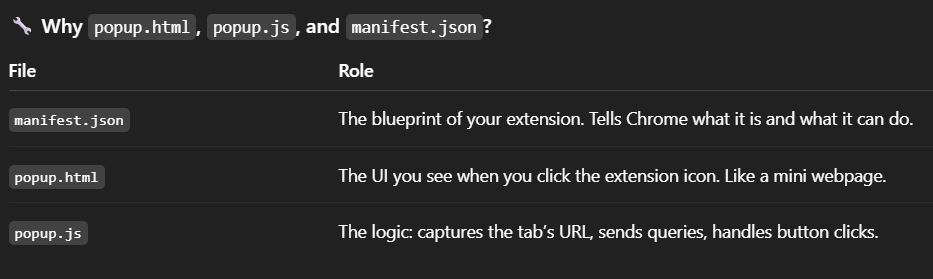
**WebGenius(Smart Chatbot-Powered Web Summarizer & Extractor as a Chrome Extension)**

**🧩 What is a Chrome Extension?**

A **Chrome extension** is like a mini web app that runs in the browser. It can:

* Access and interact with tabs
* Inject scripts into web pages
* Store data
* Add a custom UI (popup/sidebar)

It works via a config file called manifest.json which defines:

* What permissions it needs
* What scripts it runs
* What UI it shows

This is how a chrome extension works and has these three types of files in it.  
Here we are creating the unpacked version of it.

**manifest.json**

Defines the extension settings:

* default\_popup: which HTML to open
* permissions: access to activeTab, scripting, etc.
* host\_permissions: URL access allowed

**popup.html**

UI of the extension:

* “Scrape this Page” button
* Textarea to enter the question
* Ask button
* Display area for the answer

**popup.js**

Core frontend logic:

* Gets active tab URL using chrome.tabs.query
* Saves URL in chrome.storage.local
* Sends query to backend using fetch
* Displays the answer in the popup

**styles.css**

Basic styling for the popup:

* Fonts
* Button design
* Textarea
* Response display box

**🧠 Backend (FastAPI + LangChain)**

**✅ Backend Core: backend/app/**

**main.py**

* Entry point for FastAPI
* Sets up CORS (to allow Chrome extension access)
* Registers routers (like /query)

**\_\_init\_\_.py**

Empty file, just makes app/ a Python module.

**📂 routes/**

**query.py**

* Defines /query POST endpoint
* Accepts JSON with url and question
* Calls scraper.py to load the page
* Passes result to qa.py to answer the question
* Returns the answer

**📂 models/**

**schema.py**

Contains Pydantic models:

* QueryRequest: input (url, question)
* QueryResponse: output (answer)

Ensures type-checking, validation, and clear structure.

**📂 services/**

**scraper.py**

* Uses LangChain's WebBaseLoader to scrape the provided URL
* Returns LangChain Document objects

**qa.py**

* Uses OpenAI/Perplexity API (via LangChain LLM) to answer the user’s question
* Input: documents + question
* Output: natural language answer

**🌐 Backend Config Files**

**.env**

Stores sensitive info:

env

CopyEdit

OPENAI\_API\_KEY=sk-xxxx

PERPLEXITY\_API\_KEY=pxy-xxxx

Accessed in qa.py using os.getenv()  
✅ Make sure it's in .gitignore

**.gitignore**

Tells Git to ignore:

bash

CopyEdit

.env

venv/

\_\_pycache\_\_/

\*.pyc

.vscode/

**requirements.txt**

All dependencies:

nginx

CopyEdit

fastapi

uvicorn

langchain

openai

beautifulsoup4

requests

python-dotenv

Install using:

bash

CopyEdit

pip install -r requirements.txt

**README.md**

Explain:

* What the project does
* How to set up
* How to run extension + backend
* Security notes

**🔁 Full End-to-End Flow**

1. User visits a webpage
2. Clicks **"Scrape This Page"** in the popup
3. Extension stores current tab URL
4. User enters a question
5. Extension sends URL + question to backend
6. Backend:
   * Scrapes content via LangChain
   * Feeds it + question to OpenAI/Perplexity
   * Sends back the answer
7. Extension displays answer in popup