# **Contextual Weaver: Detailed Project Documentation**

Version: 1.0

Author: Sai

## **1. Vision & Executive Summary**

### **1.1 The Problem: The "Amnesia" Barrier**

Modern Large Language Models (LLMs) like Google's Gemini represent a paradigm shift in artificial intelligence. However, their power is constrained by a fundamental architectural flaw: **statelessness**. Each interaction, or "session," exists in a vacuum. The moment a session ends, the model's specific context and learned nuances are lost, a condition we term "AI Amnesia." This makes them unreliable for complex, multi-session tasks, forcing users to repeatedly provide the same context, which is inefficient, error-prone, and a significant barrier to true human-AI collaboration.

### **1.2 The Solution: The Contextual Weaver Architecture**

The Contextual Weaver is not a new AI model; it is a **robust architectural pattern** for building truly stateful, reliable, and autonomous AI agents. Instead of relying on opaque, internal memory features, this architecture treats the core LLM as a powerful but stateless "brain" and wraps it in an explicit, external, and developer-controlled memory and tool system.

**The mission is to transform the AI from a simple, forgetful chatbot into a persistent, context-aware digital assistant capable of managing long-term projects and workflows.**

## **2. In-Depth Architectural Concepts**

The agent's power comes from a clean separation of duties, inspired by human cognition and implemented with robust engineering principles.

### **2.1 The Dual-Memory System (memory\_manager.py)**

The AI's memory is bifurcated to handle different cognitive loads, preventing informational clutter and ensuring efficient recall.

* **The Episodic Journal (Working Memory):**
  + **File:** episodic\_journal.json
  + **Purpose:** This is a simple, chronological log of every single user-AI interaction. Its sole function is to provide the AI with immediate, short-term context on the *flow* of the current conversation.
  + **Mechanism:** Before each prompt, the last n (e.g., 5) interactions are formatted and sent to the AI, acting as its "working memory." This is computationally cheap and ensures conversational coherence.
* **The Semantic Core (Long-Term Memory):**
  + **File:** semantic\_knowledge\_base.json
  + **Purpose:** This is the AI's permanent "brain." It stores a structured, synthesized understanding of key facts, entities, relationships, and user preferences.
  + **The Knowledge Synthesizer:** This is the heart of our long-term memory. After each interaction, a dedicated, secondary AI process is initiated. It analyzes the latest conversation from the journal and is given a single task: extract a list of "actions" (e.g., add\_fact, add\_to\_list) to update the knowledge base. This demotes the AI to an information extractor, while our predictable Python code performs the actual, reliable database update. This prevents AI "hallucinations" from corrupting the memory.

### **2.2 The Agentic Engine & Tool Use (main.py & tools.py)**

This is the system that allows the AI to perceive, think, and act.

* **The Agentic Loop:** The main.py script runs a "Reason-Act" loop. The AI can analyze a problem, decide a tool is needed, and then pause its conversational response to request an action.
* **The Toolbox (tools.py):** This module defines a set of sandboxed functions the AI can use, such as read\_file and write\_file. All file operations are restricted to a safe workspace directory to prevent unintended access to the host system.
* **The Tool Manifest:** The AI is made aware of its capabilities via a TOOL\_MANIFEST string injected into its system prompt. This manifest clearly defines each tool, its purpose, and the arguments it requires.
* **Mission-Aware Feedback Loop:** A critical innovation in our agent is how it handles tool results. After a tool is executed, the result is fed back to the AI along with a reminder of the user's *original request*. This prevents "agent derailment," a common failure mode where an AI completes one step of a multi-step task and forgets the overall mission.

## **3. Detailed File Structure**

/contextual\_weaver/  
|  
|-- .venv/ # Your isolated Python virtual environment.  
|  
|-- memory/ # Stores the AI's persistent state.  
| |-- episodic\_journal.json # The raw, chronological log of all conversations.  
| |-- semantic\_knowledge\_base.json # The structured, synthesized "brain" of the AI.  
|  
|-- workspace/ # A sandboxed directory where the AI can safely read and write files.  
|  
|-- .gitignore # Standard file to prevent secrets and temporary files from being committed to Git.  
|-- config.py # Securely stores your API key, kept separate from the main logic.  
|-- main.py # The main application entry point. Contains the primary agentic loop.  
|-- memory\_manager.py # The "brain trust." Handles all memory I/O and runs the knowledge synthesis process.  
|-- tools.py # The "hands." Defines the set of functions the AI can execute.

## **4. Setup and Installation Guide**

Follow these steps to set up and run the project.

**Prerequisites:** Python 3.8 or higher.

1. **Create a Virtual Environment:**
   * Open the project folder in VS Code.
   * Press Cmd + Shift + P (or Ctrl + Shift + P) and select Python: Create Environment.
   * Choose Venv and select your Python interpreter (e.g., Python 3.11). This creates the .venv folder.
2. **Install Dependencies:**
   * Open a new terminal in VS Code (Terminal > New Terminal). VS Code should automatically activate the new environment (you'll see (.venv) in your prompt).
   * Run the installation command:  
     pip install google-generativeai
3. **Configure API Key:**
   * Navigate to [Google AI Studio](https://aistudio.google.com/) and generate a new API key.
   * Open the config.py file.
   * Paste your key into the string: GEMINI\_API\_KEY = "YOUR\_API\_KEY\_HERE"

## **5. How to Run the Agent**

1. Open a terminal in VS Code.
2. Ensure your virtual environment is activated. If not, run: source .venv/bin/activate
3. Run the main script:  
   python main.py

## **6. Showcase Demo Walkthrough: "The Three-Day Project"**

This walkthrough demonstrates the agent's full capabilities over multiple sessions, proving its persistence and reliability.

### **Day 1: Project Kickoff**

* **Setup:**
  1. Ensure the memory and workspace folders are empty.
  2. Create a file named reqs.txt **inside the workspace folder**.
  3. Add this content to reqs.txt:  
     Project Name: Project Chimera  
     Objective: Develop a technical proposal for our new AI-driven data analysis platform.  
     Key Deliverable: A 2-page technical proposal document.
* **Run:** Start the agent with python main.py.
* **Prompt:** Good morning. We are starting a new project called Project Chimera. Please read the initial requirements from the file named "reqs.txt" and then write a short project brief into a new file in the workspace called "brief.md".
* **Expected Outcome:**
  + The terminal will show the AI calling read\_file('reqs.txt') and then write\_file('brief.md', ...).
  + A new file, brief.md, will appear in your workspace folder containing a summary of the requirements.
  + The semantic\_knowledge\_base.json will now contain an entry for "Project Chimera."
* **Action:** Type quit.

### **Day 2: Incorporating Feedback**

* **Setup:**
  1. Create a new file in the workspace folder named feedback.txt.
  2. Add this content to feedback.txt:  
     The proposal needs to emphasize our unique data synthesis algorithm. Also, please add a section on security compliance.
* **Run:** Start the agent again (python main.py).
* **Prompt:** Good morning. Please incorporate the feedback from "feedback.txt" into the project brief.
* **Expected Outcome:**
  + The agent, using its long-term memory, will immediately know about Project Chimera and the brief.md file.
  + It will call read\_file('feedback.txt') and read\_file('brief.md').
  + It will then call write\_file('brief.md', ...) to save an updated version that includes the new sections.
* **Action:** Type quit.

### **Day 3: Finalizing**

* **Run:** Start the agent again (python main.py).
* **Prompt:** Please give me a final summary of Project Chimera and list all the files currently in the workspace.
* **Expected Outcome:**
  + The agent will consult its semantic\_knowledge\_base.json to provide a perfect, up-to-date summary of the project.
  + It will then call the list\_files tool to show all the work it has done (reqs.txt, feedback.txt, brief.md).
* **Conclusion:** This final step proves the agent's long-term memory and agency are fully functional.