AI Coder Pro – Architecture Overview

By: Khushi Malik **Project link :github**

1. Aim

To provide a powerful, context-aware and agentic AI coding assistant inside VS Code, enabling developers to chat, analyze, refactor, and automate code tasks with ease.

Objectives

- **Seamless AI Integration:**

Integrate advanced AI models (Together AI) for code generation, review, and analysis.

- **Agentic Automation:**

Enable multi-step, autonomous workflows (code review, bug fixing, refactoring) that can apply changes directly to the project.

- **User-Friendly UI:**

Offer a modern, intuitive chat interface with smart agent buttons, file/image upload, and copy features.

- **Contextual Intelligence:**

Maintain and filter conversation memory for highly relevant, non-repetitive AI responses.

- **Extensibility:**

Design the architecture to easily add new agents, workflows, or AI providers in the future.

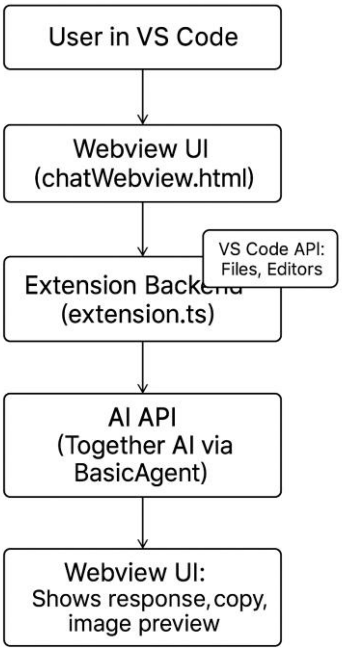
- **Productivity Boost:**

Save developers time by automating repetitive coding tasks and providing instant, actionable code insights.

2. Main Components

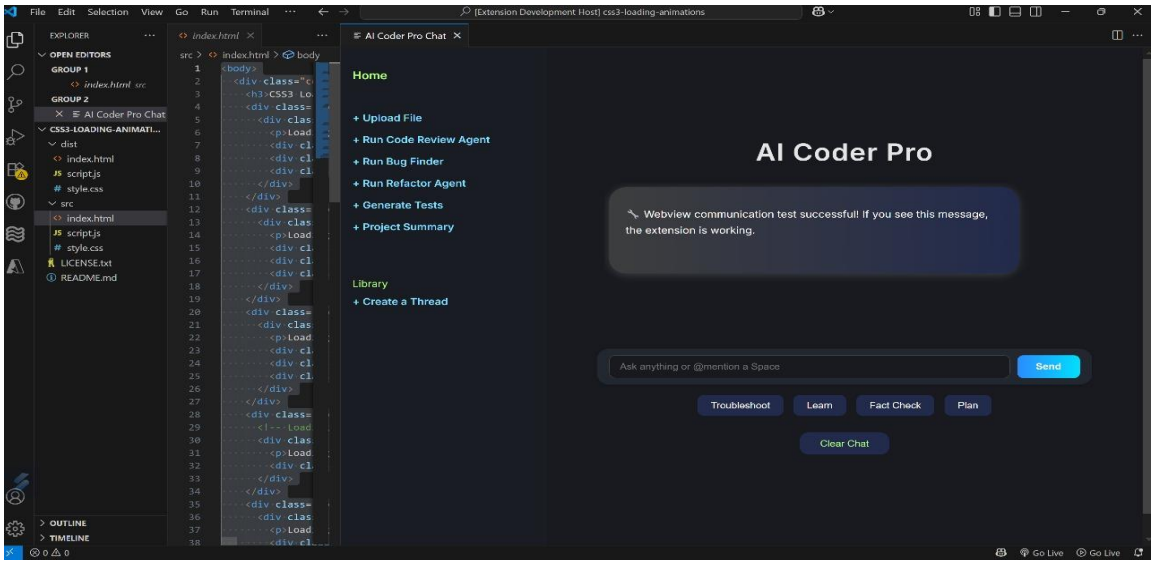
A. Webview UI (chatWebview.html)

- **Purpose:**
  - Provides a modern chat interface inside VS Code.
  - Lets users send prompts, upload files, paste images, and trigger smart agent actions.
- **Features:**
  - Chat area with copy-to-clipboard for each message.
  - Input box for prompts (supports text and image paste).
  - Sidebar with smart agent buttons (Code Review, Bug Finder, etc.).
  - File upload and image preview.
  - Sends messages to backend using vscode.postMessage.



B. Extension Backend (extension.ts)

- **Purpose:**
  - The “brain” of the extension. Handles all logic, context management, AI calls, and file operations.



Key Features

1. **Modern Chat UI**
   - Perplexity-style chat with sidebar, action buttons, and markdown/code rendering.
   - Copy-to-clipboard for every chat message.
2. **Context-Aware AI Chat**
   - Maintains conversation memory for relevant, non-repetitive responses.
   - Filters context to avoid echoing user analysis.
3. **Smart Agentic Workflows**
   - One-click agents for code review, bug finding, refactoring, test generation, and project summary.
   - Agents can analyze, refactor, and even create new files automatically.
4. **File and Image Upload**
   - Upload code files for analysis.
   - Paste images into the chat input (with preview and backend handling).
5. **Automatic File Creation & Code Writing**
   - AI can create new files and write code directly to your project.
6. **Persistence**
   - Conversation and project analysis are saved and restored across sessions.
7. **Error Handling**
   - User-friendly error messages for missing API keys, workspace issues, etc.

C. AI Agent (BasicAgent)

- **Purpose:**

- Encapsulates the logic for calling the Together AI API.
- Handles sending prompts and receiving completions.

D. VS Code API

- **Purpose:**
  - Allows the extension to interact with the user's workspace.
  - Used for:
    - Scanning files
    - Reading/writing files
    - Creating new files
    - Showing notifications

3. How a Typical Request Flows

1. **User Action:**
   - User types a prompt, pastes an image, or clicks a smart agent button in the webview.
2. **Webview Sends Message:**
   - The webview JS sends a message to the backend using vscode.postMessage.
3. **Backend Handles Message:**
   - The backend receives the message, updates conversation memory, and constructs a context-aware prompt.
   - If it's a smart agent action, it scans the project and builds a multi-file analysis prompt.
4. **AI Call:**
   - The backend uses BasicAgent to call Together AI with the constructed prompt.
5. **AI Response:**
   - The backend receives the AI's response.
   - If it's a code change, it parses the response and applies changes to files (creating new files if needed).
6. **Send to Webview:**
   - The backend sends the AI's response (and any status updates) back to the webview for display.

4. Key Design Patterns & Features

- **Separation of Concerns:**
  - UI logic is in the webview; backend logic is in the extension.
- **Message Passing:**
  - All communication between UI and backend is via postMessage and event listeners.
- **Contextual AI:**
  - Context window is managed and filtered for relevance and brevity.
- **Agentic Automation:**
  - Smart agents can analyze, refactor, and even create files automatically.
- **Extensibility:**

- The architecture allows for easy addition of new smart agents, file types, or even new AI providers in the future.

5. Example: Smart Agent Workflow

1. **User clicks "Run Code Review Agent".**
2. **Webview sends { type: 'prompt', prompt: 'Please run a code review agent on my project.' } to backend.**
3. **Backend:**
   - Scans up to 20 project files.
   - Builds a prompt with project structure and code snippets.
   - Calls Together AI.
   - Parses the AI's response for file changes.
   - Applies changes (edits or creates files).
   - Sends a summary and results back to the webview.

6. Persistence & State

- **Conversation memory** and **project analysis** are saved in VS Code's global state.
- When you reload or reopen the extension, your chat and context are restored.

7. Security & Permissions

- The extension only accesses files in the open workspace.
- API keys are managed via VS Code settings or environment variables.

8. How to Extend or Customize

- **Add new smart agents:**
  - Add new sidebar buttons and backend handlers.
- **Support more file types:**
  - Update file scanning and prompt construction logic.
- **Integrate more AI models:**
  - Add new agent classes and routing logic in the backend.

9. Summary Table

Layer	File(s)	Main Role/Responsibility
Webview UI	chatWebview.html	User interface, message sending

Layer	File(s)	Main Role/Responsibility
Backend Logic	extension.ts	Message handling, AI calls, file ops
AI Agent	agents/BasicAgent.ts	API calls to Together AI
VS Code API	vscode (import)	File, editor, and workspace operations