Intelligent Web Scraping Bot

An advanced web scraping application that integrates web scraping capabilities with natural language processing, providing an interactive chat interface for extracting and analyzing web content.

Problem Statement

Traditional web scraping tools often provide raw data that requires additional processing and analysis. Users need a more intuitive and efficient way to:

- Extract structured content from websites
- Ask questions about the scraped content
- Get meaningful insights from the data
- Interact with the scraping tool using natural language

Intelligent Web Scraping Bot addresses these challenges by combining web scraping with conversational AI, offering a seamless and interactive experience.

Features

🌐 Intelligent Web Scraping

- **Content Extraction**: Scrapes and structures content from any website (e.g., text, links, headings).
- **Dynamic Content Handling**: Works with JavaScript-rendered websites and dynamic HTML structures.
- **Error Recovery**: Robust error handling to gracefully handle issues like missing data or bad responses.

💬 Interactive Chat Interface

- **Natural Language Queries**: Ask questions in natural language (e.g., “Summarize this page,” “What are the main topics?”).
- **Contextual Responses**: Provides context-aware answers based on the content scraped from the website.
- **Real-time Interaction**: Instant feedback from the AI bot based on the web data.

🎨 Modern UI/UX

- **Responsive Design**: Mobile-first interface ensuring a smooth experience on all devices.
- **Dark/Light Mode Toggle**: Customizable theme for user comfort.
- **Loading Indicators**: Clear visual cues during content scraping and processing.

- **User-friendly Error Messages**: Helpful and non-intrusive error messages for better user experience.

🔍 Content Analysis

- **Automatic Summarization**: Summarizes large blocks of text to highlight key information.
- **Structured Presentation**: Content organized by type (e.g., headings, links, paragraphs).
- **Link Extraction**: Extracts and organizes links from the scraped webpage for easy navigation.

Technical Requirements

Backend Dependencies:

- **flask** (>= 3.0.0) - For building the web server and handling routes.
- **requests** (>= 2.31.0) - For making HTTP requests to scrape websites.
- **beautifulsoup4** (>= 4.12.2) - For parsing HTML content and extracting data.
- **aiohttp** (>= 3.9.1) - Asynchronous HTTP requests for improved performance.
- **pandas** (>= 2.1.4) - For organizing and processing extracted data.
- **rich** (>= 13.7.0) - For terminal-based output formatting and logs (if applicable).
- **openpyxl** (>= 3.1.2) - For exporting data to Excel if needed.
- **transformers** (>= 4.36.0) - For NLP models used to process and respond to user queries.
- **torch** (>= 2.1.0) - PyTorch backend for NLP models.
- **nest_asyncio** (>= 1.5.8) - Allows nested asyncio event loops.
- **brotli** (==1.1.0), **brotlipy** (==0.7.0) - For handling compressed responses.

Frontend Technologies:

- **HTML5, CSS3** (with Tailwind CSS) - For building a modern, responsive interface.
- **JavaScript (ES6+)** - For interactive functionality and API communication.
- **Font Awesome** - For incorporating icons.
- **Google Fonts (Inter)** - For sleek and readable typography.

Architecture

Core Components:

1. **Web Scraping Module** (`scraper.py`)

    - Manages HTTP requests, parses the HTML content, and extracts the relevant data.
    - Handles dynamic content and compressed responses (e.g., Brotli).

2. **Chat Bot Module** (`web_scraping_bot.py`)

    ○ Processes user queries and maintains context across interactions.
    ○ Uses NLP models to generate relevant responses based on scraped content.
3. **Flask Server** (`app.py`)

    ○ Manages HTTP routes and API endpoints.
    ○ Coordinates communication between the frontend (HTML) and backend (scraping, chat).
4. **Frontend Interface** (`templates/index.html`)

    ○ Provides the user interface to interact with the bot.
    ○ Displays scraped content and allows for user input to ask questions.

Implementation Approach

Web Scraping Strategy:

- **Asynchronous Requests**: Utilizing `aiohttp` for concurrent scraping to improve performance.
- **Error Handling**: Gracefully handles issues like missing data, broken links, or failed requests.
- **Content Extraction**: Dynamically identifies and extracts content using BeautifulSoup, adjusting to various HTML structures.
- **Brotli Compression**: Supports Brotli-compressed responses for faster data transfer.

Natural Language Processing:

- **Query Understanding**: Uses advanced NLP techniques to parse and understand user queries.
- **Context Maintenance**: Ensures conversation context is maintained across interactions for coherent responses.
- **Response Generation**: Generates meaningful, context-specific answers based on the extracted content.

User Interface Design:

- **Responsive Design**: The application is designed for mobile-first, ensuring an optimal experience on all devices.
- **Interactive Chat Interface**: Provides a clean, intuitive chat interface for users to ask questions and get real-time responses.
- **Accessibility**: The interface is designed with accessibility in mind (e.g., readable fonts, high contrast).

Setup and Installation

Follow these steps to set up the project locally:

Clone the repository:

```
 git clone
https://github.com/shivarajm8234/Infosys-Assignments/tree/main/MileStone%201/web_scraping_bot
cd web_scraping_bot
```

1.

Create and activate a virtual environment:

```
 python -m venv venv
source venv/bin/activate  # On Windows: venv\Scripts\activate
```

2.

Install the required dependencies:

```
 pip install -r requirements.txt
```

3.

Run the application:

```
 python app.py
```

4.
5. **Access the application**: Open your web browser and go to http://localhost:5000.

Usage

Scraping Content:

1. **Enter a URL**: Type the URL of the website you want to scrape in the input field.
2. **Click “Scrape”**: Press the "Scrape" button or hit Enter to start the scraping process.
3. **Wait for content extraction**: The bot will extract the content and organize it for easy access.

Asking Questions:

1. **Type your question**: Ask anything related to the scraped content (e.g., "Summarize the page", "What is the main topic?").
2. **Click “Send”**: Press the "Send" button or hit Enter to send the query.

3. **View the AI’s response**: The bot will respond with an answer based on the scraped content.

Viewing Content:

1. **Organized Content**: Extracted content appears in the right panel.
2. **Content Breakdown**: The content is categorized by type (headings, links, paragraphs).
3. **Summary**: A quick summary of the extracted content is available for quick reference.

Future Enhancements

Advanced Features:

- **PDF Export**: Option to export the extracted content and summaries to PDF.
- **Multi-language Support**: Enable querying in different languages for a global user base.
- **Custom Scraping Rules**: Users can define custom scraping parameters (e.g., specific sections or tags).
- **Data Visualization**: Visual representations of scraped data for enhanced insights.

Technical Improvements:

- **Caching**: Cache common data to reduce scraping frequency and improve performance.
- **Rate Limiting**: Implement rate limiting to avoid overloading websites with too many requests.
- **Advanced Error Recovery**: Implement advanced strategies to handle failed requests and recover gracefully.
- **Session Management**: Support for session-based scraping to maintain context across requests.

UI Enhancements:

- **More Theme Options**: Offer more themes (e.g., light, dark, and custom themes).
- **Customizable Layout**: Allow users to personalize the layout of the UI.
- **Advanced Search**: Integrate a search feature for easily navigating large amounts of scraped content.
- **Voice Interaction**: Add voice recognition to ask questions hands-free.

Contributing

We welcome contributions! If you'd like to contribute to the project, please follow these steps:

1. Fork the repository.
2. Create a new feature branch.
3. Make your changes and commit them.

4. Push your changes to your fork.
5. Submit a Pull Request for review.

License

Acknowledgments

- **Web Scraping APIs**: For helping gather raw content from websites.
- **Open-source Contributors**: For providing libraries and tools that enhance the functionality of this project.
- **NLP Models**: For enabling meaningful responses based on user queries.

Support

For any questions, issues, or support:

- Open an issue in the repository.
- Contact the development team directly.
- Join our community forum for discussions and tips.

GitHub Link :-

https://github.com/shivarajm8234/Infosys-Assignments/tree/main/MileStone%201/web_scraping_bot