Online-read-assistant

Webscraper to gather all available information from webpage and using openAI API to traverse it effectively, provided a personal communication with the processed information.

This application had been an idea for a project from before, but DAT158 oblig2 being a compulsory assignment became a motivation to get this produced.

We chose GPT-3.5-turbo for its higher token-count and low cost, as information gathered can be very extensive and openAI API having a restriction on token-count. The application combines a React frontend with a Node.js backend, using Express for server management locally and Puppeteer for web content scraping. The askGPT function in the backend facilitates communication with the GPT model, processing chunks of data split into sections of 4000 chars. Users can input URLs for scraping and receive all text-based information. The askGPT function is not yet fully functional.

Testing on various webpages revealed low use of HTML-only text, leading to the development of our Node.js backend with Puppeteer, since JavaScript content was primarily used. Tests used URLs from public institutions and news sites, yielding informative and user-friendly results. Integrating the OpenAI API was challenging, requiring resources like W3Schools for understanding OpenAI's version 4 API.

Transitioning to Node.js presented a learning curve, where React felt far more familiar. We shifted from Cheerio to Puppeteer for JavaScript content handling after extensive attempts at keeping it integrated. Console logging and a step-by-step approach aided in troubleshooting. Managing cookie requests, pop ups, image-files and integrating the OpenAI API were and still are significant challenges.

The web scraping capabilities exceeded expectations, demonstrating robust data gathering. Ethically, we focused on copyright and web traffic management, processing data client-side without storage. Information are usually property of the webpage itself, so webscraping and monetizing that information borders on the illegal. The project highlighted the potential of simple programs and the importance of human oversight in coding. Potential future challenges include evolving webpage responses to scraping technologies.

The project's GitHub repository is comprehensive, with a focus on personal understanding. It includes a step-by-step plan, code comments, and is divided into backend and frontend sections. To replicate the project, one needs a Node.js and React environment, with dependencies installable via npm install <dependency>. With openAI integration an API key is needed, API keys are personal, limited and costly. The code is housed in a private GitHub repository for those with access.

The main achievement is developing a functional web scraping application. It excels in gathering text-based information from domains, where it follows all available links iteratively and holds potential for streamlining information access, especially for data-dense sites. The only limit is currently the user’s limitation in the code itself and easily changed with a few keystrokes. The project emphasized the usefulness of tools like ChatGPT in unfamiliar programming areas, reaffirming the need for human input for clarity and coherence in coding.

Future development focuses on integrating the OpenAI API and improving handling of webpage quirks like pop-ups and cookie requests. Currently they could be solved easily by documenting the cookies hard coded, but this goes against the autonomous prospect for the program. The long-term goal is to develop the application for personal use and personal connections.

References

1. OpenAI. (October-November 2023). ChatGPT 4.0. [Software]. Used for project guidance and troubleshooting.
2. Birkeland, Mikkel. (October-November 2023). Personal Communication. Provided insights on React-based front-end development. Planned but not yet implemented transition to Next.js frontend.