### **Sales Proposal Agent (SalesProposalWriting\_Agent) Script Analysis**

This table provides a comprehensive breakdown of the entire multi-file agentic workflow designed to generate sales proposals. The analysis covers the agent's structure, state management, content generation, and final document creation.

| **File** | **Function/Method Name** | **Input Parameters** | **Output [Datatype]** | **Comments** |
| --- | --- | --- | --- | --- |
| **llms.py** | llm (variable) | model: str | ChatGoogleGenerativeAI | Initializes and configures the connection to the Gemini Flash LLM, making it available for use throughout the agent. |
| **sections.py** | create\_sections | state: State | dict | **Node 1.** This is the first logical step. It takes the seller's information from the state, uses an LLM to generate a relevant list of proposal section titles (e.g., "Executive Summary", "Scope of Work"), and updates the state with this list. |
| **sales\_proposal\_content\_writing.py** | write\_sales\_proposal | state: State | dict | **Node 2.** The core content generation step. It takes all the client/seller data and the list of sections from the state, feeds them into a detailed prompt (proposal\_template), and invokes the LLM to write the full text for every section of the proposal. |
| **nodes.py** | clean\_data | state: State | None | **Node 3.** The final processing step in the graph. It takes the raw, JSON-like string output from the LLM, cleans it, and writes the structured content into a plain text file (output.txt). |
| **agent.py** | graph (variable) | State | CompiledGraph | Defines the structure of the agent using LangGraph. It wires the nodes together in a specific sequence: create\_sections -> write\_sales\_proposal -> clean\_data. This compiled graph is the executable agent. |
| **main.py** | get\_presentation | client, seller, project\_specs, output\_format: str | (str, str, str) | **The main entry point.** It initializes the agent's state, but **it does not run the LangGraph agent**. Instead, it directly calls generate\_modern\_presentation to create the HTML and then generate\_pdf\_from\_html to convert it. |
| **main.py** | generate\_pdf\_from\_html | html\_content: str, output\_dir: str, base\_filename: str | str (file path) | A robust PDF conversion utility. It tries to generate a PDF using WeasyPrint first. If that fails, it falls back to pdfkit. If both fail, it saves the HTML file so the user can convert it manually. |
| **main.py** | clean\_html\_for\_pdf | html\_content: str | str | A helper function that uses BeautifulSoup to parse and clean the generated HTML, removing elements like <script> tags that can cause issues during PDF conversion. |
| **main.py** | get\_pdf\_css | None | str | Returns a block of CSS specifically optimized for printing and A4 page layout, ensuring the PDF looks clean and professional. |
| **sales\_proposal\_html\_writing.py** | ModernPresentationConfig | theme: str, custom\_colors: dict | ModernPresentationConfig instance | A configuration class that holds color schemes and styling options for the HTML proposal, allowing for easy theming ("corporate", "premium", etc.). |
| **sales\_proposal\_html\_writing.py** | get\_modern\_css | self, logo\_url: str, logo\_url\_2: str | str | A method within the config class that generates a complete, dynamic CSS stylesheet based on the selected theme colors and provided logos. |
| **sales\_proposal\_html\_writing.py** | generate\_modern\_presentation | filename: str, logo\_url: str, various optional params | (str, str) | Takes the raw text file (output.txt), parses it into sections, and injects the content into a sophisticated HTML template with dynamic styling based on the section titles (e.g., special formatting for "Pricing" or "About Us"). |
| **prompts.py** | (variables) | N/A | str | Contains the string templates for the prompts used by the LLM. section\_template is for generating the section list, and proposal\_template is the main, highly detailed prompt for writing the full proposal content. |
| **proposal\_temp.py** | (variable) | N/A | str | Contains a hardcoded example of the expected text output from the write\_sales\_proposal node. This is likely used for testing or as a fallback template. |