Rufus Web Scraper Documentation

# Overview

The AI Web Scraper is a tool that uses AI models for web scraping.   
 It extracts specific content from a webpage based on user-specified descriptions, processes the content,   
 and allows users to download the parsed results in different formats like text files, CSV, and JSON.

# File Structure

The project consists of three main Python files:  
 1. scrape.py: Handles web scraping using Selenium and BeautifulSoup.  
 2. parse.py: Manages parsing of scraped content using the Ollama language model.  
 3. main.py: Contains the Streamlit app interface for users to input URLs, scrape sites, and parse content.

# scrape.py

scrape.py contains functions for scraping content from websites:  
 - scrape\_website: Loads a webpage using Selenium and returns the HTML source.  
 - extract\_body\_content: Extracts the body content from the HTML.  
 - clean\_body\_content: Cleans up the body content by removing script and style tags, leaving only text.  
 - split\_dom\_content: Splits long content into smaller chunks to be parsed.

# parse.py

parse.py contains the logic to parse content using the Ollama model:  
 - parse\_with\_ollama: Uses the OllamaLLM model to extract specific information from a DOM chunk based on a description.

# main.py

main.py is the Streamlit application that provides a user interface. Users can input a URL, scrape the website,   
 view the raw DOM content, enter descriptions for parsing, and download the results in various formats.

# Features

1. Web Scraping: Uses Selenium and BeautifulSoup to scrape webpage content.  
 2. Content Cleaning: Cleans up unnecessary tags and retains only the visible text.  
 3. AI Parsing: Utilizes the Ollama model to extract specific information based on user descriptions.  
 4. Export Options: Users can download the parsed content in TXT, CSV, or JSON formats.

# Usage

1. Run the Streamlit app using the command: streamlit run main.py.  
 2. Enter a URL in the input field and click 'Scrape Site'.  
 3. View the raw DOM content and enter a parsing description.  
 4. Click 'Parse' to extract specific information.  
 5. After parsing, click 'Download' to save the results in your preferred format.

# Dependencies

- Selenium: For web scraping.  
 - BeautifulSoup: For parsing HTML content.  
 - Langchain: For utilizing the Ollama AI model.  
 - Streamlit: For building the user interface.  
 - Pandas: For handling CSV downloads.  
 - json: For handling JSON downloads.

# Error Handling

The application handles various exceptions such as:  
 - Invalid URL: Prompts the user to enter a valid URL.  
 - Parsing Errors: Catches errors during the parsing process and displays an error message.  
 - File Saving Errors: Notifies the user in case of issues when saving the parsed content.

**Summary:**

In this project, I developed an AI Web Scraper using Streamlit, Selenium, BeautifulSoup, and LangChain. The scraper extracts, processes, and parses website content. The goal was to create a tool that scrapes a website’s DOM content, cleans and formats it, and then allows users to parse it based on their own descriptions using an AI model. The final output could be downloaded as .txt, .csv, or .json files.

**Challenges Faced:**

1. Parsing with LangChain and AI models:  
 - Initially, the integration of LangChain with the OllamaLLM model was not straightforward, and I encountered errors related to improper input formatting.  
 - Solution: By refining the prompt format, ensuring correct input-output management, and using appropriate models, the parsing functionality was made reliable.  
  
2. Handling large volumes of DOM content:  
 - Scraping large websites with significant amounts of data led to issues in handling memory and managing the chunks effectively.  
 - Solution: I split the DOM content into manageable chunks (up to 6000 characters each) for processing and parsing.  
  
3. Streamlit UI/UX:  
 - There were some challenges related to displaying dynamic content and providing the ability to download parsed results in various formats like .txt, .csv, or .json.  
 - Solution: The use of st.session\_state helped persist the content, and logic for file downloads was added based on content type.