WebGenius Project Architecture

Overview

WebGenius is a Chrome Extension + FastAPI Backend project that allows users to scrape any webpage and ask questions based on its content using LangChain + LLMs (like OpenAI or Perplexity).

Frontend Files (Chrome Extension)

- manifest.json: Declares extension metadata and permissions.
- popup.html: UI for the extension (button, textarea, output box).
- popup.js: Logic to capture the URL, store it, send request to backend, and display response.
- styles.css: Styling for the popup UI.

Backend Folder Structure (FastAPI + LangChain)

- backend/
  ??? app/
  ?   ??? __init__.py: Marks app/ as a module.
  ?   ??? main.py: Entry point for FastAPI, sets up CORS and routes.
  ?   ??? routes/
  ?   ?   ??? query.py: Defines the /query endpoint logic.
  ?   ??? models/
  ?   ?   ??? schema.py: Pydantic models for request and response.
  ?   ??? services/
  ?   ??? scraper.py: Scrapes webpage using LangChain WebBaseLoader.
  ?   ??? qa.py: Performs question answering using LLMs (OpenAI/Perplexity).
  ??? .env: Stores API keys (OPENAI_API_KEY, PERPLEXITY_API_KEY).
  ??? .gitignore: Prevents secrets, virtual envs, and temp files from being tracked by Git.
  ??? requirements.txt: Lists Python dependencies.
  ??? README.md: Documents setup, usage, and structure.

Backend End-to-End Flow

1. Chrome extension captures the active tab URL.
2. User enters a question in popup UI.
3. Extension sends URL + question to FastAPI backend.
4. Backend uses LangChain to scrape the webpage.
5. Scraped content is passed to LLM to answer the question.
6. The answer is sent back and shown in the popup.

Security and Best Practices

- Use virtual environment for dependency isolation.
- Store secrets in a .env file and never commit them.
- Use .gitignore to exclude unnecessary or sensitive files.

WebGenius Project Architecture

- Add CORS middleware in FastAPI to allow extension communication.
- Document project setup and structure clearly in README.md.

Optional Enhancements

- Add unit tests (tests/ folder).
- Dockerize backend for deployment.
- Add error handling and logging.
- Add rate limiting or API usage limits.

Next Steps

1. Write backend files (main.py, query.py, scraper.py, qa.py, schema.py).
2. Integrate OpenAI or Perplexity in qa.py.
3. Connect frontend to backend and test end-to-end.
4. Push to GitHub and document everything.