### **Web Scraper (WebScraper/) Script Analysis**

This table provides a comprehensive breakdown of the scripts designed to scrape websites. It covers two main methodologies: an advanced agent-based approach using crawl4ai with an LLM for structured data extraction, and a classic approach using BeautifulSoup for rule-based scraping.

| **File** | **Function/Method Name** | **Input Parameters** | **Output [Datatype]** | **Comments** |
| --- | --- | --- | --- | --- |
| **state.py** | User (Pydantic Model) | name: str, logo: str, description: str, services: List[str] | User instance | The central data structure for the project. It defines the schema (name, logo, description, services) that all scraping methods aim to populate, ensuring consistent and validated output. |
| **scrape\_agent.py** | llm\_strategy (variable) | llm\_config, schema, instruction, etc. | LLMExtractionStrategy | Configures the AI-powered extraction. It tells the crawl4ai library to use a Gemini model and provides a detailed prompt instructing it how to find the name, logo, description, and services on a webpage and format them according to the User schema. |
| **scrape\_agent.py** | aggregate\_users | users: List[dict] | User instance | A data cleaning function. Since the AI crawler might extract multiple User objects from different parts of a page, this function intelligently merges them into a single, definitive result (e.g., by picking the most common name and the longest description). |
| **scrape\_agent.py** | get\_url\_detail\_using\_googleai | url: str | User instance | **(Primary AI Method)** The main entry point for the crawl4ai scraper. It takes a URL, runs the asynchronous crawler with the llm\_strategy, and returns the final, aggregated User object. |
| **webscraper\_beautifulsoup.py** | WebScraper (Class) | delay: float | WebScraper instance | A class that encapsulates the traditional scraping logic. It contains methods for fetching pages and extracting specific pieces of information using predefined CSS selectors. |
| **webscraper\_beautifulsoup.py** | extract\_name, extract\_logo, etc. | self, soup, url | str or List[str] | These are the core methods of the WebScraper class. Each one attempts to find a specific piece of data (like the company name) by searching through a list of common HTML tags and class names (e.g., img[class\*="logo"]). |
| **webscraper\_beautifulsoup.py** | scrape\_website | self, url: str | User instance or None | The main method of the WebScraper class. It fetches a page and calls all the individual extract\_\* methods to gather the data, then validates and returns it as a User object. |
| **webscraper\_beautifulsoup.py** | get\_url\_details\_with\_bs4 | url: str, timeout: int | User instance or str | **(Primary Traditional Method)** A wrapper function that instantiates the WebScraper class and runs the scraping process within a timeout to prevent it from hanging on slow websites. |
| **scrape\_utils.py** | extract\_hex\_colors | url: str, limit: int | list | A standalone utility function that scrapes a website's CSS to find and return a list of the most frequently used color hex codes, which could be used for branding analysis. |
| **scrape\_alternatives\_for\_playwright.py** | (Entire script) | N/A | N/A | This file appears to be an experimental or debugging version of scrape\_agent.py. It contains various commented-out configurations (BrowserConfig) for running crawl4ai with different browser engines (like Firefox or WebKit) instead of the default. |
| **main.py** | main | None | None | A simple script to test the scrape\_agent. It runs the get\_data function (which is an alias for the main scraping function in the alternative script) on a single URL (whatsapp.com) and prints the result. |