**Code-Mitra VS Code Extension (with Python Backend) - Detailed Technical Documentation**

**English Version**

**1. Architectural Overview**

This version of the project follows a **client-server architecture**. This is a robust and scalable design where tasks are separated based on their function. Think of it like a restaurant: the frontend is the waiter who takes your order, and the backend is the kitchen that prepares the food.

* **The Client (VS Code Extension - TypeScript):** This is the "waiter". It's the frontend that runs directly inside the user's editor. Its main responsibilities are:
  + Providing the User Interface (UI) in the sidebar.
  + Detecting user actions, like saving a file or clicking a button.
  + Sending the file content (the "order") to the backend server for processing.
  + Receiving the final results (the "food") from the server and displaying them beautifully to the user.
* **The Server (Backend - Python with Flask):** This is the "kitchen". It's the backend brain of the application that runs as a separate, local web server. Its main responsibilities are:
  + Handling all logic-intensive tasks to keep the extension fast and responsive.
  + Running pylint for code analysis.
  + Communicating with the external Google Gemini API to get AI insights.
  + Sending the final, processed results back to the VS Code extension.

**Why this architecture?** This separation is excellent because it lets each part do what it's best at. TypeScript is fantastic for building user interfaces and integrating with VS Code, while Python, with its rich libraries for data science and AI, is perfect for handling the heavy data processing and API communication.

**2. Features in Detail**

* **AI-Powered Code Explanation:** When a Python file is saved, the extension sends the code to the AI, which returns a simple explanation of the logic, alternative methods, and potential optimizations.
* **Intelligent Error Solutions:** The extension uses the pylint tool to find errors. It then sends both the code and the errors to the AI, which explains the errors and provides a corrected code snippet.
* **Interactive Q&A:** The user can ask a question about the currently open file. The extension sends the question along with the file's content as context to the AI to get a precise answer.
* **Markdown Task Planner:** If the user saves a Markdown file with a task description (e.g., "Create a calculator app"), the AI breaks it down into actionable steps.
* **Native UI:** The interface is built with the official @vscode/webview-ui-toolkit to ensure it looks and feels like a natural part of VS Code.

**3. File-wise Breakdown**

**package.json (The Manifest)**

* **Role:** This is the manifest file that tells VS Code everything about the extension, including its name, activation events, and dependencies.
* **Key Sections Explained:**
  + activationEvents: We use "onView:codeMitraView" to ensure the extension only activates when the user clicks its icon. This is a crucial performance optimization that prevents our extension from slowing down VS Code's startup time.
  + contributes: This section defines the extension's UI contributions, like the icon in the Activity Bar and the webview panel that hosts our UI.
  + dependencies: It lists necessary npm packages like axios (for making HTTP requests to our Python backend) and @vscode/webview-ui-toolkit (for the native UI components).

**src/extension.ts (The Brain of the Extension)**

* **Role:** This is the main entry point and core logic file for the TypeScript extension. When the extension is activated, the activate function in this file is executed.
* **Key Functions & Logic Explained:**
  + activate(): This function is the starting point. Its most important jobs are:
    1. **Starting the Backend:** It uses Node.js's built-in child\_process module to run the python backend\_server.py command. This starts our Flask server as a separate background process. It also listens to the server's output (stdout) and errors (stderr) to know its status.
    2. **Registering the UI:** It registers the CodeMitraViewProvider, which is the class responsible for creating and managing our side panel.
  + **Event Listener (onDidSaveTextDocument):** This listener waits for the user to save a file. When a .py or .md file is saved, it reads the file's content and sends it to the Python backend's /analyze endpoint using an axios POST request.
  + CodeMitraViewProvider (Class): This class manages the webview panel. Its resolveWebviewView method sets up the HTML for the UI and handles messages sent *from* the UI (like an "Ask AI" click), forwarding them to the appropriate backend endpoint.

**backend/backend\_server.py (The Brain of the Operation)**

* **Role:** This is a lightweight web server built with **Flask**, a popular Python web framework. It exposes API endpoints (URLs) that our VS Code extension can call to get work done.
* **Key Functions (Endpoints) Explained:**
  + @app.route('/analyze', methods=['POST']): This is the main analysis endpoint.
    - It receives file content and file type in a JSON format from the extension.
    - It runs pylint on the code to find errors.
    - It then creates specific prompts (questions) and calls the query\_gemini function to get AI explanations and error solutions.
    - Finally, it bundles all the results into a single JSON object and sends it back as the HTTP response to the extension.
  + @app.route('/ask', methods=['POST']): This endpoint handles the Q&A feature. It receives a question and code context, sends them to the Gemini API, and returns the answer.
  + query\_gemini(): This is a helper function that contains the actual logic for making the requests.post call to the Google Gemini API. It handles the API key and the request format.

**4. Execution Flow (Step-by-Step)**

Here is how all the pieces work together when a user saves a file:

1. **Activation:** The user clicks the Code-Mitra icon in the VS Code Activity Bar.
2. **Backend Start:** The activate() function in extension.ts runs, which starts the backend\_server.py in the background.
3. **UI Creation:** The CodeMitraViewProvider creates the webview panel.
4. **User Action:** The user saves a Python file (example.py).
5. **Event Fired:** The onDidSaveTextDocument event listener in extension.ts catches this action.
6. **Client Request:** extension.ts reads the content of example.py and sends it to the Python backend via an axios POST request to http://localhost:5001/analyze.
7. **Server Processing:** The Flask server running backend\_server.py receives the request at the /analyze endpoint.
8. **AI & Pylint:** The server runs pylint on the code and also sends prompts to the Gemini API for explanation and solutions.
9. **Server Response:** The server collects all the results and sends them back to the extension as a single JSON response.
10. **UI Update:** extension.ts receives the JSON response and uses postMessage to send the data to the webview.
11. **Final Display:** The main.js script inside the webview catches the message and updates the HTML to display the explanation and error solutions to the user.

**Hindi Version (हिन्दी संस्करण)**

**1. Architectural Overview (आर्किटेक्चर का अवलोकन)**

यह प्रोजेक्ट एक **क्लाइंट-सर्वर आर्किटेक्चर** का पालन करता है। यह एक मजबूत और स्केलेबल डिज़ाइन है जिसमें कार्यों को उनके फ़ंक्शन के आधार पर अलग किया जाता है। इसे एक रेस्टोरेंट की तरह समझें: फ्रंटएंड वेटर है जो आपका ऑर्डर लेता है, और बैकएंड किचन है जो खाना तैयार करता है।

* **क्लाइंट (VS Code Extension - TypeScript):** यह "वेटर" है। यह फ्रंटएंड है जो सीधे उपयोगकर्ता के संपादक के अंदर चलता है। इसकी मुख्य जिम्मेदारियाँ हैं:
  + साइडबार में यूजर इंटरफेस (UI) प्रदान करना।
  + उपयोगकर्ता क्रियाओं का पता लगाना, जैसे फ़ाइल सहेजना या बटन क्लिक करना।
  + फ़ाइल सामग्री ("ऑर्डर") को प्रसंस्करण के लिए बैकएंड सर्वर पर भेजना।
  + सर्वर से अंतिम परिणाम ("भोजन") प्राप्त करना और उन्हें उपयोगकर्ता को खूबसूरती से प्रदर्शित करना।
* **सर्वर (बैकएंड - Python with Flask):** यह "किचन" है। यह एप्लिकेशन का बैकएंड दिमाग है जो एक अलग, स्थानीय वेब सर्वर के रूप में चलता है। इसकी मुख्य जिम्मेदारियाँ हैं:
  + एक्सटेंशन को तेज़ और उत्तरदायी रखने के लिए सभी तर्क-गहन कार्यों को संभालना।
  + कोड विश्लेषण के लिए pylint चलाना।
  + AI अंतर्दृष्टि प्राप्त करने के लिए बाहरी Google Gemini API के साथ संचार करना।
  + अंतिम, संसाधित परिणामों को VS Code एक्सटेंशन पर वापस भेजना।

**यह आर्किटेक्चर क्यों?** यह अलगाव उत्कृष्ट है क्योंकि यह प्रत्येक भाग को वह करने देता है जिसमें वह सबसे अच्छा है। TypeScript उपयोगकर्ता इंटरफेस बनाने और VS Code के साथ एकीकृत करने के लिए शानदार है, जबकि Python, डेटा विज्ञान और AI के लिए अपनी समृद्ध पुस्तकालयों के साथ, भारी डेटा प्रसंस्करण और API संचार को संभालने के लिए एकदम सही है।

**2. Features in Detail (फीचर्स विस्तार से)**

* **AI-Powered Code Explanation:** जब एक Python फ़ाइल सहेजी जाती है, तो एक्सटेंशन कोड को AI को भेजता है, जो तर्क, वैकल्पिक तरीकों और संभावित अनुकूलन का एक सरल स्पष्टीकरण लौटाता है।
* **Intelligent Error Solutions:** एक्सटेंशन त्रुटियों को खोजने के लिए pylint टूल का उपयोग करता है। फिर यह कोड और त्रुटियों दोनों को AI को भेजता है, जो त्रुटियों की व्याख्या करता है और एक सही कोड स्निपेट प्रदान करता है।
* **Interactive Q&A:** उपयोगकर्ता वर्तमान में खुली फ़ाइल के बारे में कोई भी प्रश्न पूछ सकता है। एक्सटेंशन प्रश्न को फ़ाइल की सामग्री के साथ संदर्भ के रूप में AI को एक सटीक उत्तर प्राप्त करने के लिए भेजता है।
* **Markdown Task Planner:** यदि उपयोगकर्ता किसी कार्य विवरण के साथ एक Markdown फ़ाइल सहेजता है, तो AI इसे कार्रवाई योग्य चरणों में तोड़ देता है।
* **Native UI:** इंटरफ़ेस आधिकारिक @vscode/webview-ui-toolkit के साथ बनाया गया है ताकि यह सुनिश्चित हो सके कि यह VS Code का एक स्वाभाविक हिस्सा जैसा दिखता और महसूस होता है।

**3. File-wise Vivaran (फाइल-वाइज विवरण)**

**package.json (The Manifest / घोषणापत्र)**

* **भूमिका:** यह एक VS Code एक्सटेंशन के लिए सबसे महत्वपूर्ण फाइल है। यह एक घोषणापत्र के रूप में कार्य करता है, जो VS Code को एक्सटेंशन के बारे में सब कुछ बताता है, जैसे इसका नाम, सक्रियण घटनाएँ, और निर्भरताएँ।
* **मुख्य खंड (Key Sections):**
  + activationEvents: हम "onView:codeMitraView" का उपयोग करते हैं ताकि यह सुनिश्चित हो सके कि एक्सटेंशन केवल तभी सक्रिय हो जब उपयोगकर्ता उसके आइकन पर क्लिक करे, जिससे VS Code का प्रदर्शन बेहतर होता है।
  + contributes: यह खंड एक्सटेंशन के UI योगदानों को परिभाषित करता है, जैसे एक्टिविटी बार में आइकन और वेबव्यू पैनल जो हमारे UI की मेजबानी करता है।
  + dependencies: इसमें axios (हमारे Python बैकएंड से HTTP अनुरोध करने के लिए) और @vscode/webview-ui-toolkit (देशी UI घटकों के लिए) जैसे आवश्यक npm पैकेज सूचीबद्ध हैं।

**src/extension.ts (एक्सटेंशन का दिमाग)**

* **भूमिका:** यह TypeScript एक्सटेंशन के लिए मुख्य प्रवेश बिंदु और कोर लॉजिक फाइल है। जब एक्सटेंशन सक्रिय होता है, तो इस फाइल में activate फ़ंक्शन निष्पादित होता है।
* **मुख्य फंक्शन्स और लॉजिक:**
  + activate(): यह फ़ंक्शन तब कहा जाता है जब एक्सटेंशन शुरू होता है। इसके सबसे महत्वपूर्ण कार्य हैं:
    1. **बैकएंड शुरू करना:** यह Node.js के child\_process का उपयोग करके python backend\_server.py कमांड चलाता है, जिससे हमारा Flask सर्वर पृष्ठभूमि में शुरू होता है। यह सर्वर की स्थिति जानने के लिए सर्वर के आउटपुट (stdout) और त्रुटियों (stderr) को भी सुनता है।
    2. **UI पंजीकृत करना:** यह CodeMitraViewProvider को पंजीकृत करता है जो हमारे साइड पैनल को बनाने और प्रबंधित करने के लिए जिम्मेदार है।
  + **Event Listener (onDidSaveTextDocument):** यह श्रोता उपयोगकर्ता द्वारा फ़ाइल सहेजने की प्रतीक्षा करता है। जब .py या .md फ़ाइल सहेजी जाती है, तो यह फ़ाइल की सामग्री को पढ़ता है और इसे axios POST अनुरोध का उपयोग करके Python बैकएंड के /analyze समापन बिंदु पर भेजता है।
  + CodeMitraViewProvider (क्लास): यह क्लास वेबव्यू पैनल का प्रबंधन करती है। इसका resolveWebviewView मेथड UI के लिए HTML सेट करता है और UI *से* आने वाले संदेशों को संभालता है (जैसे "Ask AI" क्लिक), उन्हें उपयुक्त बैकएंड समापन बिंदु पर अग्रेषित करता है।

**backend/backend\_server.py (ऑपरेशन का दिमाग)**

* **भूमिका:** यह **Flask**, एक लोकप्रिय Python वेब फ्रेमवर्क के साथ बनाया गया एक हल्का वेब सर्वर है। यह API समापन बिंदुओं (URLs) को उजागर करता है जिन्हें हमारा VS Code एक्सटेंशन काम करवाने के लिए कॉल कर सकता है।
* **मुख्य फंक्शन्स (Endpoints):**
  + @app.route('/analyze', methods=['POST']): यह मुख्य विश्लेषण समापन बिंदु है।
    - यह एक्सटेंशन से JSON प्रारूप में फ़ाइल सामग्री और फ़ाइल प्रकार प्राप्त करता है।
    - यह त्रुटियों को खोजने के लिए कोड पर pylint चलाता है।
    - फिर यह विशिष्ट प्रॉम्प्ट बनाता है और AI स्पष्टीकरण और त्रुटि समाधान प्राप्त करने के लिए query\_gemini फ़ंक्शन को कॉल करता है।
    - अंत में, यह सभी परिणामों को एक एकल JSON ऑब्जेक्ट में बंडल करता है और इसे एक्सटेंशन पर HTTP प्रतिक्रिया के रूप में वापस भेजता है।
  + @app.route('/ask', methods=['POST']): यह समापन बिंदु Q&A सुविधा को संभालता है। यह एक प्रश्न और कोड संदर्भ प्राप्त करता है, उन्हें Gemini API पर भेजता है, और उत्तर लौटाता है।
  + query\_gemini(): यह एक सहायक फ़ंक्शन है जिसमें Google Gemini API पर requests.post कॉल करने का वास्तविक तर्क होता है। यह API कुंजी और अनुरोध प्रारूप को संभालता है।

**4. Execution Flow (एग्जीक्यूशन फ्लो स्टेप-बाय-स्टेप)**

यहाँ बताया गया है कि जब कोई उपयोगकर्ता फ़ाइल सहेजता है तो सभी टुकड़े एक साथ कैसे काम करते हैं:

1. **सक्रियण:** उपयोगकर्ता VS Code एक्टिविटी बार में Code-Mitra आइकन पर क्लिक करता है।
2. **बैकएंड स्टार्ट:** extension.ts में activate() फ़ंक्शन चलता है, जो पृष्ठभूमि में backend\_server.py शुरू करता है।
3. **UI निर्माण:** CodeMitraViewProvider वेबव्यू पैनल बनाता है।
4. **उपयोगकर्ता क्रिया:** उपयोगकर्ता एक Python फ़ाइल (example.py) सहेजता है।
5. **ईवेंट फायर:** extension.ts में onDidSaveTextDocument ईवेंट श्रोता इस क्रिया को पकड़ता है।
6. **क्लाइंट अनुरोध:** extension.ts example.py की सामग्री को पढ़ता है और इसे axios POST अनुरोध के माध्यम से Python बैकएंड पर http://localhost:5001/analyze पर भेजता है।
7. **सर्वर प्रसंस्करण:** backend\_server.py चलाने वाला Flask सर्वर /analyze समापन बिंदु पर अनुरोध प्राप्त करता है।
8. **AI और Pylint:** सर्वर कोड पर pylint चलाता है और स्पष्टीकरण और समाधान के लिए Gemini API पर प्रॉम्प्ट भी भेजता है।
9. **सर्वर प्रतिक्रिया:** सर्वर सभी परिणामों को एकत्र करता है और उन्हें एक एकल JSON प्रतिक्रिया के रूप में एक्सटेंशन पर वापस भेजता है।
10. **UI अपडेट:** extension.ts JSON प्रतिक्रिया प्राप्त करता है और डेटा को वेबव्यू पर भेजने के लिए postMessage का उपयोग करता है।
11. **अंतिम प्रदर्शन:** वेबव्यू के अंदर main.js स्क्रिप्ट संदेश को पकड़ती है और उपयोगकर्ता को स्पष्टीकरण और त्रुटि समाधान प्रदर्शित करने के लिए HTML को अपडेट करती है।