ColorTheme>({
        primary: '#0047AB', background: '#F5F7FA', surface: '#FFFFFF', textPrimary:
        '#111827', textSecondary: '#6B7280',
    });
    const [prompt, setPrompt] = useState(initialPrompt || 'A professional and clean
    corporate blue theme.');
    const [isLoading, setIsLoading] = useState(false);
    const [error, setError] = useState('');

    const handleGenerate = useCallback(async (p: string) => {
        if (!p.trim()) { setError('Please enter a description.'); return; }
        setIsLoading(true); setError('');
        try {
            const newTheme = await generateThemeFromDescription(p);
            setTheme(newTheme);
        } catch (err) {
            setError(err instanceof Error ? err.message : "An unknown error occurred.");
        } finally {
            setIsLoading(false);
        }
    }, []);

    const handleRandom = () => {
        const randomPrompt = randomPrompts[Math.floor(Math.random() *
        randomPrompts.length)];
        setPrompt(randomPrompt);
        handleGenerate(randomPrompt);
    };

    const handleColorChange = (key: keyof ColorTheme, color: string) => {
        setTheme(prev => ({...prev, [key]: color}));
    };

    const getExportContent = (type: 'css' | 'tailwind') => {
        if (type === 'css') {
            return `:root {\n` +
                   `  --primary: ${theme.primary};\n` +
                   `  --background: ${theme.background};\n` +
                   `  --surface: ${theme.surface};\n` +
                   `  --text-primary: ${theme.textPrimary};\n` +
                   `  --text-secondary: ${theme.textSecondary};\n` +
                   `}`;
```

```
        }
        return `// tailwind.config.js
module.exports = {
  theme: {
    extend: {
      colors: {
        primary: '${theme.primary}',
        background: '${theme.background}',
        surface: '${theme.surface}',
        'text-primary': '${theme.textPrimary}',
        'text-secondary': '${theme.textSecondary}',
      },
    },
  },
};`;
    };

    useEffect(() => {
        if (initialPrompt) handleGenerate(initialPrompt);
    }, [initialPrompt, handleGenerate]);

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
            <header className="mb-6">
                <h1 className="text-3xl font-bold flex items-center"><SparklesIcon /><span
                className="ml-3">AI Theme Designer</span></h1>
                <p className="text-text-secondary mt-1">Describe, generate, fine-tune, and
                export a color theme for your application.</p>
            </header>
            <div className="flex-grow grid grid-cols-1 md:grid-cols-2 gap-6 min-h-0">
                <div className="md:col-span-1 flex flex-col gap-4 bg-surface border border-
                border p-6 rounded-lg overflow-y-auto">
                    <h3 className="text-xl font-bold">Describe your theme</h3>
                    <textarea value={prompt} onChange={e => setPrompt(e.target.value)}
                    className="p-2 bg-background border border-border rounded-md resize-none text-sm
                    h-24" placeholder="e.g., A light, airy theme for a blog" />
                     <div className="flex gap-2">
                         <button onClick={() => handleGenerate(prompt)} disabled={isLoading}
                         className="btn-primary w-full flex items-center justify-center gap-2 px-4 py-2">
                            {isLoading ? <LoadingSpinner /> : 'Generate Theme'}
                        </button>
                         <button onClick={handleRandom} disabled={isLoading} className="px-4 py-2 bg-
                         gray-100 border border-border rounded-md hover:bg-gray-200" title="Random
                         Theme">■</button>
                     </div>
                    {error && <p className="text-red-500 text-xs text-center">{error}</p>}

                    <div className="mt-4 border-t border-border pt-4 space-y-2 flex-grow overflow-y-
                    auto pr-2">
                         <h3 className="text-lg font-bold">Fine-Tune Palette</h3>
                         <ColorInput label="Primary" color={theme.primary} onChange={c =>
                         handleColorChange('primary', c)} />
                         <ColorInput label="Background" color={theme.background} onChange={c =>
                         handleColorChange('background', c)} />
                         <ColorInput label="Surface" color={theme.surface} onChange={c =>
                         handleColorChange('surface', c)} />
                         <ColorInput label="Text Primary" color={theme.textPrimary} onChange={c =>
                         handleColorChange('textPrimary', c)} />
                         <ColorInput label="Text Secondary" color={theme.textSecondary} onChange={c =>
                         handleColorChange('textSecondary', c)} />
                    </div>
                    <div className="flex-shrink-0 pt-4 border-t border-border flex flex-col gap-2">
```

```
                        <button onClick={() => downloadFile(getExportContent('css'), 'theme.css',
                        'text/css')} className="flex-1 text-sm py-2 bg-gray-100 border border-border
                        rounded-md flex items-center justify-center gap-2 hover:bg-gray-200">
                            <ArrowDownTrayIcon className="w-4 h-4"/> Download CSS
                        </button>
                        <button onClick={() => downloadFile(getExportContent('tailwind'),
                        'tailwind.config.js', 'application/javascript')} className="flex-1 text-sm py-2
                        bg-gray-100 border border-border rounded-md flex items-center justify-center
                        gap-2 hover:bg-gray-200">
                             <ArrowDownTrayIcon className="w-4 h-4"/> Download Tailwind Config
                        </button>
                    </div>
                </div>
                <div className="md:col-span-1 rounded-lg p-8 overflow-y-auto border border-
                border" style={{ backgroundColor: theme.background, color: theme.textPrimary }}>
                     <h3 className="text-2xl font-bold mb-6" style={{ color: theme.textPrimary
                     }}>Live Preview</h3>
                     <div className="p-6 rounded-lg grid grid-cols-1 md:grid-cols-2 gap-6" style={{
                     backgroundColor: theme.surface }}>
                        <div className="space-y-4">
                            <h4 className="text-lg font-bold">Sample Card</h4>
                            <p className="text-sm" style={{color: theme.textSecondary}}>This is a sample
                            card to demonstrate the theme colors. It contains a primary button and some
                            secondary text.</p>
                            <button className="px-4 py-2 rounded-md font-bold transition-colors" style={{
                            backgroundColor: theme.primary, color: theme.textSecondary.includes('#F') ?
                            '#000' : '#FFF' }}>Primary Button</button>
                        </div>
                         <div className="space-y-4">
                            <input type="text" placeholder="Text input" className="w-full px-3 py-2 rounded-
                            md border" style={{backgroundColor: theme.background, borderColor:
                            theme.primary, color: theme.textPrimary}} />
                            <div className="p-3 border rounded" style={{borderColor: theme.primary, color:
                            theme.textSecondary}}>
                                <p>A bordered container.</p>
                            </div>
                         </div>
                     </div>
                     <div className="mt-6">
                        <h4 className="text-lg font-bold mb-2">Typography</h4>
                        <h1>Heading 1</h1>
                        <h2>Heading 2</h2>
                        <p style={{color: theme.textPrimary}}>This is a paragraph of primary text. Lorem
                        ipsum dolor sit amet, consectetur adipiscing elit.</p>
                        <p style={{color: theme.textSecondary}}>This is secondary text, for less
                        important information.</p>
                     </div>
                </div>
            </div>
        </div>
    );
};


// ===== VisualGitTree_66.tsx =====



import React, { useState, useCallback, useEffect, useMemo } from 'react';
import { GitBranchIcon, ArrowDownTrayIcon } from '../icons.tsx';
// FIX: Import from index.ts where all services are exported
import { generateChangelogFromLogStream } from '../../services/index.ts';
```

```
import { LoadingSpinner, MarkdownRenderer } from '../shared/index.tsx';
import { downloadFile } from '../../services/index.ts';

const exampleLog = `* commit 3a4b5c6d7e8f9g0h1i2j3k4l5m6n7o8p9q0r (HEAD -> main,
origin/main)
|\\  Merge: 1a2b3c4 2d3e4f5
| | Author: Dev One <dev.one@example.com>
| | Date:   Mon Jul 15 11:30:00 2024 -0400
| |
| |     feat: Implement collapsible sidebar navigation
| |
* | commit 2d3e4f5g6h7i8j9k0l1m2n3o4p5q6r7s8t9u (feature/new-sidebar)
| | Author: Dev Two <dev.two@example.com>
| | Date:   Mon Jul 15 10:00:00 2024 -0400
| |
| |     feat: Add icons to sidebar items
| |
* | commit 1a2b3c4d5e6f7g8h9i0j1k2l3m4n5o6p7q8r
|/  Author: Dev One <dev.one@example.com>
|   Date:   Fri Jul 12 16:45:00 2024 -0400
|
|       fix: Correct user authentication bug`;

const CommitGraph = ({ logInput }: { logInput: string }) => {
    const commits = useMemo(() => {
        const lines = logInput.split('\n');
        const parsedCommits: any[] = [];
        let currentCommit: any = null;

        lines.forEach(line => {
            const commitMatch = line.match(/^.?[\\|/ ]*\* commit (\w+)(.*)/);
            if (commitMatch) {
                if (currentCommit) parsedCommits.push(currentCommit);
                currentCommit = {
                    hash: commitMatch[1],
                    shortHash: commitMatch[1].substring(0, 7),
                    refs: commitMatch[2].trim(),
                    message: '',
                    author: '',
                };
            } else if (currentCommit) {
                 if (line.includes('Author:')) currentCommit.author =
                 line.split('Author:')[1].trim();
                 else if (line.trim().length > 0 && !line.match(/^[\\|/ ]*[\\|/ ]/)) {
                     currentCommit.message += line.trim() + ' ';
                 }
            }
        });
        if (currentCommit) parsedCommits.push(currentCommit);

        return parsedCommits.map((c, i) => ({ ...c, x: 50, y: 50 + i * 60 }));
    }, [logInput]);

    return (
         <svg width="100%" height={50 + commits.length * 60} className="min-h-[200px]">
            {commits.map((commit, i) => {
                const parent = commits[i + 1];
                return (
                    <g key={commit.hash}>
                        {parent && <line x1={commit.x} y1={commit.y} x2={parent.x} y2={parent.y}
                        stroke="var(--color-border)" strokeWidth="2" />}
                        <g className="group cursor-pointer">
```

```
                            <circle cx={commit.x} cy={commit.y} r="8" fill="var(--color-primary)"
                            stroke="var(--color-surface)" strokeWidth="3" />
                            <foreignObject x={commit.x + 20} y={commit.y - 25} width="350" height="50">
                                <div className="text-sm p-1">
                                    <p className="font-bold truncate text-text-primary">{commit.message}</p>
                                    <p className="text-xs text-text-secondary font-mono">{commit.shortHash} <span
                                    className="text-amber-600">{commit.refs}</span></p>
                                </div>
                            </foreignObject>
                            <title>{`Commit: ${commit.hash}\nAuthor:
                            ${commit.author}\n\n${commit.message}`}</title>
                        </g>
                    </g>
                );
            })}
        </svg>
    );
};

export const VisualGitTree: React.FC<{ logInput?: string }> = ({ logInput:
initialLogInput }) => {
    const [logInput, setLogInput] = useState(initialLogInput || exampleLog);
    const [analysis, setAnalysis] = useState('');
    const [isLoading, setIsLoading] = useState(false);
    const [error, setError] = useState('');

    const handleAnalyze = useCallback(async (logToAnalyze: string) => {
        if (!logToAnalyze.trim()) {
            setError('Please paste git log output.');
            return;
        }
        setIsLoading(true);
        setError('');
        setAnalysis('');
        try {
            const stream = generateChangelogFromLogStream(logToAnalyze);
            let fullResponse = '';
            for await (const chunk of stream) {
                fullResponse += chunk;
                setAnalysis(fullResponse);
            }
        } catch (err) {
            const errorMessage = err instanceof Error ? err.message : 'An unknown error
            occurred.';
            setError(`Failed to analyze log: ${errorMessage}`);
        } finally {
            setIsLoading(false);
        }
    }, []);

    useEffect(() => {
        if (initialLogInput) {
            setLogInput(initialLogInput);
            handleAnalyze(initialLogInput);
        }
    }, [initialLogInput, handleAnalyze]);

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
            <header className="mb-6">
                <h1 className="text-3xl font-bold flex items-center">
                    <GitBranchIcon />
```

```
                    <span className="ml-3">Visual Git Tree</span>
                </h1>
                <p className="text-text-secondary mt-1">Paste your `git log --graph` output to
                visualize the history and get an AI summary.</p>
            </header>
            <div className="flex-grow grid grid-cols-1 lg:grid-cols-2 gap-6 h-full overflow-
            hidden">
                <div className="flex flex-col h-full">
                    <label htmlFor="log-input" className="text-sm font-medium text-text-secondary
                    mb-2">Git Log Output</label>
                    <textarea
                        id="log-input"
                        value={logInput}
                        onChange={(e) => setLogInput(e.target.value)}
                        placeholder="Paste your git log output here..."
                        className="flex-grow p-4 bg-surface border border-border rounded-md resize-none
                        font-mono text-sm"
                    />
                    <button
                        onClick={() => handleAnalyze(logInput)}
                        disabled={isLoading}
                        className="btn-primary mt-4 w-full flex items-center justify-center px-6 py-3"
                    >
                        {isLoading ? <LoadingSpinner /> : 'Analyze & Summarize'}
                    </button>
                </div>
                <div className="flex flex-col h-full gap-4">
                    <div className="flex flex-col h-1/2">
                        <label className="text-sm font-medium text-text-secondary mb-2">Commit
                        Graph</label>
                        <div className="flex-grow p-2 bg-surface border border-border rounded-md
                        overflow-auto">
                            <CommitGraph logInput={logInput} />
                        </div>
                    </div>
                     <div className="flex flex-col h-1/2">
                        <div className="flex justify-between items-center mb-2">
                            <label className="text-sm font-medium text-text-secondary">AI Summary</label>
                            {analysis && !isLoading && (
                                <button onClick={() => downloadFile(analysis, 'summary.md', 'text/markdown')}
                                className="flex items-center gap-1 px-3 py-1 bg-gray-100 text-xs rounded-md
                                hover:bg-gray-200">
                                    <ArrowDownTrayIcon className="w-4 h-4"/> Download Summary
                                </button>
                            )}
                        </div>
                        <div className="flex-grow p-4 bg-background border border-border rounded-md
                        overflow-y-auto">
                            {isLoading && <div className="flex items-center justify-center
                            h-full"><LoadingSpinner /></div>}
                            {error && <p className="text-red-500">{error}</p>}
                            {analysis && !isLoading && <MarkdownRenderer content={analysis} />}
                            {!isLoading && !analysis && !error && <div className="text-text-secondary h-full
                            flex items-center justify-center">AI summary will appear here.</div>}
                        </div>
                    </div>
                </div>
            </div>
        </div>
    );
};
// ===== WorkerThreadDebugger_66.tsx =====
```

```
import React, { useState, useCallback, useEffect } from 'react';
import { BugAntIcon, ArrowDownTrayIcon } from '../icons.tsx';
// FIX: Import from index.ts where all services are exported
import { analyzeConcurrencyStream } from '../../services/index.ts';
import { LoadingSpinner, MarkdownRenderer } from '../shared/index.tsx';
import { downloadFile } from '../../services/index.ts';

const exampleCode = `// main.js
const worker = new Worker('worker.js');

// This object is sent back and forth.
// A race condition can occur because both threads
// read the counter, increment it, and send it back.
// The final value depends on which thread's message
// is processed last.
const data = { counter: 0 };

worker.onmessage = function(e) {
  // Main thread reads and updates
  data.counter = e.data.counter;
  console.log('Main received:', data.counter);
  data.counter++;
  worker.postMessage(data);
};

// Start the process
console.log('Main starting with:', data.counter);
data.counter++;
worker.postMessage(data);


// worker.js
// onmessage = function(e) {
//   // Worker reads and updates
//   let receivedCounter = e.data.counter;
//   console.log('Worker received:', receivedCounter);
//   receivedCounter++;
//   postMessage({ counter: receivedCounter });
// }
`;

export const WorkerThreadDebugger: React.FC<{ codeInput?: string }> = ({
codeInput: initialCode }) => {
    const [codeInput, setCodeInput] = useState(initialCode || exampleCode);
    const [analysis, setAnalysis] = useState('');
    const [isLoading, setIsLoading] = useState(false);
    const [error, setError] = useState('');

    const handleAnalyze = useCallback(async (codeToAnalyze: string) => {
        if (!codeToAnalyze.trim()) {
            setError('Please paste some code to analyze.');
            return;
        }
        setIsLoading(true);
        setError('');
        setAnalysis('');
        try {
            const stream = analyzeConcurrencyStream(codeToAnalyze);
            let fullResponse = '';
            for await (const chunk of stream) {
                fullResponse += chunk;
                setAnalysis(fullResponse);
```

```
        }
    } catch (err) {
        const errorMessage = err instanceof Error ? err.message : 'An unknown error
        occurred.';
        setError(`Failed to analyze code: ${errorMessage}`);
    } finally {
        setIsLoading(false);
    }
}, []);

useEffect(() => {
    if (initialCode) {
        setCodeInput(initialCode);
        handleAnalyze(initialCode);
    }
}, [initialCode, handleAnalyze]);

return (
    <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
        <header className="mb-6">
            <h1 className="text-3xl font-bold flex items-center">
                <BugAntIcon />
                <span className="ml-3">AI Concurrency Analyzer</span>
            </h1>
            <p className="text-text-secondary mt-1">Analyze JavaScript code for potential
            Web Worker concurrency issues.</p>
        </header>
        <div className="flex-grow flex flex-col gap-4 min-h-0">
            <div className="flex flex-col flex-1 min-h-0">
                <label htmlFor="code-input" className="text-sm font-medium text-text-secondary
                mb-2">JavaScript Code</label>
                <textarea
                    id="code-input"
                    value={codeInput}
                    onChange={(e) => setCodeInput(e.target.value)}
                    placeholder="Paste your worker-related JS code here..."
                    className="flex-grow p-4 bg-surface border border-border rounded-md resize-none
                    font-mono text-sm"
                />
            </div>
             <div className="flex-shrink-0">
                <button
                    onClick={() => handleAnalyze(codeInput)}
                    disabled={isLoading}
                    className="btn-primary w-full max-w-xs mx-auto flex items-center justify-center
                    px-6 py-3"
                >
                    {isLoading ? <LoadingSpinner /> : 'Analyze Code'}
                </button>
            </div>
            <div className="flex flex-col flex-1 min-h-0">
                <div className="flex justify-between items-center mb-2">
                    <label className="text-sm font-medium text-text-secondary">AI Analysis</label>
                    {analysis && !isLoading && (
                         <button onClick={() => downloadFile(analysis, 'analysis.md', 'text/markdown')}
                         className="flex items-center gap-1 px-3 py-1 bg-gray-100 text-xs rounded-md
                         hover:bg-gray-200">
                            <ArrowDownTrayIcon className="w-4 h-4"/> Download
                        </button>
                    )}
                </div>
                <div className="flex-grow p-4 bg-background border border-border rounded-md
```

```
                    overflow-y-auto">
                        {isLoading && <div className="flex items-center justify-center
                        h-full"><LoadingSpinner /></div>}
                        {error && <p className="text-red-500">{error}</p>}
                        {analysis && !isLoading && <MarkdownRenderer content={analysis} />}
                        {!isLoading && !analysis && !error && <div className="text-text-secondary h-full
                        flex items-center justify-center">Analysis will appear here.</div>}
                    </div>
                </div>
            </div>
        </div>
    );
};


// ===== XbrlConverter_64.tsx =====



import React, { useState, useCallback } from 'react';
// FIX: Import from index.ts where all services are exported
import { convertJsonToXbrlStream } from '../../services/index.ts';
import { XbrlConverterIcon } from '../icons.tsx';
import { LoadingSpinner, MarkdownRenderer } from '../shared/index.tsx';

const exampleJson = `{
  "company": "ExampleCorp",
  "year": 2024,
  "quarter": 2,
  "revenue": {
    "amount": 1500000,
    "currency": "USD"
  },
  "profit": {
    "amount": 250000,
    "currency": "USD"
  }
}`;

export const XbrlConverter: React.FC<{ jsonInput?: string }> = ({ jsonInput:
initialJsonInput }) => {
    const [jsonInput, setJsonInput] = useState<string>(initialJsonInput ||
    exampleJson);
    const [xbrlOutput, setXbrlOutput] = useState<string>('');
    const [isLoading, setIsLoading] = useState<boolean>(false);
    const [error, setError] = useState<string>('');

    const handleConvert = useCallback(async (jsonToConvert: string) => {
        if (!jsonToConvert.trim()) {
            setError('Please enter valid JSON to convert.');
            return;
        }
        setIsLoading(true);
        setError('');
        setXbrlOutput('');
        try {
            const stream = convertJsonToXbrlStream(jsonToConvert);
            let fullResponse = '';
            for await (const chunk of stream) {
                fullResponse += chunk;
                setXbrlOutput(fullResponse);
            }
```

```
    } catch (err) {
        const errorMessage = err instanceof Error ? err.message : 'An unknown error
        occurred.';
        setError(`Failed to convert: ${errorMessage}`);
    } finally {
        setIsLoading(false);
    }
}, []);

return (
    <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
        <header className="mb-6">
            <h1 className="text-3xl font-bold flex items-center">
                <XbrlConverterIcon />
                <span className="ml-3">JSON to XBRL Converter</span>
            </h1>
            <p className="text-text-secondary mt-1">Convert JSON data into a simplified
            XBRL-like XML format using AI.</p>
        </header>
        <div className="flex-grow flex flex-col gap-4 min-h-0">
            <div className="flex flex-col flex-1 min-h-0">
                <label htmlFor="json-input" className="text-sm font-medium text-text-secondary
                mb-2">JSON Input</label>
                <textarea
                    id="json-input"
                    value={jsonInput}
                    onChange={(e) => setJsonInput(e.target.value)}
                    placeholder="Paste your JSON here..."
                    className="flex-grow p-4 bg-surface border border-border rounded-md resize-none
                    font-mono text-sm"
                />
            </div>
             <div className="flex-shrink-0">
                <button
                    onClick={() => handleConvert(jsonInput)}
                    disabled={isLoading}
                    className="btn-primary w-full max-w-xs mx-auto flex items-center justify-center
                    px-6 py-3"
                >
                    {isLoading ? <LoadingSpinner /> : 'Convert to XBRL'}
                </button>
            </div>
            <div className="flex flex-col flex-1 min-h-0">
                <label className="text-sm font-medium text-text-secondary mb-2">XBRL-like XML
                Output</label>
                <div className="relative flex-grow p-1 bg-background border border-border
                rounded-md overflow-y-auto">
                    {isLoading && !xbrlOutput && <div className="flex items-center justify-center
                    h-full"><LoadingSpinner /></div>}
                    {error && <p className="p-4 text-red-500">{error}</p>}
                    {xbrlOutput && <MarkdownRenderer content={'```xml\n' +
                    xbrlOutput.replace(/```xml\n|```/g, '') + '\n```'} />}
                    {!isLoading && xbrlOutput && <button onClick={() =>
                    navigator.clipboard.writeText(xbrlOutput)} className="absolute top-2 right-2
                    px-2 py-1 bg-gray-100 hover:bg-gray-200 rounded-md text-xs">Copy XML</button>}
                    {!isLoading && !xbrlOutput && !error && <div className="text-text-secondary
                    h-full flex items-center justify-center">Output will appear here.</div>}
                </div>
            </div>
        </div>
    </div>
);
```

```
};


// ===== CronJobBuilder_68.tsx =====

import React, { useState, useMemo, useCallback, useEffect } from 'react';
import { CommandLineIcon, SparklesIcon } from '../icons.tsx';
// FIX: Import CronParts type from its definition file instead of the service
barrel.
import { generateCronFromDescription } from '../../services/index.ts';
import type { CronParts } from '../../types.ts';
import { LoadingSpinner } from '../shared/index.tsx';

const CronPartSelector: React.FC<{ label: string, value: string, onChange:
(value: string) => void, options: (string|number)[] }> = ({ label, value,
onChange, options }) => {
    return (
        <div>
            <label className="block text-sm font-medium text-text-secondary">{label}</label>
            <select value={value} onChange={e => onChange(e.target.value)} className="w-full
            mt-1 px-3 py-2 rounded-md bg-surface border border-border">
                <option value="*">* (every)</option>
                {options.map(o => <option key={o} value={o}>{o}</option>)}
            </select>
        </div>
    );
};

export const CronJobBuilder: React.FC<{ initialPrompt?: string }> = ({
initialPrompt }) => {
    const [minute, setMinute] = useState('0');
    const [hour, setHour] = useState('17');
    const [dayOfMonth, setDayOfMonth] = useState('*');
    const [month, setMonth] = useState('*');
    const [dayOfWeek, setDayOfWeek] = useState('1-5');
    const [aiPrompt, setAiPrompt] = useState(initialPrompt || 'every weekday at
    5pm');
    const [isLoading, setIsLoading] = useState(false);

    const cronExpression = useMemo(() => {
        return `${minute} ${hour} ${dayOfMonth} ${month} ${dayOfWeek}`;
    }, [minute, hour, dayOfMonth, month, dayOfWeek]);

    const handleAiGenerate = useCallback(async (p: string) => {
        if (!p) return;
        setIsLoading(true);
        try {
            const result: CronParts = await generateCronFromDescription(p);
            setMinute(result.minute);
            setHour(result.hour);
            setDayOfMonth(result.dayOfMonth);
            setMonth(result.month);
            setDayOfWeek(result.dayOfWeek);
        } catch (e) {
            console.error(e);
        } finally {
            setIsLoading(false);
        }
    }, []);

    useEffect(() => {
        if (initialPrompt) {
```

```
            setAiPrompt(initialPrompt);
            handleAiGenerate(initialPrompt);
        }
    }, [initialPrompt, handleAiGenerate]);

    return (
        <div className="h-full flex flex-col p-4 sm:p-6 lg:p-8 text-text-primary">
            <header className="mb-6">
                <h1 className="text-3xl font-bold flex items-center">
                    <CommandLineIcon />
                    <span className="ml-3">AI Cron Job Builder</span>
                </h1>
                <p className="text-text-secondary mt-1">Visually construct a cron expression or
                describe it in plain English.</p>
            </header>
             <div className="flex gap-2 mb-6">
                <input type="text" value={aiPrompt} onChange={e => setAiPrompt(e.target.value)}
                placeholder="Describe a schedule..." className="flex-grow px-3 py-1.5 rounded-md
                bg-surface border border-border text-sm"/>
                <button onClick={() => handleAiGenerate(aiPrompt)} disabled={isLoading}
                className="btn-primary px-4 py-1.5 flex items-center gap-2">
                    {isLoading ? <LoadingSpinner /> : <SparklesIcon />} AI Generate
                </button>
            </div>
            <div className="grid grid-cols-2 md:grid-cols-5 gap-4 mb-6">
                <CronPartSelector label="Minute" value={minute} onChange={setMinute}
                options={Array.from({length: 60}, (_, i) => i)} />
                <CronPartSelector label="Hour" value={hour} onChange={setHour}
                options={Array.from({length: 24}, (_, i) => i)} />
                <CronPartSelector label="Day (Month)" value={dayOfMonth}
                onChange={setDayOfMonth} options={Array.from({length: 31}, (_, i) => i + 1)} />
                <CronPartSelector label="Month" value={month} onChange={setMonth}
                options={Array.from({length: 12}, (_, i) => i + 1)} />
                <CronPartSelector label="Day (Week)" value={dayOfWeek} onChange={setDayOfWeek}
                options={Array.from({length: 7}, (_, i) => i)} />
            </div>
            <div className="bg-surface p-4 rounded-lg text-center border border-border">
                <p className="text-text-secondary text-sm">Generated Expression</p>
                <p className="font-mono text-primary text-2xl mt-1">{cronExpression}</p>
                 <button onClick={() => navigator.clipboard.writeText(cronExpression)}
                 className="mt-4 px-3 py-1 bg-gray-100 hover:bg-gray-200 rounded-md text-
                 xs">Copy</button>
            </div>
        </div>
    );
};

// ===== AiStoryScaffolding.tsx =====

import React, { useState, useEffect, useRef } from 'react';
import type { StoryDocument, AppState, RobotState, EditorActions, PageHandlers }
from '../../types';
import * as gemini from '../../services/geminiService_story';
import { createPdf } from '../../services/pdfService';
import { APP_TITLE } from '../../constants.ts';
import { BookOpenIcon, DownloadIcon, BackArrowIcon } from '../icons.tsx';
import Loader from './story-scaffolding/Loader';
import DataInput from './story-scaffolding/DataInput';
import StoryEditor from './story-scaffolding/StoryViewer';
import Robot from './story-scaffolding/Robot';

const PAGE_CHUNK_SIZE = 3200; // ~3200 tokens per page as per blueprint
```

```
const AiStoryScaffolding: React.FC = () => {
    const [appState, setAppState] = useState<AppState>('INPUT');
    const [storyDocument, setStoryDocument] = useState<StoryDocument | null>(null);
    const [isLoading, setIsLoading] = useState<boolean>(false);
    const [loadingMessage, setLoadingMessage] = useState<string>('');
    const [error, setError] = useState<string | null>(null);
    const [generationStatus, setGenerationStatus] = useState({ active: false,
    completed: 0, total: 0 });
    const generationController = useRef<AbortController | null>(null);

    // Robot State
    const [mousePosition, setMousePosition] = useState({ x: 0, y: 0 });
    const [robotState, setRobotState] = useState<RobotState>('idle');

    useEffect(() => {
        const handleMouseMove = (e: MouseEvent) => setMousePosition({ x: e.clientX, y:
        e.clientY });
        window.addEventListener('mousemove', handleMouseMove);
        return () => window.removeEventListener('mousemove', handleMouseMove);
    }, []);

    const handleReset = () => {
        if (generationController.current) generationController.current.abort();
        setAppState('INPUT');
        setStoryDocument(null);
        setError(null);
        setIsLoading(false);
        setGenerationStatus({ active: false, completed: 0, total: 0 });
        setRobotState('idle');
    };

    const handleScaffoldGeneration = async (data: string, mood: string) => {
        setIsLoading(true);
        setLoadingMessage('Analyzing document...');
        setError(null);
        setRobotState('thinking');

        try {
            const chunks: string[] = [];
            for (let i = 0; i < data.length; i += PAGE_CHUNK_SIZE) {
                chunks.push(data.substring(i, i + PAGE_CHUNK_SIZE));
            }
            if (chunks.length === 0) throw new Error("No text data to process.");

            setLoadingMessage('Generating story title...');
            const title = await gemini.generateStoryTitle(chunks[0]);

            const doc: StoryDocument = { id: crypto.randomUUID(), title, mood, chapters: []
            };
            const chapterChunks = chunks.reduce((acc, chunk, i) => {
                const chapterIndex = Math.floor(i / 5); // Group chunks into chapters of ~5
                pages
                if (!acc[chapterIndex]) acc[chapterIndex] = [];
                acc[chapterIndex].push(chunk);
                return acc;
            }, [] as string[][]);

            doc.chapters = chapterChunks.map((_, i) => ({
                id: crypto.randomUUID(),
                title: `Chapter ${i + 1} (pending...)`,
                summary: 'AI summary will appear here.',
                pages: [],
```

```
        }));

        setStoryDocument(doc);
        setGenerationStatus({ active: true, completed: 0, total: chapterChunks.length
        });
        setAppState('EDITING');
        setIsLoading(false);

        startChapterGeneration(doc.id, chapterChunks);

    } catch (err) {
        console.error(err);
        setError('Failed to generate a story scaffold. Please try again.');
        handleReset();
    }
};

const startChapterGeneration = async (docId: string, chapterChunks: string[][])
=> {
    generationController.current = new AbortController();
    const { signal } = generationController.current;
    setRobotState('writing');

    for (let i = 0; i < chapterChunks.length; i++) {
        if (signal.aborted) {
            setGenerationStatus(s => ({ ...s, active: false }));
            setRobotState('idle');
            return;
        }
        try {
            const chapter = await gemini.generateChapterFromChunk(chapterChunks[i], i + 1,
            chapterChunks.length, storyDocument?.title || 'Untitled');
            setStoryDocument(doc => {
                if (!doc || doc.id !== docId) return doc;
                const newChapters = [...doc.chapters];
                newChapters[i] = chapter;
                return { ...doc, chapters: newChapters };
            });
            setGenerationStatus(s => ({ ...s, completed: i + 1 }));
        } catch (err) {
            if (err instanceof Error && err.name === 'AbortError') return;
            console.error(`Failed to generate chapter ${i + 1}`, err);
            setStoryDocument(doc => {
                if (!doc) return null;
                doc.chapters[i].title = `Chapter ${i+1} (Failed)`;
                return { ...doc };
            });
        }
    }
    setGenerationStatus({ active: false, completed: chapterChunks.length, total:
    chapterChunks.length });
    setRobotState('idle');
};

const handleStopGeneration = () => generationController.current?.abort();

const handleContinueGeneration = () => {
    alert("Resuming generation is a complex feature! For now, you can manually
    complete the story.");
};

const runPageAction = async <T,>(originalRobotState: RobotState, action: () =>
```

```
Promise<T>) => {
    setRobotState(originalRobotState);
    try {
        await action();
    } catch (err) {
        console.error("Page action failed", err);
        setError("An AI action failed. Please try again.");
    } finally {
        setRobotState('idle');
    }
};

const pageHandlers: PageHandlers = {
    onUpdatePage: (chapterId, pageId, updates) => {
        setStoryDocument(doc => {
            if (!doc) return null;
            return {
                ...doc,
                chapters: doc.chapters.map(c => c.id === chapterId ? {
                    ...c,
                    pages: c.pages.map(p => p.id === pageId ? { ...p, ...updates } : p)
                } : c)
            };
        });
    },
    onExpandTextStream: async (chapterId, pageId) => {
         await runPageAction('writing', async () => {
             const chapter = storyDocument?.chapters.find(c => c.id === chapterId);
             const page = chapter?.pages.find(p => p.id === pageId);
             if (!page) return;
             const stream = gemini.expandPageTextStream(page.page_text);
             for await (const chunk of stream) {
                 setStoryDocument(doc => {
                    if (!doc) return null;
                    return {
                        ...doc,
                        chapters: doc.chapters.map(c => c.id === chapterId ? {
                            ...c,
                            pages: c.pages.map(p => p.id === pageId ? { ...p, page_text: p.page_text + chunk
                            } : p)
                        } : c)
                    };
                });
             }
        });
    },
    onGenerateImage: async (chapterId, pageId) => {
        await runPageAction('illustrating', async () => {
            const chapter = storyDocument?.chapters.find(c => c.id === chapterId);
            const page = chapter?.pages.find(p => p.id === pageId);
            if (!page || !storyDocument) return;
            const imageUrl = await gemini.generatePageImage(page.page_text ||
            page.ai_suggestions[0] || "A fantasy landscape", storyDocument.mood);
            if (imageUrl) {
                pageHandlers.onUpdatePage(chapterId, pageId, { images: [...page.images,
                imageUrl] });
            }
        });
    },
    onAutoWritePageStream: async (chapterId, pageId) => {
        await runPageAction('writing', async () => {
             pageHandlers.onUpdatePage(chapterId, pageId, { page_text: '' }); // Clear text
```

```
            first
            const stream = gemini.autoWritePageStream(storyDocument?.title || '',
            storyDocument?.chapters.find(c=>c.id === chapterId)?.title || '',
            storyDocument?.chapters.find(c => c.id === chapterId)?.pages.find(p => p.id ===
            pageId)?.ai_suggestions.join(' ') || '');
            for await (const chunk of stream) {
                setStoryDocument(doc => {
                   if (!doc) return null;
                   return {
                       ...doc,
                       chapters: doc.chapters.map(c => c.id === chapterId ? {
                           ...c,
                           pages: c.pages.map(p => p.id === pageId ? { ...p, page_text: p.page_text + chunk
                           } : p)
                       } : c)
                   };
                });
            }
        });
    }
};

const editorActions: EditorActions = {
   onAutoDraftAll: async () => {
        if (!storyDocument) return;
        setRobotState('writing');
        try {
            for (const chapter of storyDocument.chapters) {
                for (const page of chapter.pages) {
                    if (!page.page_text.trim()) {
                        await pageHandlers.onAutoWritePageStream(chapter.id, page.id);
                    }
                }
            }
        } catch (err) {
            console.error(err);
            setError("Auto-drafting failed.");
        } finally {
            setRobotState('idle');
        }
    },
    onSuggestTitles: async () => {
        if (!storyDocument) return;
        await runPageAction('thinking', async () => {
            const newTitles = await gemini.suggestNewChapterTitles(storyDocument);
            setStoryDocument(doc => {
                if (!doc) return null;
                return {
                    ...doc,
                    chapters: doc.chapters.map((c, i) => ({ ...c, title: newTitles[i] || c.title }))
                };
            });
        });
    },
    onSummarizeChapters: async () => {
         if (!storyDocument) return;
         await runPageAction('thinking', async () => {
            const summaries = await gemini.generateChapterSummaries(storyDocument);
             setStoryDocument(doc => {
                if (!doc) return null;
                return {
                    ...doc,
```

```
                    chapters: doc.chapters.map((c, i) => ({ ...c, summary: summaries[i] || c.summary
                    }))
                };
            });
        });
    }
};

const handlePdfDownload = () => {
    if (!storyDocument) return;
    setIsLoading(true);
    setLoadingMessage('Generating PDF...');
    setRobotState('thinking');
    try {
        createPdf(storyDocument);
    } catch (err) {
        console.error(err);
        setError('Failed to create PDF.');
    } finally {
        setIsLoading(false);
        setRobotState('idle');
    }
};

const renderContent = () => {
    if (isLoading) return <Loader message={loadingMessage} />;
    switch (appState) {
        case 'INPUT': return <DataInput onProcess={handleScaffoldGeneration} />;
        case 'EDITING': return storyDocument ? (
            <StoryEditor
               doc={storyDocument}
               setDoc={setStoryDocument}
               pageHandlers={pageHandlers}
               editorActions={editorActions}
               generationStatus={generationStatus}
               onStopGeneration={handleStopGeneration}
               onContinueGeneration={handleContinueGeneration}
            />
        ) : <Loader message="Loading editor..." />;
        default: return <DataInput onProcess={handleScaffoldGeneration} />;
    }
};

return (
    <div className="text-gray-100 font-sans h-full bg-gray-900">
        <Robot robotState={robotState} mousePosition={mousePosition} />
        <div className="min-h-full p-4 sm:p-6 lg:p-8">
            <div className="max-w-[120rem] mx-auto">
                <header className="flex justify-between items-center mb-6">
                    <div className="flex items-center space-x-3">
                        <BookOpenIcon className="w-8 h-8 text-violet-400" />
                        <h1 className="text-2xl sm:text-3xl font-bold tracking-tight bg-gradient-to-r
                        from-purple-400 to-violet-300 bg-clip-text text-transparent">
                            {APP_TITLE}
                        </h1>
                    </div>
                     {appState !== 'INPUT' && (
                         <div className="flex items-center gap-4">
                            <button onClick={handlePdfDownload} className="flex items-center space-x-2 px-4
                            py-2 bg-violet-600 hover:bg-violet-500 rounded-lg text-sm font-semibold
                            transition-colors"><DownloadIcon className="w-4 h-4"/><span>Export
                            PDF</span></button>
```

```
                                <button onClick={handleReset} className="flex items-center space-x-2 px-4 py-2
                                bg-gray-700 hover:bg-gray-600 rounded-lg text-sm font-medium transition-
                                colors"><BackArrowIcon className="w-4 h-4"/><span>Start Over</span></button>
                            </div>
                        )}
                    </header>
                    <main className="bg-gray-800/50 rounded-2xl shadow-2xl backdrop-blur-sm border
                    border-gray-700/50 min-h-[calc(100vh-12rem)] overflow-hidden">
                        {error && (
                            <div className="bg-red-500/20 border border-red-500 text-red-300 px-4 py-3
                            rounded-lg m-6 relative" role="alert">
                                <strong className="font-bold">Error: </strong>
                                <span className="block sm:inline">{error}</span>
                                <button onClick={() => setError(null)} className="absolute top-0 bottom-0
                                right-0 px-4 py-3">&times;</button>
                            </div>
                        )}
                        {renderContent()}
                    </main>
                </div>
            </div>
        </div>
    );
};

export default AiStoryScaffolding;

// ===== DataInput.tsx =====

import React, { useState } from 'react';
import { UploadIcon, SparklesIcon } from '../../icons';

interface DataInputProps {
    onProcess: (data: string, mood: string) => void;
}

const MAX_FILE_SIZE = 100 * 1024 * 1024; // 100 MB

const DataInput: React.FC<DataInputProps> = ({ onProcess }) => {
    const [file, setFile] = useState<File | null>(null);
    const [mood, setMood] = useState<string>('');
    const [isParsing, setIsParsing] = useState(false);
    const [dragActive, setDragActive] = useState(false);
    const [error, setError] = useState<string | null>(null);

    const extractTextFromPdf = async (pdfFile: File): Promise<string> => {
        window.pdfjsLib.GlobalWorkerOptions.workerSrc =
        `https://cdnjs.cloudflare.com/ajax/libs/pdf.js/2.6.347/pdf.worker.min.js`;
        const loadingTask = window.pdfjsLib.getDocument(URL.createObjectURL(pdfFile));
        const pdf = await loadingTask.promise;
        let fullText = '';
        for (let i = 1; i <= pdf.numPages; i++) {
            const page = await pdf.getPage(i);
            const textContent = await page.getTextContent();
            const pageText = textContent.items.map((item: any) => item.str).join(' ');
            fullText += pageText + '\n\n';
        }
        return fullText;
    };

    const extractTextFromTxt = (txtFile: File): Promise<string> => {
        return new Promise((resolve, reject) => {
```

```
        const reader = new FileReader();
        reader.onload = (event) => resolve(event.target?.result as string);
        reader.onerror = (error) => reject(error);
        reader.readAsText(txtFile);
    });
};

const handleFile = (selectedFile: File | null) => {
    if (!selectedFile) return;
    if (selectedFile.size > MAX_FILE_SIZE) {
        setError(`File is too large. Maximum size is 100MB.`);
        setFile(null);
        return;
    }
    if (selectedFile.type === 'application/pdf' || selectedFile.type ===
    'text/plain') {
        setFile(selectedFile);
        setError(null);
    } else {
        setFile(null);
        setError('Please upload a valid PDF or TXT file.');
    }
};

const handleDrag = (e: React.DragEvent) => {
    e.preventDefault();
    e.stopPropagation();
    if (e.type === "dragenter" || e.type === "dragover") setDragActive(true);
    else if (e.type === "dragleave") setDragActive(false);
};

const handleDrop = (e: React.DragEvent) => {
    e.preventDefault();
    e.stopPropagation();
    setDragActive(false);
    if (e.dataTransfer.files?.[0]) handleFile(e.dataTransfer.files[0]);
};

const handleChange = (e: React.ChangeEvent<HTMLInputElement>) => {
    e.preventDefault();
    if (e.target.files?.[0]) handleFile(e.target.files[0]);
};

const handleSubmit = async (e: React.FormEvent) => {
    e.preventDefault();
    if (!file) return;

    setIsParsing(true);
    setError(null);
    try {
        const text = file.type === 'application/pdf' ? await extractTextFromPdf(file) :
        await extractTextFromTxt(file);
        if (!text.trim()) {
             setError('Could not extract any text from the file.');
             setIsParsing(false);
             return;
        }
        onProcess(text, mood || 'cinematic, detailed illustration');
    } catch (err) {
        console.error("Failed to parse file", err);
        setError("Could not read the file. It might be corrupted.");
        setIsParsing(false);
```

```
        }
    };

    return (
        <div className="flex flex-col items-center justify-center h-full text-center
        p-4">
            <div className="w-full max-w-2xl">
                <UploadIcon className="mx-auto h-16 w-16 text-violet-400 mb-4" />
                <h2 className="text-2xl font-bold text-white mb-2">Upload Your Document</h2>
                <p className="text-gray-400 mb-6">Provide a PDF or TXT document and an art
                style. The AI will generate a story scaffold.</p>
                <form onSubmit={handleSubmit} className="w-full space-y-4">
                    <label htmlFor="file-upload" onDragEnter={handleDrag} onDragLeave={handleDrag}
                    onDragOver={handleDrag} onDrop={handleDrop} className={`relative flex flex-col
                    items-center justify-center w-full h-40 p-4 border-2 border-dashed rounded-lg
                    cursor-pointer transition-colors ${dragActive ? "border-violet-400 bg-
                    violet-900/30" : "border-gray-600 hover:border-gray-500"}`}>
                        <p className="text-gray-400 mb-2">{ file ? file.name : 'Drag & drop a file here,
                        or click to select' }</p>
                        <p className="text-xs text-gray-500">PDF or TXT, up to 100MB</p>
                        <input id="file-upload" type="file" className="hidden" onChange={handleChange}
                        accept="application/pdf,text/plain" />
                    </label>
                    <input id="mood-input" type="text" value={mood} onChange={(e) =>
                    setMood(e.target.value)} className="w-full bg-gray-900/50 border border-gray-600
                    rounded-lg p-3 text-gray-300 focus:outline-none focus:ring-2 focus:ring-
                    violet-500 placeholder:text-gray-500" placeholder="Optional: Describe the art
                    style (e.g., dark fantasy, watercolor)" />
                    {error && <p className="mt-2 text-sm text-red-400">{error}</p>}
                    <button type="submit" disabled={!file || isParsing} className="mt-4 group
                    relative inline-flex items-center justify-center px-8 py-3 text-lg font-bold
                    text-white transition-all duration-200 bg-gray-900 rounded-xl focus:outline-none
                    focus:ring-2 focus:ring-offset-2 focus:ring-gray-900 disabled:opacity-50
                    disabled:cursor-not-allowed">
                         <span className="absolute -inset-2 rounded-xl bg-gradient-to-r from-purple-600
                         to-violet-600 opacity-75 blur transition-all duration-1000 group-
                         hover:opacity-100 group-hover:-inset-1 disabled:opacity-0"></span>
                         <SparklesIcon className="w-6 h-6 mr-3"/>
                         {isParsing ? 'Reading file...' : 'Generate Story Scaffold'}
                    </button>
                </form>
            </div>
        </div>
    );
};

export default DataInput;

// ===== Loader.tsx =====

import React from 'react';
import { SparklesIcon } from '../../icons';

interface LoaderProps {
    message: string;
}

const Loader: React.FC<LoaderProps> = ({ message }) => {
    return (
        <div className="flex flex-col items-center justify-center h-full p-8 text-
        center">
            <div className="relative">
```

```
                <div className="w-24 h-24 border-4 border-dashed rounded-full animate-spin
                border-violet-400"></div>
                <div className="absolute inset-0 flex items-center justify-center">
                    <SparklesIcon className="w-10 h-10 text-violet-400" />
                </div>
            </div>
            <p className="mt-6 text-lg font-medium text-gray-300">{message}</p>
        </div>
    );
};

export default Loader;

// ===== MagazinePreview.tsx =====

import React from 'react';
import type { StoryDocument } from '../../../types';

interface MagazinePreviewProps {
    doc: StoryDocument;
}

const MagazinePreview: React.FC<MagazinePreviewProps> = ({ doc }) => {

    const renderPageContent = (text: string, images: string[], isFirstPage: boolean)
    => {
        const paragraphs = text.split('\n').filter(p => p.trim() !== '');
        const contentElements = [];
        const mutableImages = [...images];

        if (isFirstPage && mutableImages.length > 0) {
            const heroImage = mutableImages.shift();
            contentElements.push(<img key="hero-img" src={heroImage} alt="Chapter
            illustration" className="magazine-hero-image" />);
        }

        const textBlock = [];
        for (let i = 0; i < paragraphs.length; i++) {
            if (i > 0 && i % 2 === 0 && mutableImages.length > 0) {
                 const imgUrl = mutableImages.shift();
                 if(imgUrl) textBlock.push(<img key={`img-${i}`} src={imgUrl} alt="Story
                 illustration" className={i % 4 === 0 ? 'img-left' : 'img-right'} />);
            }
            textBlock.push(<p key={`p-${i}`}>{paragraphs[i]}</p>);
        }
        contentElements.push(<div key="text-content" className="magazine-
        columns">{textBlock}</div>);

        if (mutableImages.length > 0) {
            contentElements.push(
                <div key="remaining-images" className="mt-4">
                    {mutableImages.map((imgUrl, i) => <img key={`img-rem-${i}`} src={imgUrl}
                    alt="Story illustration" className="w-full h-auto rounded-lg mb-4" />)}
                </div>
            );
        }

        return contentElements;
    };

    return (
        <div className="p-4 magazine-preview">
```

```
            <h1>{doc.title}</h1>
            {doc.chapters.map(chapter => (
                <div key={chapter.id}>
                    <h2>{chapter.title}</h2>
                    {chapter.pages.map((page, index) => (
                         <div key={page.id} className="clearfix">
                           {renderPageContent(page.page_text, page.images, index === 0)}
                        </div>
                    ))}
                </div>
            ))}
        </div>
    );
};

export default MagazinePreview;


// ===== PlanDisplay.tsx =====

import React, { useState } from 'react';
import type { ChapterScaffold, PageScaffold, PageHandlers } from
'../../../types';
import { ImagePlusIcon, FileTextIcon, WandIcon, ChevronDownIcon } from
'../../icons';

interface PageComponentProps {
    page: PageScaffold;
    chapterId: string;
    handlers: PageHandlers;
    isActionLoading: boolean;
}

const PageComponent: React.FC<PageComponentProps> = ({ page, chapterId,
handlers, isActionLoading }) => {
    const [mainImage, setMainImage] = useState(page.images[0] || null);

    React.useEffect(() => {
        if (!mainImage && page.images.length > 0) setMainImage(page.images[0]);
        else if (page.images.length > 0 && !page.images.includes(mainImage!))
        setMainImage(page.images[0]);
        else if (page.images.length === 0) setMainImage(null);
    }, [page.images, mainImage]);

    return (
        <div className="grid grid-cols-1 md:grid-cols-2 gap-4 p-4 bg-gray-800/50
        rounded-lg border border-gray-700">
            <div className="flex flex-col">
                <h4 className="font-bold text-violet-300 mb-2">Page {page.page_number}</h4>
                <textarea value={page.page_text} onChange={(e) =>
                handlers.onUpdatePage(chapterId, page.id, { page_text: e.target.value })}
                className="w-full flex-grow bg-gray-900/50 border border-gray-600 rounded-md p-2
                text-gray-300 focus:outline-none focus:ring-2 focus:ring-violet-500 resize-none
                min-h-[200px]" placeholder="Start writing or use an AI action..." />
                <div className="flex flex-wrap gap-2 mt-2">
                     <button onClick={() => handlers.onAutoWritePageStream(chapterId, page.id)}
                     disabled={isActionLoading} className="flex-1 flex items-center justify-center
                     gap-2 px-3 py-2 bg-violet-600 hover:bg-violet-500 rounded-lg text-xs font-medium
                     transition-colors disabled:opacity-50"><WandIcon className="w-4 h-4" /> AI Write
                     Page</button>
                     <button onClick={() => handlers.onExpandTextStream(chapterId, page.id)}
                     disabled={isActionLoading || !page.page_text} className="flex-1 flex items-
```

```
                    center justify-center gap-2 px-3 py-2 bg-gray-600 hover:bg-gray-500 rounded-lg
                    text-xs font-medium transition-colors disabled:opacity-50"><FileTextIcon
                    className="w-4 h-4" /> Expand Text</button>
                </div>
            </div>
            <div className="flex flex-col">
                <div className="aspect-video bg-gray-700 rounded-lg border border-gray-600 flex
                items-center justify-center overflow-hidden flex-grow">
                    {mainImage ? <img src={mainImage} alt={`Illustration for page
                    ${page.page_number}`} className="w-full h-full object-cover"/> : <p
                    className="text-gray-500 text-sm">No Image</p>}
                </div>
                {page.images.length > 1 && (
                    <div className="flex gap-2 mt-2">
                        {page.images.map((img, idx) => <button key={idx} onClick={() =>
                        setMainImage(img)} className={`w-16 h-10 rounded-md overflow-hidden border-2
                        ${mainImage === img ? 'border-violet-400' : 'border-transparent'}`}><img
                        src={img} alt={`Thumbnail ${idx+1}`} className="w-full h-full object-cover"
                        /></button>)}
                    </div>
                )}
                 <button onClick={() => handlers.onGenerateImage(chapterId, page.id)}
                 disabled={isActionLoading} className="w-full mt-2 flex items-center justify-
                 center space-x-2 px-3 py-2 bg-purple-600 hover:bg-purple-500 rounded-lg text-sm
                 font-medium transition-colors disabled:opacity-50"><ImagePlusIcon className="w-5
                 h-5"/><span>{page.images.length > 0 ? 'Add Another Image' : 'Generate
                 Image'}</span></button>
            </div>
        </div>
    );
};

interface ChapterComponentProps {
    chapter: ChapterScaffold;
    pageHandlers: PageHandlers;
    isActionLoading: boolean;
}

const ChapterComponent: React.FC<ChapterComponentProps> = ({ chapter,
pageHandlers, isActionLoading }) => {
    const [isExpanded, setIsExpanded] = useState(true);

    return (
        <div className="mb-4 bg-gray-800/30 rounded-lg border border-gray-700 overflow-
        hidden">
            <button onClick={() => setIsExpanded(!isExpanded)} className="w-full flex
            justify-between items-center p-4 bg-gray-700/50 hover:bg-gray-700/80 transition-
            colors">
                <div>
                    <h3 className="text-xl font-bold text-left">{chapter.title}</h3>
                    <p className="text-sm text-gray-400 text-left">{chapter.summary}</p>
                </div>
                <ChevronDownIcon className={`w-6 h-6 transform transition-transform duration-300
                ${isExpanded ? 'rotate-180' : ''}`} />
            </button>
            {isExpanded && (
                <div className="p-4 space-y-4">
                    {chapter.pages.length > 0 ? chapter.pages.map(page => <PageComponent
                    key={page.id} page={page} chapterId={chapter.id} handlers={pageHandlers}
                    isActionLoading={isActionLoading} />) : <p className="text-center py-8 text-
                    gray-500">This chapter has no pages yet.</p>}
                </div>
```

```
            )}
        </div>
    );
};

export default ChapterComponent;

// ===== Robot.tsx =====

import React from 'react';
import { motion, AnimatePresence } from 'framer-motion';
import type { RobotState } from '../../../types';
import { PaintBrushIcon } from '../../icons';

interface RobotProps {
    robotState: RobotState;
    mousePosition: { x: number; y: number };
}

const Robot: React.FC<RobotProps> = ({ robotState, mousePosition }) => {
    const { x, y } = mousePosition;
    const centerX = window.innerWidth / 2;
    const angle = Math.atan2(y - (window.innerHeight - 80), x - centerX) * (180 /
    Math.PI) + 90;
    const clampedAngle = Math.max(-45, Math.min(45, angle));

    const variants = {
        idle: { y: 0 },
        thinking: { y: -5, scale: 1.05 },
        writing: { y: 2, rotate: [0, 1, -1, 0] },
        illustrating: { y: 2, rotate: [0, 2, -2, 0] },
    };

    return (
        <motion.div className="fixed bottom-0 left-1/2 -translate-x-1/2 z-50 pointer-
        events-none" style={{ x: x - window.innerWidth / 2, y: y - window.innerHeight }}
        initial={{ opacity: 0, y: 100 }} animate={{ opacity: 1, y: 0 }} transition={{
        duration: 0.5 }}>
            <motion.div animate={robotState} variants={variants} transition={{ y: { repeat:
            Infinity, repeatType: 'reverse', duration: 1.5, ease: 'easeInOut' }, rotate: {
            repeat: Infinity, repeatType: 'reverse', duration: 0.5 } }}>
                <svg width="120" height="120" viewBox="0 0 100 100">
                    <ellipse cx="50" cy="95" rx="30" ry="5" fill="black" opacity="0.2" />
                    <motion.g animate={{ rotate: clampedAngle }} style={{ transformOrigin: '50px
                    70px' }}>
                        <rect x="30" y="40" width="40" height="40" rx="10" fill="#4B5563" />
                        <rect x="25" y="45" width="50" height="30" rx="10" fill="#6B7280" />
                        <motion.circle cx="50" cy="55" r="12" fill="#1F2937" />
                        <motion.circle cx="50" cy="55" r="8" fill="#A78BFA" animate={{ scale: robotState
                        === 'thinking' ? [1, 1.2, 1] : 1 }} transition={{ repeat: Infinity, duration: 1
                        }} />
                        <circle cx="52" cy="53" r="3" fill="white" />
                        <line x1="50" y1="40" x2="50" y2="25" stroke="#4B5563" strokeWidth="3" />
                        <motion.circle cx="50" cy="25" r="5" fill="#A78BFA" animate={{ scale: robotState
                        === 'thinking' ? [1, 1.5, 1] : 1, boxShadow: robotState === 'thinking' ? '0 0
                        15px #A78BFA' : '0 0 0px #A78BFA' }} transition={{ repeat: Infinity, duration:
                        0.8 }} />
                    </motion.g>
                    <AnimatePresence>
                    {robotState === 'writing' && (
                        <motion.g key="writing" initial={{ y: 20, opacity: 0 }} animate={{ y: 0,
                        opacity: 1 }} exit={{ y: 20, opacity: 0 }}>
```

```
                            <rect x="20" y="80" width="60" height="15" rx="3" fill="#374151" />
                            <motion.circle cx="35" cy="80" r="5" fill="#6B7280" animate={{ y: [0, -3, 0]}}
                            transition={{ repeat: Infinity, duration: 0.3, delay: 0.1 }} />
                            <motion.circle cx="65" cy="80" r="5" fill="#6B7280" animate={{ y: [0, -3, 0]}}
                            transition={{ repeat: Infinity, duration: 0.3 }} />
                        </motion.g>
                    )}
                     {robotState === 'illustrating' && (
                        <motion.g key="illustrating" initial={{ y: 10, opacity: 0, rotate: -20, x: 10 }}
                        animate={{ y: 0, opacity: 1, rotate: 10, x: 0 }} exit={{ y: 10, opacity: 0,
                        rotate: -20 }} transition={{ type: 'spring', stiffness: 300, damping: 20 }}>
                            <circle cx="75" cy="75" r="6" fill="#6B7280" />
                            <g transform="translate(65, 55) rotate(45)"><PaintBrushIcon className="w-8 h-8
                            text-purple-400" /></g>
                        </motion.g>
                    )}
                    </AnimatePresence>
                </svg>
            </motion.div>
        </motion.div>
    );
};

export default Robot;

// ===== StoryViewer.tsx =====

import React, { useState, useEffect, useRef } from 'react';
import type { StoryDocument, PageHandlers, EditorActions } from
'../../../types';
import ChapterComponent from './PlanDisplay';
import MagazinePreview from './MagazinePreview';
import { PlusIcon, DocumentPlusIcon, WandIcon, ChatBubbleBottomCenterTextIcon,
SparklesIcon, PlayIcon, StopIcon } from '../../icons';

interface StoryEditorProps {
    doc: StoryDocument;
    setDoc: React.Dispatch<React.SetStateAction<StoryDocument | null>>;
    pageHandlers: PageHandlers;
    editorActions: EditorActions;
    generationStatus: { active: boolean, completed: number, total: number };
    onStopGeneration: () => void;
    onContinueGeneration: () => void;
}

const StoryEditor: React.FC<StoryEditorProps> = ({ doc, setDoc, pageHandlers,
editorActions, generationStatus, onStopGeneration, onContinueGeneration }) => {
    const fileInputRef = useRef<HTMLInputElement>(null);
    const [activeChapterId, setActiveChapterId] = useState<string | null>(null);

    useEffect(() => {
        if (doc && doc.chapters.length > 0 && !activeChapterId) {
            setActiveChapterId(doc.chapters[0].id);
        }
    }, [doc, activeChapterId]);

    const activeChapter = doc.chapters.find(c => c.id === activeChapterId);

    return (
        <div className="grid grid-cols-1 xl:grid-cols-12 gap-6 h-full p-4">
            <aside className="xl:col-span-3 bg-gray-900/60 p-4 rounded-lg border border-
            gray-700 flex flex-col">
```

```
    <h3 className="font-bold mb-3 text-violet-300">Table of Contents</h3>
    <div className="flex-grow overflow-y-auto space-y-1 pr-2 -mr-2">
        {doc.chapters.map((chapter, index) => (
            <button key={chapter.id} onClick={() => setActiveChapterId(chapter.id)}
            className={`block w-full text-left p-2 rounded-md transition-colors text-sm
            ${activeChapterId === chapter.id ? 'bg-violet-900/50 text-white' : 'hover:bg-
            gray-700/50 text-gray-300'}`}>
                <span className="font-semibold text-gray-500 mr-2">{index + 1}.</span>
                <span className="font-semibold">{chapter.title}</span>
                 <p className="text-xs text-gray-400 pl-6 truncate">{chapter.summary}</p>
            </button>
        ))}
    </div>
    {generationStatus.total > 0 && generationStatus.completed <
    generationStatus.total && (
        <div className="mt-4 p-3 bg-gray-800 rounded-lg border border-gray-700">
            <div className="flex justify-between items-center mb-2">
                <p className="text-sm font-medium text-gray-300">{generationStatus.active ?
                'Generating Chapters...' : 'Generation Paused'}</p>
                <p className="text-sm text-gray-400">{generationStatus.completed} /
                {generationStatus.total}</p>
            </div>
            <div className="w-full bg-gray-600 rounded-full h-2 mb-2">
                <div className="bg-violet-500 h-2 rounded-full" style={{ width:
                `${(generationStatus.completed / generationStatus.total) * 100}%` }}></div>
            </div>
            {generationStatus.active ? <button onClick={onStopGeneration} className="w-full
            flex items-center justify-center gap-2 text-sm p-2 bg-red-600/50 hover:bg-
            red-600/80 rounded-md"><StopIcon className="w-4 h-4" /> Stop</button> : <button
            onClick={onContinueGeneration} className="w-full flex items-center justify-
            center gap-2 text-sm p-2 bg-green-600/50 hover:bg-green-600/80 rounded-
            md"><PlayIcon className="w-4 h-4" /> Continue</button>}
        </div>
    )}
     <div className="mt-4 space-y-2 border-t border-gray-700 pt-4">
        <div className="relative group">
            <button className="w-full flex items-center justify-center gap-2 text-sm p-2 bg-
            violet-600/80 hover:bg-violet-600 rounded-md"><WandIcon className="w-4 h-4"/> AI
            Assist</button>
            <div className="absolute bottom-full mb-2 w-full bg-gray-800 border border-
            gray-700 rounded-lg p-2 space-y-2 z-10 opacity-0 group-hover:opacity-100
            transition-opacity pointer-events-none group-hover:pointer-events-auto">
                <button onClick={editorActions.onSuggestTitles} className="w-full text-left
                text-sm p-2 hover:bg-gray-700 rounded-md flex items-center gap-2"><SparklesIcon
                className="w-4 h-4"/> Improve Titles</button>
                <button onClick={editorActions.onSummarizeChapters} className="w-full text-left
                text-sm p-2 hover:bg-gray-700 rounded-md flex items-center
                gap-2"><ChatBubbleBottomCenterTextIcon className="w-4 h-4"/> Update
                Summaries</button>
            </div>
        </div>
     </div>
</aside>
<div className="xl:col-span-5 overflow-y-auto pr-2 -mr-2">
    <button onClick={editorActions.onAutoDraftAll} className="w-full mb-4 flex
    items-center justify-center gap-2 p-3 bg-violet-800 hover:bg-violet-700 rounded-
    lg font-bold"><WandIcon className="w-5 h-5"/> AI, Write Full Draft</button>
    {activeChapter ? <ChapterComponent chapter={activeChapter}
    pageHandlers={pageHandlers} isActionLoading={false} /> : <p className="flex
    items-center justify-center h-full text-gray-500">Select a chapter to begin
    editing.</p>}
</div>
```

```
            <aside className="xl:col-span-4 bg-gray-900/60 p-4 rounded-lg border border-
            gray-700 overflow-y-auto">
                <MagazinePreview doc={doc} />
            </aside>
        </div>
    );
};

export default StoryEditor;

// ===== AiStoryScaffolding_1.tsx =====

import React, { useState, useEffect, useRef } from 'react';
import type { StoryDocument, AppState, RobotState, EditorActions, PageHandlers }
from '../../types';
import * as gemini from '../../services/geminiService_story';
import { createPdf } from '../../services/pdfService';
import { APP_TITLE } from '../../constants.ts';
import { BookOpenIcon, DownloadIcon, BackArrowIcon } from '../icons.tsx';
import Loader from './story-scaffolding/Loader';
import DataInput from './story-scaffolding/DataInput';
import StoryEditor from './story-scaffolding/StoryViewer';
import Robot from './story-scaffolding/Robot';

const PAGE_CHUNK_SIZE = 3200; // ~3200 tokens per page as per blueprint

export const AiStoryScaffolding: React.FC = () => {
    const [appState, setAppState] = useState<AppState>('INPUT');
    const [storyDocument, setStoryDocument] = useState<StoryDocument | null>(null);
    const [isLoading, setIsLoading] = useState<boolean>(false);
    const [loadingMessage, setLoadingMessage] = useState<string>('');
    const [error, setError] = useState<string | null>(null);
    const [generationStatus, setGenerationStatus] = useState({ active: false,
    completed: 0, total: 0 });
    const generationController = useRef<AbortController | null>(null);

    // Robot State
    const [mousePosition, setMousePosition] = useState({ x: 0, y: 0 });
    const [robotState, setRobotState] = useState<RobotState>('idle');

    useEffect(() => {
        const handleMouseMove = (e: MouseEvent) => setMousePosition({ x: e.clientX, y:
        e.clientY });
        window.addEventListener('mousemove', handleMouseMove);
        return () => window.removeEventListener('mousemove', handleMouseMove);
    }, []);

    const handleReset = () => {
        if (generationController.current) generationController.current.abort();
        setAppState('INPUT');
        setStoryDocument(null);
        setError(null);
        setIsLoading(false);
        setGenerationStatus({ active: false, completed: 0, total: 0 });
        setRobotState('idle');
    };

    const handleScaffoldGeneration = async (data: string, mood: string) => {
        setIsLoading(true);
        setLoadingMessage('Analyzing document...');
        setError(null);
        setRobotState('thinking');
```

```
    try {
        const chunks: string[] = [];
        for (let i = 0; i < data.length; i += PAGE_CHUNK_SIZE) {
            chunks.push(data.substring(i, i + PAGE_CHUNK_SIZE));
        }
        if (chunks.length === 0) throw new Error("No text data to process.");

        setLoadingMessage('Generating story title...');
        const title = await gemini.generateStoryTitle(chunks[0]);

        const doc: StoryDocument = { id: crypto.randomUUID(), title, mood, chapters: []
        };
        const chapterChunks = chunks.reduce((acc, chunk, i) => {
            const chapterIndex = Math.floor(i / 5); // Group chunks into chapters of ~5
            pages
            if (!acc[chapterIndex]) acc[chapterIndex] = [];
            acc[chapterIndex].push(chunk);
            return acc;
        }, [] as string[][]);

        doc.chapters = chapterChunks.map((_, i) => ({
            id: crypto.randomUUID(),
            title: `Chapter ${i + 1} (pending...)`,
            summary: 'AI summary will appear here.',
            pages: [],
        }));

        setStoryDocument(doc);
        setGenerationStatus({ active: true, completed: 0, total: chapterChunks.length
        });
        setAppState('EDITING');
        setIsLoading(false);

        startChapterGeneration(doc.id, chapterChunks);

    } catch (err) {
        console.error(err);
        setError('Failed to generate a story scaffold. Please try again.');
        handleReset();
    }
};

const startChapterGeneration = async (docId: string, chapterChunks: string[][])
=> {
    generationController.current = new AbortController();
    const { signal } = generationController.current;
    setRobotState('writing');

    for (let i = 0; i < chapterChunks.length; i++) {
        if (signal.aborted) {
            setGenerationStatus(s => ({ ...s, active: false }));
            setRobotState('idle');
            return;
        }
        try {
            const chapter = await gemini.generateChapterFromChunk(chapterChunks[i], i + 1,
            chapterChunks.length, storyDocument?.title || 'Untitled');
            setStoryDocument(doc => {
                if (!doc || doc.id !== docId) return doc;
                const newChapters = [...doc.chapters];
                newChapters[i] = chapter;
                return { ...doc, chapters: newChapters };
```

```
            });
            setGenerationStatus(s => ({ ...s, completed: i + 1 }));
        } catch (err) {
            if (err instanceof Error && err.name === 'AbortError') return;
            console.error(`Failed to generate chapter ${i + 1}`, err);
            setStoryDocument(doc => {
                if (!doc) return null;
                doc.chapters[i].title = `Chapter ${i+1} (Failed)`;
                return { ...doc };
            });
        }
    }
    setGenerationStatus({ active: false, completed: chapterChunks.length, total:
    chapterChunks.length });
    setRobotState('idle');
};

const handleStopGeneration = () => generationController.current?.abort();

const handleContinueGeneration = () => {
    alert("Resuming generation is a complex feature! For now, you can manually
    complete the story.");
};

const runPageAction = async <T,>(originalRobotState: RobotState, action: () =>
Promise<T>) => {
    setRobotState(originalRobotState);
    try {
        await action();
    } catch (err) {
        console.error("Page action failed", err);
        setError("An AI action failed. Please try again.");
    } finally {
        setRobotState('idle');
    }
};

const pageHandlers: PageHandlers = {
    onUpdatePage: (chapterId, pageId, updates) => {
        setStoryDocument(doc => {
            if (!doc) return null;
            return {
                ...doc,
                chapters: doc.chapters.map(c => c.id === chapterId ? {
                    ...c,
                    pages: c.pages.map(p => p.id === pageId ? { ...p, ...updates } : p)
                } : c)
            };
        });
    },
    onExpandTextStream: async (chapterId, pageId) => {
         await runPageAction('writing', async () => {
             const chapter = storyDocument?.chapters.find(c => c.id === chapterId);
             const page = chapter?.pages.find(p => p.id === pageId);
             if (!page) return;
             const stream = gemini.expandPageTextStream(page.page_text);
             for await (const chunk of stream) {
                 setStoryDocument(doc => {
                    if (!doc) return null;
                    return {
                        ...doc,
                        chapters: doc.chapters.map(c => c.id === chapterId ? {
```

```
                            ...c,
                            pages: c.pages.map(p => p.id === pageId ? { ...p, page_text: p.page_text + chunk
                            } : p)
                        } : c)
                    };
                });
            }
        });
    },
    onGenerateImage: async (chapterId, pageId) => {
        await runPageAction('illustrating', async () => {
            const chapter = storyDocument?.chapters.find(c => c.id === chapterId);
            const page = chapter?.pages.find(p => p.id === pageId);
            if (!page || !storyDocument) return;
            const imageUrl = await gemini.generatePageImage(page.page_text ||
            page.ai_suggestions[0] || "A fantasy landscape", storyDocument.mood);
            if (imageUrl) {
                pageHandlers.onUpdatePage(chapterId, pageId, { images: [...page.images,
                imageUrl] });
            }
        });
    },
    onAutoWritePageStream: async (chapterId, pageId) => {
        await runPageAction('writing', async () => {
             pageHandlers.onUpdatePage(chapterId, pageId, { page_text: '' }); // Clear text
             first
             const stream = gemini.autoWritePageStream(storyDocument?.title || '',
             storyDocument?.chapters.find(c=>c.id === chapterId)?.title || '',
             storyDocument?.chapters.find(c => c.id === chapterId)?.pages.find(p => p.id ===
             pageId)?.ai_suggestions.join(' ') || '');
             for await (const chunk of stream) {
                 setStoryDocument(doc => {
                    if (!doc) return null;
                    return {
                        ...doc,
                        chapters: doc.chapters.map(c => c.id === chapterId ? {
                            ...c,
                            pages: c.pages.map(p => p.id === pageId ? { ...p, page_text: p.page_text + chunk
                            } : p)
                        } : c)
                    };
                });
            }
        });
    }
};

const editorActions: EditorActions = {
   onAutoDraftAll: async () => {
        if (!storyDocument) return;
        setRobotState('writing');
        try {
            for (const chapter of storyDocument.chapters) {
                for (const page of chapter.pages) {
                    if (!page.page_text.trim()) {
                        await pageHandlers.onAutoWritePageStream(chapter.id, page.id);
                    }
                }
            }
        } catch (err) {
            console.error(err);
            setError("Auto-drafting failed.");
```

```
        } finally {
            setRobotState('idle');
        }
    },
    onSuggestTitles: async () => {
        if (!storyDocument) return;
        await runPageAction('thinking', async () => {
            const newTitles = await gemini.suggestNewChapterTitles(storyDocument);
            setStoryDocument(doc => {
                if (!doc) return null;
                return {
                    ...doc,
                    chapters: doc.chapters.map((c, i) => ({ ...c, title: newTitles[i] || c.title }))
                };
            });
        });
    },
    onSummarizeChapters: async () => {
         if (!storyDocument) return;
         await runPageAction('thinking', async () => {
            const summaries = await gemini.generateChapterSummaries(storyDocument);
             setStoryDocument(doc => {
                if (!doc) return null;
                return {
                    ...doc,
                    chapters: doc.chapters.map((c, i) => ({ ...c, summary: summaries[i] || c.summary
                    }))
                };
            });
        });
    }
};

const handlePdfDownload = () => {
    if (!storyDocument) return;
    setIsLoading(true);
    setLoadingMessage('Generating PDF...');
    setRobotState('thinking');
    try {
        createPdf(storyDocument);
    } catch (err) {
        console.error(err);
        setError('Failed to create PDF.');
    } finally {
        setIsLoading(false);
        setRobotState('idle');
    }
};

const renderContent = () => {
    if (isLoading) return <Loader message={loadingMessage} />;
    switch (appState) {
        case 'INPUT': return <DataInput onProcess={handleScaffoldGeneration} />;
        case 'EDITING': return storyDocument ? (
             <StoryEditor
                doc={storyDocument}
                setDoc={setStoryDocument}
                pageHandlers={pageHandlers}
                editorActions={editorActions}
                generationStatus={generationStatus}
                onStopGeneration={handleStopGeneration}
                onContinueGeneration={handleContinueGeneration}
```

```
                />
            ) : <Loader message="Loading editor..." />;
            default: return <DataInput onProcess={handleScaffoldGeneration} />;
        }
    };

    return (
        <div className="text-gray-100 font-sans h-full bg-gray-900">
            <Robot robotState={robotState} mousePosition={mousePosition} />
            <div className="min-h-full p-4 sm:p-6 lg:p-8">
                <div className="max-w-[120rem] mx-auto">
                    <header className="flex justify-between items-center mb-6">
                        <div className="flex items-center space-x-3">
                            <BookOpenIcon className="w-8 h-8 text-violet-400" />
                            <h1 className="text-2xl sm:text-3xl font-bold tracking-tight bg-gradient-to-r
                            from-purple-400 to-violet-300 bg-clip-text text-transparent">
                                {APP_TITLE}
                            </h1>
                        </div>
                         {appState !== 'INPUT' && (
                             <div className="flex items-center gap-4">
                                <button onClick={handlePdfDownload} className="flex items-center space-x-2 px-4
                                py-2 bg-violet-600 hover:bg-violet-500 rounded-lg text-sm font-semibold
                                transition-colors"><DownloadIcon className="w-4 h-4"/><span>Export
                                PDF</span></button>
                                <button onClick={handleReset} className="flex items-center space-x-2 px-4 py-2
                                bg-gray-700 hover:bg-gray-600 rounded-lg text-sm font-medium transition-
                                colors"><BackArrowIcon className="w-4 h-4"/><span>Start Over</span></button>
                             </div>
                        )}
                    </header>
                    <main className="bg-gray-800/50 rounded-2xl shadow-2xl backdrop-blur-sm border
                    border-gray-700/50 min-h-[calc(100vh-12rem)] overflow-hidden">
                        {error && (
                            <div className="bg-red-500/20 border border-red-500 text-red-300 px-4 py-3
                            rounded-lg m-6 relative" role="alert">
                                <strong className="font-bold">Error: </strong>
                                <span className="block sm:inline">{error}</span>
                                <button onClick={() => setError(null)} className="absolute top-0 bottom-0
                                right-0 px-4 py-3">&times;</button>
                            </div>
                        )}
                        {renderContent()}
                    </main>
                </div>
            </div>
        </div>
    );
};


// ===== DataInput_1.tsx =====

import React, { useState } from 'react';
import { UploadIcon, SparklesIcon } from '../../icons.tsx';

interface DataInputProps {
    onProcess: (data: string, mood: string) => void;
}

const MAX_FILE_SIZE = 100 * 1024 * 1024; // 100 MB
const DataInput: React.FC<DataInputProps> = ({ onProcess }) => {
```

```
const [file, setFile] = useState<File | null>(null);
const [mood, setMood] = useState<string>('');
const [isParsing, setIsParsing] = useState(false);
const [dragActive, setDragActive] = useState(false);
const [error, setError] = useState<string | null>(null);

const extractTextFromPdf = async (pdfFile: File): Promise<string> => {
    window.pdfjsLib.GlobalWorkerOptions.workerSrc =
    `https://cdnjs.cloudflare.com/ajax/libs/pdf.js/2.6.347/pdf.worker.min.js`;
    const loadingTask = window.pdfjsLib.getDocument(URL.createObjectURL(pdfFile));
    const pdf = await loadingTask.promise;
    let fullText = '';
    for (let i = 1; i <= pdf.numPages; i++) {
        const page = await pdf.getPage(i);
        const textContent = await page.getTextContent();
        const pageText = textContent.items.map((item: any) => item.str).join(' ');
        fullText += pageText + '\n\n';
    }
    return fullText;
};

const extractTextFromTxt = (txtFile: File): Promise<string> => {
    return new Promise((resolve, reject) => {
        const reader = new FileReader();
        reader.onload = (event) => resolve(event.target?.result as string);
        reader.onerror = (error) => reject(error);
        reader.readAsText(txtFile);
    });
};

const handleFile = (selectedFile: File | null) => {
    if (!selectedFile) return;
    if (selectedFile.size > MAX_FILE_SIZE) {
        setError(`File is too large. Maximum size is 100MB.`);
        setFile(null);
        return;
    }
    if (selectedFile.type === 'application/pdf' || selectedFile.type ===
    'text/plain') {
        setFile(selectedFile);
        setError(null);
    } else {
        setFile(null);
        setError('Please upload a valid PDF or TXT file.');
    }
};

const handleDrag = (e: React.DragEvent) => {
    e.preventDefault();
    e.stopPropagation();
    if (e.type === "dragenter" || e.type === "dragover") setDragActive(true);
    else if (e.type === "dragleave") setDragActive(false);
};

const handleDrop = (e: React.DragEvent) => {
    e.preventDefault();
    e.stopPropagation();
    setDragActive(false);
    if (e.dataTransfer.files?.[0]) handleFile(e.dataTransfer.files[0]);
};

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

```
    e.preventDefault();
    if (e.target.files?.[0]) handleFile(e.target.files[0]);
};

const handleSubmit = async (e: React.FormEvent) => {
    e.preventDefault();
    if (!file) return;

    setIsParsing(true);
    setError(null);
    try {
        const text = file.type === 'application/pdf' ? await extractTextFromPdf(file) :
        await extractTextFromTxt(file);
        if (!text.trim()) {
             setError('Could not extract any text from the file.');
             setIsParsing(false);
             return;
        }
        onProcess(text, mood || 'cinematic, detailed illustration');
    } catch (err) {
        console.error("Failed to parse file", err);
        setError("Could not read the file. It might be corrupted.");
        setIsParsing(false);
    }
};

return (
    <div className="flex flex-col items-center justify-center h-full text-center
    p-4">
        <div className="w-full max-w-2xl">
            <UploadIcon className="mx-auto h-16 w-16 text-violet-400 mb-4" />
            <h2 className="text-2xl font-bold text-white mb-2">Upload Your Document</h2>
            <p className="text-gray-400 mb-6">Provide a PDF or TXT document and an art
            style. The AI will generate a story scaffold.</p>
            <form onSubmit={handleSubmit} className="w-full space-y-4">
                <label htmlFor="file-upload" onDragEnter={handleDrag} onDragLeave={handleDrag}
                onDragOver={handleDrag} onDrop={handleDrop} className={`relative flex flex-col
                items-center justify-center w-full h-40 p-4 border-2 border-dashed rounded-lg
                cursor-pointer transition-colors ${dragActive ? "border-violet-400 bg-
                violet-900/30" : "border-gray-600 hover:border-gray-500"}`}>
                    <p className="text-gray-400 mb-2">{ file ? file.name : 'Drag & drop a file here,
                    or click to select' }</p>
                    <p className="text-xs text-gray-500">PDF or TXT, up to 100MB</p>
                    <input id="file-upload" type="file" className="hidden" onChange={handleChange}
                    accept="application/pdf,text/plain" />
                </label>
                <input id="mood-input" type="text" value={mood} onChange={(e) =>
                setMood(e.target.value)} className="w-full bg-gray-900/50 border border-gray-600
                rounded-lg p-3 text-gray-300 focus:outline-none focus:ring-2 focus:ring-
                violet-500 placeholder:text-gray-500" placeholder="Optional: Describe the art
                style (e.g., dark fantasy, watercolor)" />
                {error && <p className="mt-2 text-sm text-red-400">{error}</p>}
                <button type="submit" disabled={!file || isParsing} className="mt-4 group
                relative inline-flex items-center justify-center px-8 py-3 text-lg font-bold
                text-white transition-all duration-200 bg-gray-900 rounded-xl focus:outline-none
                focus:ring-2 focus:ring-offset-2 focus:ring-gray-900 disabled:opacity-50
                disabled:cursor-not-allowed">
                     <span className="absolute -inset-2 rounded-xl bg-gradient-to-r from-purple-600
                     to-violet-600 opacity-75 blur transition-all duration-1000 group-
                     hover:opacity-100 group-hover:-inset-1 disabled:opacity-0"></span>
                     <SparklesIcon className="w-6 h-6 mr-3"/>
                     {isParsing ? 'Reading file...' : 'Generate Story Scaffold'}
```

```
                    </button>
                </form>
            </div>
        </div>
    );
};

export default DataInput;


// ===== Loader_1.tsx =====

import React from 'react';
import { SparklesIcon } from '../../icons.tsx';

interface LoaderProps {
    message: string;
}

const Loader: React.FC<LoaderProps> = ({ message }) => {
    return (
        <div className="flex flex-col items-center justify-center h-full p-8 text-
        center">
            <div className="relative">
                <div className="w-24 h-24 border-4 border-dashed rounded-full animate-spin
                border-violet-400"></div>
                <div className="absolute inset-0 flex items-center justify-center">
                    <SparklesIcon className="w-10 h-10 text-violet-400" />
                </div>
            </div>
            <p className="mt-6 text-lg font-medium text-gray-300">{message}</p>
        </div>
    );
};

export default Loader;


// ===== PlanDisplay_1.tsx =====

import React, { useState } from 'react';
import type { ChapterScaffold, PageScaffold, PageHandlers } from
'../../../types';
import { ImagePlusIcon, FileTextIcon, WandIcon, ChevronDownIcon } from
'../../icons.tsx';

interface PageComponentProps {
    page: PageScaffold;
    chapterId: string;
    handlers: PageHandlers;
    isActionLoading: boolean;
}

const PageComponent: React.FC<PageComponentProps> = ({ page, chapterId,
handlers, isActionLoading }) => {
    const [mainImage, setMainImage] = useState(page.images[0] || null);

    React.useEffect(() => {
        if (!mainImage && page.images.length > 0) setMainImage(page.images[0]);
        else if (page.images.length > 0 && !page.images.includes(mainImage!))
        setMainImage(page.images[0]);
        else if (page.images.length === 0) setMainImage(null);
```

```
    }, [page.images, mainImage]);

    return (
        <div className="grid grid-cols-1 md:grid-cols-2 gap-4 p-4 bg-gray-800/50
        rounded-lg border border-gray-700">
            <div className="flex flex-col">
                <h4 className="font-bold text-violet-300 mb-2">Page {page.page_number}</h4>
                <textarea value={page.page_text} onChange={(e) =>
                handlers.onUpdatePage(chapterId, page.id, { page_text: e.target.value })}
                className="w-full flex-grow bg-gray-900/50 border border-gray-600 rounded-md p-2
                text-gray-300 focus:outline-none focus:ring-2 focus:ring-violet-500 resize-none
                min-h-[200px]" placeholder="Start writing or use an AI action..." />
                <div className="flex flex-wrap gap-2 mt-2">
                     <button onClick={() => handlers.onAutoWritePageStream(chapterId, page.id)}
                     disabled={isActionLoading} className="flex-1 flex items-center justify-center
                     gap-2 px-3 py-2 bg-violet-600 hover:bg-violet-500 rounded-lg text-xs font-medium
                     transition-colors disabled:opacity-50"><WandIcon className="w-4 h-4" /> AI Write
                     Page</button>
                     <button onClick={() => handlers.onExpandTextStream(chapterId, page.id)}
                     disabled={isActionLoading || !page.page_text} className="flex-1 flex items-
                     center justify-center gap-2 px-3 py-2 bg-gray-600 hover:bg-gray-500 rounded-lg
                     text-xs font-medium transition-colors disabled:opacity-50"><FileTextIcon
                     className="w-4 h-4" /> Expand Text</button>
                </div>
            </div>
            <div className="flex flex-col">
                <div className="aspect-video bg-gray-700 rounded-lg border border-gray-600 flex
                items-center justify-center overflow-hidden flex-grow">
                     {mainImage ? <img src={mainImage} alt={`Illustration for page
                     ${page.page_number}`} className="w-full h-full object-cover"/> : <p
                     className="text-gray-500 text-sm">No Image</p>}
                </div>
                {page.images.length > 1 && (
                    <div className="flex gap-2 mt-2">
                        {page.images.map((img, idx) => <button key={idx} onClick={() =>
                        setMainImage(img)} className={`w-16 h-10 rounded-md overflow-hidden border-2
                        ${mainImage === img ? 'border-violet-400' : 'border-transparent'}`}><img
                        src={img} alt={`Thumbnail ${idx+1}`} className="w-full h-full object-cover"
                        /></button>)}
                    </div>
                )}
                 <button onClick={() => handlers.onGenerateImage(chapterId, page.id)}
                 disabled={isActionLoading} className="w-full mt-2 flex items-center justify-
                 center space-x-2 px-3 py-2 bg-purple-600 hover:bg-purple-500 rounded-lg text-sm
                 font-medium transition-colors disabled:opacity-50"><ImagePlusIcon className="w-5
                 h-5"/><span>{page.images.length > 0 ? 'Add Another Image' : 'Generate
                 Image'}</span></button>
            </div>
        </div>
    );
};

interface ChapterComponentProps {
    chapter: ChapterScaffold;
    pageHandlers: PageHandlers;
    isActionLoading: boolean;
}

const ChapterComponent: React.FC<ChapterComponentProps> = ({ chapter,
pageHandlers, isActionLoading }) => {
    const [isExpanded, setIsExpanded] = useState(true);
    return (
```

```
        <div className="mb-4 bg-gray-800/30 rounded-lg border border-gray-700 overflow-
        hidden">
            <button onClick={() => setIsExpanded(!isExpanded)} className="w-full flex
            justify-between items-center p-4 bg-gray-700/50 hover:bg-gray-700/80 transition-
            colors">
                <div>
                    <h3 className="text-xl font-bold text-left">{chapter.title}</h3>
                    <p className="text-sm text-gray-400 text-left">{chapter.summary}</p>
                </div>
                <ChevronDownIcon className={`w-6 h-6 transform transition-transform duration-300
                ${isExpanded ? 'rotate-180' : ''}`} />
            </button>
            {isExpanded && (
                <div className="p-4 space-y-4">
                    {chapter.pages.length > 0 ? chapter.pages.map(page => <PageComponent
                    key={page.id} page={page} chapterId={chapter.id} handlers={pageHandlers}
                    isActionLoading={isActionLoading} />) : <p className="text-center py-8 text-
                    gray-500">This chapter has no pages yet.</p>}
                </div>
            )}
        </div>
    );
};

export default ChapterComponent;


// ===== Robot_1.tsx =====

import React from 'react';
import { motion, AnimatePresence } from 'framer-motion';
import type { RobotState } from '../../../types';
import { PaintBrushIcon } from '../../icons.tsx';

interface RobotProps {
    robotState: RobotState;
    mousePosition: { x: number; y: number };
}

const Robot: React.FC<RobotProps> = ({ robotState, mousePosition }) => {
    const { x, y } = mousePosition;
    const centerX = window.innerWidth / 2;
    const angle = Math.atan2(y - (window.innerHeight - 80), x - centerX) * (180 /
    Math.PI) + 90;
    const clampedAngle = Math.max(-45, Math.min(45, angle));

    const variants = {
        idle: { y: 0 },
        thinking: { y: -5, scale: 1.05 },
        writing: { y: 2, rotate: [0, 1, -1, 0] },
        illustrating: { y: 2, rotate: [0, 2, -2, 0] },
    };

    return (
        <motion.div className="fixed bottom-0 left-1/2 -translate-x-1/2 z-50 pointer-
        events-none" style={{ x: x - window.innerWidth / 2, y: y - window.innerHeight }}
        initial={{ opacity: 0, y: 100 }} animate={{ opacity: 1, y: 0 }} transition={{
        duration: 0.5 }}>
            <motion.div animate={robotState} variants={variants} transition={{ y: { repeat:
            Infinity, repeatType: 'reverse', duration: 1.5, ease: 'easeInOut' }, rotate: {
            repeat: Infinity, repeatType: 'reverse', duration: 0.5 } }}>
                <svg width="120" height="120" viewBox="0 0 100 100">
```

```
                    <ellipse cx="50" cy="95" rx="30" ry="5" fill="black" opacity="0.2" />
                    <motion.g animate={{ rotate: clampedAngle }} style={{ transformOrigin: '50px
                    70px' }}>
                        <rect x="30" y="40" width="40" height="40" rx="10" fill="#4B5563" />
                        <rect x="25" y="45" width="50" height="30" rx="10" fill="#6B7280" />
                        <motion.circle cx="50" cy="55" r="12" fill="#1F2937" />
                        <motion.circle cx="50" cy="55" r="8" fill="#A78BFA" animate={{ scale: robotState
                        === 'thinking' ? [1, 1.2, 1] : 1 }} transition={{ repeat: Infinity, duration: 1
                        }} />
                        <circle cx="52" cy="53" r="3" fill="white" />
                        <line x1="50" y1="40" x2="50" y2="25" stroke="#4B5563" strokeWidth="3" />
                        <motion.circle cx="50" cy="25" r="5" fill="#A78BFA" animate={{ scale: robotState
                        === 'thinking' ? [1, 1.5, 1] : 1, boxShadow: robotState === 'thinking' ? '0 0
                        15px #A78BFA' : '0 0 0px #A78BFA' }} transition={{ repeat: Infinity, duration:
                        0.8 }} />
                    </motion.g>
                    <AnimatePresence>
                    {robotState === 'writing' && (
                        <motion.g key="writing" initial={{ y: 20, opacity: 0 }} animate={{ y: 0,
                        opacity: 1 }} exit={{ y: 20, opacity: 0 }}>
                            <rect x="20" y="80" width="60" height="15" rx="3" fill="#374151" />
                            <motion.circle cx="35" cy="80" r="5" fill="#6B7280" animate={{ y: [0, -3, 0]}}
                            transition={{ repeat: Infinity, duration: 0.3, delay: 0.1 }} />
                            <motion.circle cx="65" cy="80" r="5" fill="#6B7280" animate={{ y: [0, -3, 0]}}
                            transition={{ repeat: Infinity, duration: 0.3 }} />
                        </motion.g>
                    )}
                     {robotState === 'illustrating' && (
                        <motion.g key="illustrating" initial={{ y: 10, opacity: 0, rotate: -20, x: 10 }}
                        animate={{ y: 0, opacity: 1, rotate: 10, x: 0 }} exit={{ y: 10, opacity: 0,
                        rotate: -20 }} transition={{ type: 'spring', stiffness: 300, damping: 20 }}>
                            <circle cx="75" cy="75" r="6" fill="#6B7280" />
                            <g transform="translate(65, 55) rotate(45)"><PaintBrushIcon className="w-8 h-8
                            text-purple-400" /></g>
                        </motion.g>
                    )}
                    </AnimatePresence>
                </svg>
            </motion.div>
        </motion.div>
    );
};

export default Robot;


// ===== StoryViewer_1.tsx =====

import React, { useState, useEffect, useRef } from 'react';
import type { StoryDocument, PageHandlers, EditorActions } from
'../../../types';
import ChapterComponent from './PlanDisplay';
import MagazinePreview from './MagazinePreview';
import { PlusIcon, DocumentPlusIcon, WandIcon, ChatBubbleBottomCenterTextIcon,
SparklesIcon, PlayIcon, StopIcon } from '../../icons.tsx';

interface StoryEditorProps {
    doc: StoryDocument;
    setDoc: React.Dispatch<React.SetStateAction<StoryDocument | null>>;
    pageHandlers: PageHandlers;
    editorActions: EditorActions;
    generationStatus: { active: boolean, completed: number, total: number };
```

```
    onStopGeneration: () => void;
    onContinueGeneration: () => void;
}

const StoryEditor: React.FC<StoryEditorProps> = ({ doc, setDoc, pageHandlers,
editorActions, generationStatus, onStopGeneration, onContinueGeneration }) => {
    const fileInputRef = useRef<HTMLInputElement>(null);
    const [activeChapterId, setActiveChapterId] = useState<string | null>(null);

    useEffect(() => {
        if (doc && doc.chapters.length > 0 && !activeChapterId) {
            setActiveChapterId(doc.chapters[0].id);
        }
    }, [doc, activeChapterId]);

    const activeChapter = doc.chapters.find(c => c.id === activeChapterId);

    return (
        <div className="grid grid-cols-1 xl:grid-cols-12 gap-6 h-full p-4">
            <aside className="xl:col-span-3 bg-gray-900/60 p-4 rounded-lg border border-
            gray-700 flex flex-col">
                <h3 className="font-bold mb-3 text-violet-300">Table of Contents</h3>
                <div className="flex-grow overflow-y-auto space-y-1 pr-2 -mr-2">
                    {doc.chapters.map((chapter, index) => (
                        <button key={chapter.id} onClick={() => setActiveChapterId(chapter.id)}
                        className={`block w-full text-left p-2 rounded-md transition-colors text-sm
                        ${activeChapterId === chapter.id ? 'bg-violet-900/50 text-white' : 'hover:bg-
                        gray-700/50 text-gray-300'}`}>
                            <span className="font-semibold text-gray-500 mr-2">{index + 1}.</span>
                            <span className="font-semibold">{chapter.title}</span>
                             <p className="text-xs text-gray-400 pl-6 truncate">{chapter.summary}</p>
                        </button>
                    ))}
                </div>
                {generationStatus.total > 0 && generationStatus.completed <
                generationStatus.total && (
                    <div className="mt-4 p-3 bg-gray-800 rounded-lg border border-gray-700">
                        <div className="flex justify-between items-center mb-2">
                            <p className="text-sm font-medium text-gray-300">{generationStatus.active ?
                            'Generating Chapters...' : 'Generation Paused'}</p>
                            <p className="text-sm text-gray-400">{generationStatus.completed} /
                            {generationStatus.total}</p>
                        </div>
                        <div className="w-full bg-gray-600 rounded-full h-2 mb-2">
                            <div className="bg-violet-500 h-2 rounded-full" style={{ width:
                            `${(generationStatus.completed / generationStatus.total) * 100}%` }}></div>
                        </div>
                        {generationStatus.active ? <button onClick={onStopGeneration} className="w-full
                        flex items-center justify-center gap-2 text-sm p-2 bg-red-600/50 hover:bg-
                        red-600/80 rounded-md"><StopIcon className="w-4 h-4" /> Stop</button> : <button
                        onClick={onContinueGeneration} className="w-full flex items-center justify-
                        center gap-2 text-sm p-2 bg-green-600/50 hover:bg-green-600/80 rounded-
                        md"><PlayIcon className="w-4 h-4" /> Continue</button>}
                    </div>
                )}
                 <div className="mt-4 space-y-2 border-t border-gray-700 pt-4">
                    <div className="relative group">
                        <button className="w-full flex items-center justify-center gap-2 text-sm p-2 bg-
                        violet-600/80 hover:bg-violet-600 rounded-md"><WandIcon className="w-4 h-4"/> AI
                        Assist</button>
                        <div className="absolute bottom-full mb-2 w-full bg-gray-800 border border-
                        gray-700 rounded-lg p-2 space-y-2 z-10 opacity-0 group-hover:opacity-100
```

```
                        transition-opacity pointer-events-none group-hover:pointer-events-auto">
                            <button onClick={editorActions.onSuggestTitles} className="w-full text-left
                            text-sm p-2 hover:bg-gray-700 rounded-md flex items-center gap-2"><SparklesIcon
                            className="w-4 h-4"/> Improve Titles</button>
                            <button onClick={editorActions.onSummarizeChapters} className="w-full text-left
                            text-sm p-2 hover:bg-gray-700 rounded-md flex items-center
                            gap-2"><ChatBubbleBottomCenterTextIcon className="w-4 h-4"/> Update
                            Summaries</button>
                        </div>
                    </div>
                </div>
            </aside>
            <div className="xl:col-span-5 overflow-y-auto pr-2 -mr-2">
                <button onClick={editorActions.onAutoDraftAll} className="w-full mb-4 flex
                items-center justify-center gap-2 p-3 bg-violet-800 hover:bg-violet-700 rounded-
                lg font-bold"><WandIcon className="w-5 h-5"/> AI, Write Full Draft</button>
                {activeChapter ? <ChapterComponent chapter={activeChapter}
                pageHandlers={pageHandlers} isActionLoading={false} /> : <p className="flex
                items-center justify-center h-full text-gray-500">Select a chapter to begin
                editing.</p>}
            </div>
            <aside className="xl:col-span-4 bg-gray-900/60 p-4 rounded-lg border border-
            gray-700 overflow-y-auto">
                <MagazinePreview doc={doc} />
            </aside>
        </div>
    );
};

export default StoryEditor;
```