ScrapyBot Documentation

Problem Statement

Users often need to quickly understand and extract information from websites, but manually reading through entire webpages is time-consuming and inefficient. There's a need for a tool that can:

- Extract relevant content from any given website.
- Provide quick summaries of webpage content.
- Allow users to ask specific questions about the content.
- Present information in an interactive and user-friendly way.

Solution Overview

ScrapyBot is a web application that combines web scraping capabilities with an AI-powered chatbot interface. It allows users to:

- Input any website URL for content extraction.
- Receive an automatic summary of the webpage.
- Engage in a Q&A conversation about the scraped content.
- Get accurate, context-aware responses based on the webpage's content.

Key Features

1. Web Scraping

- Headless browser implementation: Enables seamless content extraction.
- Intelligent content extraction: Focuses on meta descriptions, page titles, main content areas, and important HTML elements (headings, paragraphs, lists).
- Error handling: Covers scenarios such as inaccessible pages or missing data.

2. Content Processing

- Smart content truncation: Ensures that summaries are concise while preserving meaning.
- Session-based storage: Maintains context for seamless interactions.
- Structured data formatting: Organizes extracted content for easy interpretation.

3. AI Integration

- Context-aware responses: Uses the LLaMA model to generate answers based on the webpage content.
- Automatic webpage summarization: Provides concise overviews of scraped data.
- Clear response formatting: Ensures readability and consistency.

Technical Requirements

Backend Dependencies

- Flask: For building the web server and handling routes.
- Selenium: For dynamic content scraping.
- Requests: For HTTP requests.

Frontend Technologies

- HTML, CSS, JavaScript: For creating a responsive UI.
- Font Awesome: For icons.
- Google Fonts: For typography.

Architecture

1. Frontend Interface

- Built using HTML, CSS, and JavaScript, the frontend provides a responsive and user-friendly experience.
- Features input fields for entering URLs, a chat interface for queries, and sections for displaying extracted content.
- Communicates with the backend via API endpoints.

2. Flask Backend

- Serves as the bridge between the frontend and the processing modules.
- Hosts API endpoints for triggering scraping operations and AI interactions.
- Manages user sessions to retain context across multiple queries.

3. Web Scraping Module

- Powered by Selenium, this module handles dynamic webpage rendering and extraction of structured content.
- Dynamically adapts to various HTML structures to identify critical elements like headings, paragraphs, and metadata.
- Incorporates error-handling mechanisms to ensure robust operations even with problematic webpages.

4. AI Processing Module

- Utilizes the LLaMA API for generating context-aware responses and automatic summaries.
- Leverages state-of-the-art natural language processing techniques to maintain conversational context.
- Processes user queries and aligns responses with the scraped content.

5. Data Storage and Session Management

- Stores scraped data and user interactions temporarily to ensure continuity during sessions.
- Prevents redundant scraping by caching results during active user sessions.

6. Communication Flow

1. The user interacts with the frontend to input a URL or a query.
2. The backend triggers the web scraping module to retrieve content from the given URL.
3. Extracted content is processed and structured, then stored in session memory.
4. For queries, the AI processing module generates responses based on the stored content.
5. Responses and extracted content are displayed on the frontend for user consumption.

Implementation Approach

1. Web Scraping Strategy:
    - Uses headless browser automation with Selenium.
    - Parses HTML content dynamically rendered by JavaScript.
    - Extracts relevant metadata and main content areas.
2. Content Processing:
    - Organizes scraped data into structured formats.
    - Stores session-based information for ongoing Q&A.
3. AI Integration:
    - Interacts with the LLaMA model for generating responses.
    - Maintains conversation context for meaningful interactions.

Setup and Installation

1. Clone the repository:

```
git clone <repository-url>

cd <repository-directory>
```

2. Install dependencies:

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

3. Configure environment variables:

```
export LLAMA_API_URL=<your_api_url>

export API_KEY=<your_api_key>

export FLASK_SECRET_KEY=<your_secret_key>
```

4. Run the application:

```
python backend/app.py
```

Usage

Scraping Content

1. Enter the URL of the target website.
2. Click "Scrape" to start extracting content.
3. View organized content in the interface.

Asking Questions

1. Input your question in the chat interface.
2. Click "Send" or press Enter.
3. Receive context-aware AI-generated responses.

Viewing Content

1. Content is categorized (headings, links, paragraphs).
2. Summaries provide a quick overview.

Future Enhancements

1. Support for authentication-required websites: Scrape gated content.
2. PDF and document scraping: Expand capabilities beyond HTML.
3. Multi-language support: Enable global accessibility.
4. Custom scraping rules: Allow users to define specific sections or tags.
5. Advanced error handling: Improve reliability.
6. Caching: Reduce repetitive scraping for frequently visited sites.
7. Voice interaction: Introduce voice-based Q&A.

GitHub Repository

For more details, visit the ScrapyBot GitHub repository.