```
.sidebar-item {
    cursor: default;
}

.sidebar-item.sidebar-item-file {
    cursor: pointer;
}
```

Sidebar.tsx

```
import React from "react";
import fileLinks from "../file_links.json";
import "./Sidebar.css";

interface SidebarProps {
    onFileSelect: (fileContent: string) => void;
}

export const Sidebar: React.FC<SidebarProps> = ({ onFileSelect }) => {
    const handleFileClick = (file: { name: string; type: string }) => {
        if (file.type === "file" && file.name.endsWith(".md")) {
            fetch(`/public/${file.name}`)
                .then((response) => response.text())
                .then((content) => onFileSelect(content))
                .catch((error) => console.error("Error loading file:", error));
        }
    };

    return (
        <aside className="sidebar">
            <h3 className="sidebar-title">Explorer</h3>
            <ul className="sidebar-list">
                {fileLinks.files.map((entry: { name: string; type: string }, i: number) => (
                    <li
                        key={i}
                        className={`sidebar-item sidebar-item-${entry.type}`}
                        onClick={() => handleFileClick(entry)}
                        style={{ cursor: entry.type === "file" ? "pointer" : "default" }}
                    >
                        {entry.type === "folder" ? "■" : "■"} {entry.name}
                    </li>
                ))}
            </ul>
        </aside>
    );
};
```

SuggestionsDisplay.tsx

```
import React, { useState } from 'react';
import ReactMarkdown from 'https://esm.sh/react-markdown@9';
import remarkGfm from 'https://esm.sh/remark-gfm@4';
import { Spinner, WandSparklesIcon, PackageIcon, HistoryIcon } from './Icon';
import { AIGeneratedCode, isAIGeneratedCode, Dependency, AIAction } from '../services/geminiService';
import { Revision } from '../App';
import DiffViewer from './DiffViewer';
import DependencyViewer from './DependencyViewer';
import GeneratedTestViewer from './GeneratedTestViewer';
import RevisionHistoryViewer from './RevisionHistoryViewer';

type Tab = 'suggestions' | 'dependencies' | 'history';

interface SuggestionsDisplayProps {
  suggestions: string | AIGeneratedCode | null;
  originalCode: string;
  isLoading: boolean;
  error: string | null;
  onAcceptChange: (newCode: string) => void;
  onRejectChange: () => void;
  lastAction: AIAction | null;
```

```
  dependencies: Dependency[] | null;
  isAnalyzingDependencies: boolean;
  onGenerateRequirements: () => void;
  requirementsFile: string | null;
  revisionHistory: Revision[];
  currentCode: string;
  onRestoreRevision: (code: string) => void;
}

const SuggestionsDisplay: React.FC<SuggestionsDisplayProps> = (props) => {
  const [activeTab, setActiveTab] = useState<Tab>('suggestions');

  const renderSuggestionsContent = () => {
    if (props.isLoading) {
      return (
        <div className="flex flex-col items-center justify-center h-full text-text-secondary">
          <Spinner className="h-12 w-12 animate-spin mb-4 text-secondary" />
          <p className="text-lg font-medium">Analyzing code...</p>
          <p className="text-sm text-gray-500">The AI is thinking. This may take a moment.</p>
        </div>
      );
    }

    if (props.error && activeTab === 'suggestions') {
      return (
        <div className="flex items-center justify-center h-full text-red-400">
          <div className="text-center p-4 bg-red-900/20 rounded-lg">
            <p className="font-bold text-lg">An Error Occurred</p>
            <p>{props.error}</p>
          </div>
        </div>
      );
    }

    if (!props.suggestions) {
      return (
        <div className="flex flex-col items-center justify-center h-full text-text-secondary p-8 text-center">
            <h3 className="text-2xl font-bold mb-2">Ready for Analysis</h3>
            <p className="max-w-md">Select a piece of code and click the ■ wand to get started, or click "Analyze & Ref
        </div>
      );
    }

    if (isAIGeneratedCode(props.suggestions)) {
        const suggestions = props.suggestions;

        if (props.lastAction === 'generate_tests') {
            return <GeneratedTestViewer explanation={suggestions.explanation} code={suggestions.code} />
        }

        return (
            <div>
                 <div className="prose prose-invert max-w-none mb-6">
                    <ReactMarkdown children={suggestions.explanation} remarkPlugins={[remarkGfm]} />
                 </div>
                 <h3 className="text-lg font-semibold text-text-secondary mb-2">Suggested Change:</h3>
                 <div className="border border-surface rounded-lg overflow-hidden">
                    <DiffViewer
                        oldCode={props.originalCode}
                        newCode={suggestions.code}
                    />
                 </div>
                 <div className="flex justify-end space-x-3 mt-4">
                    <button
                        onClick={props.onRejectChange}
                        className="px-4 py-2 text-sm font-medium text-text-secondary rounded-md hover:bg-surface transi
                    >
                        Reject
                    </button>
                    <button
```

```
                        onClick={() => props.onAcceptChange(suggestions.code)}
                        className="px-4 py-2 text-sm font-medium text-white bg-secondary rounded-md hover:opacity-80 tr
                    >
                        Accept Change
                    </button>
                </div>
            </div>
        )
    }

    return (
      <ReactMarkdown
        children={props.suggestions}
        remarkPlugins={[remarkGfm]}
        components={{
          h1: ({node, ...props}) => <h1 className="text-3xl font-bold mt-6 mb-3 border-b border-gray-600 pb-2" {...prop
          h2: ({node, ...props}) => <h2 className="text-2xl font-semibold mt-5 mb-2" {...props} />,
          h3: ({node, ...props}) => <h3 className="text-xl font-semibold mt-4 mb-2" {...props} />,
          p: ({node, ...props}) => <p className="mb-4 leading-relaxed" {...props} />,
          ul: ({node, ...props}) => <ul className="list-disc list-inside mb-4 pl-4 space-y-2" {...props} />,
          ol: ({node, ...props}) => <ol className="list-decimal list-inside mb-4 pl-4 space-y-2" {...props} />,
          li: ({node, ...props}) => <li className="mb-1" {...props} />,
          code: ({node, inline, className, children, ...props}) => {
            const match = /language-(\w+)/.exec(className || '')
            return !inline ? (
              <pre className="bg-gray-900/70 p-4 rounded-lg my-4 overflow-x-auto text-sm font-mono shadow-inner">
                <code className={className} {...props}>
                  {children}
                </code>
              </pre>
            ) : (
              <code className="bg-primary/50 text-blue-200 px-1.5 py-1 rounded-md font-mono text-sm" {...props}>
                {children}
              </code>
            )
          },
          blockquote: ({node, ...props}) => <blockquote className="border-l-4 border-secondary pl-4 italic my-4 text-te
        }}
      />
    );
};

const TabButton: React.FC<{tab: Tab, label: string, icon: React.ElementType}> = ({ tab, label, icon: Icon }) => {
    const isActive = activeTab === tab;
    return (
        <button
            onClick={() => setActiveTab(tab)}
            className={`flex items-center space-x-2 px-4 py-2 text-sm font-medium border-b-2 transition-colors ${
                isActive
                    ? 'border-secondary text-text-primary'
                    : 'border-transparent text-text-secondary hover:text-text-primary'
            }`}
        >
            <Icon className="w-5 h-5" />
            <span>{label}</span>
        </button>
    );
}

return (
    <div className="bg-surface rounded-lg shadow-lg flex flex-col overflow-hidden h-full">
      <div className="flex items-center border-b border-gray-700 px-2">
        <TabButton tab="suggestions" label="AI Suggestions" icon={WandSparklesIcon} />
        <TabButton tab="dependencies" label="Dependencies" icon={PackageIcon} />
        <TabButton tab="history" label="History" icon={HistoryIcon} />
      </div>
      <div className="p-4 lg:p-6 overflow-auto flex-grow">
        {activeTab === 'suggestions' && renderSuggestionsContent()}
        {activeTab === 'dependencies' && (
            <DependencyViewer
```

```
                dependencies={props.dependencies}
                isLoading={props.isAnalyzingDependencies}
                error={props.error && activeTab === 'dependencies' ? props.error : null}
                onGenerate={props.onGenerateRequirements}
                requirementsFile={props.requirementsFile}
            />
        )}
        {activeTab === 'history' && (
            <RevisionHistoryViewer
                revisions={props.revisionHistory}
                currentCode={props.currentCode}
                onRestore={props.onRestoreRevision}
            />
        )}
      </div>
    </div>
  );
};

export default SuggestionsDisplay;
```

themes.ts

```
export type Theme = {
  name: string;
  colors: {
    '--color-primary': string;
    '--color-secondary': string;
    '--color-background': string;
    '--color-surface': string;
    '--color-text-primary': string;
    '--color-text-secondary': string;
  };
};

export const themes: Theme[] = [
  {
    name: 'Midnight Dusk',
    colors: {
      '--color-primary': '#1e3a8a',
      '--color-secondary': '#10b981',
      '--color-background': '#111827',
      '--color-surface': '#1f2937',
      '--color-text-primary': '#f9fafb',
      '--color-text-secondary': '#9ca3af',
    },
  },
  {
    name: 'Arctic Light',
    colors: {
      '--color-primary': '#2563eb',
      '--color-secondary': '#db2777',
      '--color-background': '#f9fafb',
      '--color-surface': '#ffffff',
      '--color-text-primary': '#1f2937',
      '--color-text-secondary': '#6b7280',
    },
  },
  {
    name: 'Solarized Flare',
    colors: {
      '--color-primary': '#268bd2',
      '--color-secondary': '#cb4b16',
      '--color-background': '#002b36',
      '--color-surface': '#073642',
      '--color-text-primary': '#93a1a1',
      '--color-text-secondary': '#586e75',
    },
  },
  {
```

```
    name: 'Matrix Green',
    colors: {
        '--color-primary': '#008F11',
        '--color-secondary': '#33FF44',
        '--color-background': '#000000',
        '--color-surface': '#0D0D0D',
        '--color-text-primary': '#33FF44',
        '--color-text-secondary': '#00A82B',
    },
  },
  {
    name: 'Corporate Blue',
    colors: {
      '--color-primary': '#9333ea',
      '--color-secondary': '#db2777',
      '--color-background': '#1e1b4b',
      '--color-surface': '#312e81',
      '--color-text-primary': '#67e8f9',
      '--color-text-secondary': '#a78bfa',
    },
  },
  {
    name: 'Minimalist Gray',
    colors: {
      '--color-primary': '#c2410c',
      '--color-secondary': '#f59e0b',
      '--color-background': '#422006',
      '--color-surface': '#78350f',
      '--color-text-primary': '#fef3c7',
      '--color-text-secondary': '#fdba74',
    },
  },
  {
    name: 'Cosmic Nightmare',
    colors: {
      '--color-primary': '#8b5cf6',
      '--color-secondary': '#f472b6',
      '--color-background': '#282a36',
      '--color-surface': '#44475a',
      '--color-text-primary': '#f8f8f2',
      '--color-text-secondary': '#bd93f9',
    },
  },
  {
    name: 'Synthwave Sunset Overdrive',
    colors: {
      '--color-primary': '#83a598',
      '--color-secondary': '#fe8019',
      '--color-background': '#282828',
      '--color-surface': '#3c3836',
      '--color-text-primary': '#ebdbb2',
      '--color-text-secondary': '#a89984',
    },
  },
];
```

SettingsContext.tsx

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

export type AIProvider = 'gemini' | 'local';
export type QualitySetting = 'balanced' | 'max_quality';

export interface Settings {
  provider: AIProvider;
  apiKey: string | null;
  localAIUrl: string | null;
  qualitySetting: QualitySetting;
}
```

```
interface SettingsContextType {
  settings: Settings;
  setSettings: (settings: Settings) => void;
}

const defaultSettings: Settings = {
  provider: 'gemini',
  apiKey: process.env.API_KEY || null, // Pre-fill with env var if available
  localAIUrl: null,
  qualitySetting: 'max_quality',
};

export const SettingsContext = createContext<SettingsContextType>({
  settings: defaultSettings,
  setSettings: () => {},
});

interface SettingsProviderProps {
  children: ReactNode;
}

export const SettingsProvider: React.FC<SettingsProviderProps> = ({ children }) => {
  const [settings, setSettingsState] = useState<Settings>(() => {
    try {
      const savedSettings = localStorage.getItem('ai_refactor_settings');
      if (savedSettings) {
        // Merge saved settings with defaults to handle new fields
        return { ...defaultSettings, ...JSON.parse(savedSettings) };
      }
    } catch (error) {
      console.error("Failed to parse settings from localStorage", error);
    }
    return defaultSettings;
  });

  const setSettings = (newSettings: Settings) => {
    try {
      localStorage.setItem('ai_refactor_settings', JSON.stringify(newSettings));
      setSettingsState(newSettings);
    } catch (error) {
      console.error("Failed to save settings to localStorage", error);
    }
  };

  return (
    <SettingsContext.Provider value={{ settings, setSettings }}>
      {children}
    </SettingsContext.Provider>
  );
};
```

ThemeContext.tsx

```
import React, { createContext, useState, useEffect, ReactNode } from 'react';
import { themes, Theme } from '../constants/themes';

interface ThemeContextType {
  theme: Theme;
  setTheme: (themeName: string) => void;
}

export const ThemeContext = createContext<ThemeContextType>({
  theme: themes[0],
  setTheme: () => {},
});

interface ThemeProviderProps {
  children: ReactNode;
}
```

```
export const ThemeProvider: React.FC<ThemeProviderProps> = ({ children }) => {
  const [theme, setThemeState] = useState<Theme>(() => {
    try {
      const savedThemeName = localStorage.getItem('ai_refactor_theme');
      const savedTheme = themes.find(t => t.name === savedThemeName);
      return savedTheme || themes[0];
    } catch (error) {
      console.error("Failed to parse theme from localStorage", error);
      return themes[0];
    }
  });

  const setTheme = (themeName: string) => {
    const newTheme = themes.find(t => t.name === themeName);
    if (newTheme) {
        try {
            localStorage.setItem('ai_refactor_theme', newTheme.name);
            setThemeState(newTheme);
        } catch (error) {
            console.error("Failed to save theme to localStorage", error);
        }
    }
  };

  return (
    <ThemeContext.Provider value={{ theme, setTheme }}>
      {children}
    </ThemeContext.Provider>
  );
};
```

execution.js

```
// This is a web worker for code execution.

// Suppress console.log for a cleaner worker environment
self.console.log = () => {};
self.console.warn = () => {};
self.console.error = () => {};


let pyodide = null;

async function initPyodide() {
  if (pyodide) {
    return;
  }

  self.importScripts("https://cdn.jsdelivr.net/pyodide/v0.25.1/full/pyodide.js");
  pyodide = await self.loadPyodide();
  console.log("Pyodide for execution initialized");
  self.postMessage({ type: 'ready' });
}

const pyodideReadyPromise = initPyodide();

self.onmessage = async (event) => {
  await pyodideReadyPromise;
  const { code } = event.data;

  try {
    // Redirect stdout and stderr
    pyodide.setStdout({
      batched: (msg) => {
        self.postMessage({ type: 'stdout', data: msg });
      }
    });
    pyodide.setStderr({
      batched: (msg) => {
        self.postMessage({ type: 'stderr', data: msg });
```

```
      }
    });

    await pyodide.runPythonAsync(code);

  } catch (e) {
    self.postMessage({ type: 'error', data: e.message });
  } finally {
    // Restore default stdout/stderr
    pyodide.setStdout({});
    pyodide.setStderr({});
    self.postMessage({ type: 'done' });
  }
};
```

linter.js

```
// This is a web worker, so we can't use ES6 modules.
// We'll use importScripts to load Pyodide.

// Suppress console.log for a cleaner worker environment
self.console.log = () => {};
self.console.warn = () => {};
self.console.error = () => {};


let pyodide = null;
let ruff = null;

async function initPyodide() {
  if (pyodide) {
    return;
  }

  self.importScripts("https://cdn.jsdelivr.net/pyodide/v0.25.1/full/pyodide.js");
  pyodide = await self.loadPyodide();
  await pyodide.loadPackage("micropip");
  const micropip = pyodide.pyimport("micropip");
  await micropip.install("ruff");
  ruff = pyodide.pyimport("ruff");
  console.log("Pyodide and Ruff initialized");
  self.postMessage({ type: 'ready' });
}


self.onmessage = async (event) => {
  const { code } = event.data;

  if (!pyodide || !ruff) {
    await initPyodide();
  }

  if (ruff) {
    try {
      const diagnostics = ruff.check(code, "code.py");
      const errors = diagnostics.map(d => ({
        line: d.location.row,
        column: d.location.column,
        end_line: d.end_location.row,
        end_column: d.end_location.column,
        message: d.message,
        code: d.kind,
      }));
      self.postMessage({ type: 'lintResults', errors });
    } catch (e) {
      // In case ruff itself throws an error on invalid syntax
       self.postMessage({ type: 'lintResults', errors: [] });
    }
  }
};
```

```
initPyodide().catch(e => console.error("Worker initialization failed:", e));
```

geminiService.ts

```
import { GoogleGenAI, Type } from "@google/genai";
import { Settings } from "../contexts/SettingsContext";

export type AIAction = 'explain' | 'refactor' | 'add_docs' | 'find_bugs' | 'generate_tests' | 'explain_error';

// This is the structured response we expect for code modifications.
export interface AIGeneratedCode {
  explanation: string;
  code: string;
}

export interface Dependency {
    module: string;
    packageName: string;
    justification: string;
}

// A type guard to check if the response is a structured code object.
export const isAIGeneratedCode = (response: any): response is AIGeneratedCode => {
    return typeof response === 'object' && response !== null && 'explanation' in response && 'code' in response;
}

const refactorResponseSchema = {
    type: Type.OBJECT,
    properties: {
        explanation: {
            type: Type.STRING,
            description: "A detailed explanation of the code changes, formatted in Markdown."
        },
        code: {
            type: Type.STRING,
            description: "The complete, refactored, or documented code snippet."
        }
    },
    required: ["explanation", "code"]
};

const dependencyResponseSchema = {
    type: Type.OBJECT,
    properties: {
        dependencies: {
            type: Type.ARRAY,
            description: "A list of identified third-party Python dependencies.",
            items: {
                type: Type.OBJECT,
                properties: {
                    module: {
                        type: Type.STRING,
                        description: "The name of the module as used in the import statement (e.g., 'yaml', 'github')."
                    },
                    packageName: {
                        type: Type.STRING,
                        description: "The correct package name for installation via pip (e.g., 'PyYAML', 'PyGithub')."
                    },
                    justification: {
                        type: Type.STRING,
                        description: "A brief explanation of what the library is used for in this script."
                    }
                },
                required: ["module", "packageName", "justification"]
            }
        }
    },
    required: ["dependencies"]
};
```

```
const getPromptForAction = (code: string, action: AIAction): [string, boolean, object | undefined] => {
  switch (action) {
    case 'explain':
      return [`As an expert Python developer, explain the following code snippet. Describe its purpose, how it works, a
      \n\n\`\`\`python\n${code}\n\`\`\`\`, false, undefined];
    case 'refactor':
      return [`As a senior software architect, refactor the following Python code snippet to improve its readability, e
      \n\n\`\`\`python\n${code}\n\`\`\`\`, true, refactorResponseSchema];
    case 'add_docs':
      return [`As an expert Python developer, add comprehensive PEP 257 docstrings and PEP 484 type hints to the follow
      \n\n\`\`\`python\n${code}\n\`\`\`\`, true, refactorResponseSchema];
    case 'find_bugs':
      return [`As a senior quality assurance engineer, analyze the following Python code snippet for potential bugs, se
      \n\n\`\`\`python\n${code}\n\`\`\`\`, false, undefined];
    case 'generate_tests':
      return [`As a Senior QA Engineer, write a comprehensive suite of unit tests for the following Python code snippet
      - Your response must be a single, complete, and runnable Python script.
      - It should include all necessary imports (e.g., \`import unittest\`).
      - If the code snippet is a function, create a test class that inherits from \`unittest.TestCase\` and write test
      - Include a brief Markdown explanation of the testing strategy and what cases are covered.
      - Return only the JSON object as specified in the schema.
      \n\n\`\`\`python\n${code}\n\`\`\`\`, true, refactorResponseSchema];
    default:
      // This should not be reached if all actions are handled, but provides a fallback.
      if (action === 'explain_error') {
        throw new Error("explain_error should be handled by its own dedicated function.");
      }
      throw new Error("Invalid AI action");
  }
};

const FULL_ANALYSIS_PROMPT = `
    As an expert Python developer and senior software architect, please provide a comprehensive review of the following

    Focus on the following areas:
    1.  **Code Quality & Best Practices:** Check for adherence to PEP 8, clarity, readability, and overall maintainabil
    2.  **Potential Bugs & Edge Cases:** Identify any logical errors, potential race conditions, or unhandled exception
    3.  **Refactoring Suggestions:** Propose specific, actionable changes to improve the architecture, reduce complexit
    4.  **Dependency Management:** Comment on how external libraries are imported and handled. Is there a better way? A
    5.  **Security:** Point out any potential security vulnerabilities, such as improper handling of secrets (like API
    6.  **Configuration and Environment Management:** Evaluate the approach to loading configuration and environment va

    Please present your feedback in a well-structured Markdown format. Use clear headings, bullet points, and code bloc
`;

const DEPENDENCY_ANALYSIS_PROMPT = `
    As a Python packaging expert, analyze the provided Python script to identify all third-party dependencies.
    - Examine all \`import\` and \`from ... import\` statements.
    - Distinguish between standard library modules (e.g., 'os', 'sys', 'logging') and third-party libraries.
    - For each third-party library, determine the correct package name for installation via pip. For example, the modul
    - Provide a brief justification for why each library is used in the script.
    - Return the result as a single JSON object matching the provided schema, with a root key "dependencies" containing
`;

const REQUIREMENTS_GENERATION_PROMPT = `
    Based on the following list of Python packages, generate the content for a standard \`requirements.txt\` file.
    List one package per line, using the format \`packageName\`. Do not add comments or any extra text.
`;

const EXPLAIN_ERROR_PROMPT = `
    As an expert Python debugger, the user's script has failed. Your task is to analyze the user's code and the resulti

    1.  **Explain the Error:** In simple terms, explain what the error message means and why it occurred in the context
    2.  **Provide the Fix:** Provide the complete, corrected Python script. The fix should be the minimum necessary cha

    Return your response as a single JSON object matching the provided schema, with an 'explanation' in Markdown and th
`;
```

```
const callGeminiAPI = async (prompt: string, settings: Settings, useJson: boolean, schema?: object): Promise<string | o
  const apiKey = settings.apiKey;
  if (!apiKey) {
    throw new Error("Gemini API key is not configured.");
  }
  const ai = new GoogleGenAI({ apiKey });

  const modelName = settings.qualitySetting === 'balanced' ? 'gemini-2.5-flash' : 'gemini-2.5-pro';

  const response = await ai.models.generateContent({
    model: modelName,
    contents: prompt,
    config: useJson ? {
        responseMimeType: "application/json",
        responseSchema: schema,
    } : {},
  });

  const text = response.text;

  if (useJson) {
      try {
          return JSON.parse(text);
      } catch (e) {
          console.error("Failed to parse JSON response:", text);
          throw new Error("The API returned an unexpected response format.");
      }
  }

  return text;
};

const callLocalAI = async (prompt: string, url: string, useJson: boolean, schema?: object): Promise<string | object> =>
    if (!url) {
        throw new Error("Local AI URL is not configured.");
    }
    const response = await fetch(url, {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({
            prompt: prompt,
            json_schema: useJson ? schema : undefined,
        }),
    });
    if (!response.ok) {
        throw new Error(`Local AI request failed with status ${response.status}`);
    }
    const data = await response.json();
    const result = data.text || data.choices?.[0]?.text || '';

    if (useJson) {
      try {
          return JSON.parse(result);
      } catch (e) {
           console.error("Failed to parse JSON response from local AI:", result);
           throw new Error("The local AI returned an unexpected response format.");
      }
    }
    return result;
};

const generateResponse = async (prompt: string, settings: Settings, useJson: boolean, schema?: object): Promise<string
  try {
    if (settings.provider === 'gemini') {
      return await callGeminiAPI(prompt, settings, useJson, schema);
    } else if (settings.provider === 'local') {
      return await callLocalAI(prompt, settings.localAIUrl ?? '', useJson, schema);
    } else {
      throw new Error("Invalid AI provider configured.");
    }
```

```
  } catch (error) {
    console.error(`Error calling ${settings.provider} API:`, error);
    if (error instanceof Error) {
        throw new Error(`The request to the ${settings.provider} service failed: ${error.message}`);
    }
    throw new Error(`The request to the ${settings.provider} service failed.`);
  }
};

export const analyzeCode = async (code: string, settings: Settings): Promise<string> => {
  const prompt = `${FULL_ANALYSIS_PROMPT}
    Here is the Python code to analyze:
    ---
    \`\`\`python
    ${code}
    \`\`\`
    ---
  `;
  const result = await generateResponse(prompt, settings, false);
  return result as string;
};

export const analyzeSelection = async (code: string, action: AIAction, settings: Settings): Promise<string | AIGenerate
    const [prompt, useJson, schema] = getPromptForAction(code, action);
    const result = await generateResponse(prompt, settings, useJson, schema);
    if (useJson && isAIGeneratedCode(result)) {
        return result;
    }
    return result as string;
};

export const analyzeDependencies = async (code: string, settings: Settings): Promise<Dependency[]> => {
    const prompt = `${DEPENDENCY_ANALYSIS_PROMPT}
    ---
    \`\`\`python
    ${code}
    \`\`\`
    ---
  `;
  const result = await generateResponse(prompt, settings, true, dependencyResponseSchema) as { dependencies: Dependency
  return result.dependencies || [];
}

export const generateRequirementsFile = async (dependencies: Dependency[], settings: Settings): Promise<string> => {
    const packageList = dependencies.map(d => d.packageName).join('\n');
    const prompt = `${REQUIREMENTS_GENERATION_PROMPT}\n${packageList}`;
    const result = await generateResponse(prompt, settings, false);
    return result as string;
}

export const explainError = async (code: string, traceback: string, settings: Settings): Promise<AIGeneratedCode> => {
    const prompt = `${EXPLAIN_ERROR_PROMPT}

    ### User's Code:
    \`\`\`python
    ${code}
    \`\`\`

    ### Traceback:
    \`\`\`
    ${traceback}
    \`\`\`
    `;
    const result = await generateResponse(prompt, settings, true, refactorResponseSchema);
    if (isAIGeneratedCode(result)) {
        return result;
    }
    throw new Error("The API returned an unexpected format for the error explanation.");
};
```

```
// Simple chat helper for free-form prompts
export const chatWithAI = async (message: string, settings: Settings): Promise<string> => {
  const prompt = message;
  const result = await generateResponse(prompt, settings, false);
  return result as string;
};
```