Technical Architecture Document

Overview

The Jinnie MVP1 project is organized into two main parts: a **frontend Chrome extension** and a **Flask-based backend**. The frontend includes files like popup.html, popup.js, and contentScript.js, which handle the user interface, page scraping, and communication with the backend. The backend, built with Flask, receives the scraped content and user question, formats a prompt, and queries the DeepSeek LLM through the OpenRouter API. The overall architecture is designed to keep the frontend lightweight and reactive, while the backend handles all AI processing

Jinnie-MVP 1.0

```
├── frontend/                # Chrome Extension code
|   ├── manifest.json          # Chrome extension config
|   ├── popup.html             # Extension popup UI
|   ├── popup.js             # JS logic for popup (submit, fetch, feedback)
|   ├── contentScript.js       # Injected script to scrape webpage content

├── backend/                 # Flask backend server
|   ├── app.py               # Main Flask app (LLM prompt handling)
|   ├── requirements.txt        # Flask + requests + flask_cors
|   └── .env                 # API keys (OpenRouter key)
```

Flow Overview

Chrome Extension

- popup.html + popup.js: User interface (question input, response area, feedback)
- contentScript.js: Extracts visible text from the web page using DOM parsing
- manifest.json: Declares permissions, content scripts, and background setup

2. Message Bridge

- Uses chrome.runtime.sendMessage() and chrome.tabs.sendMessage() to connect:

    - Popup → Content script → Back to popup
- Ensures the popup can access webpage content (since popup scripts cannot directly access DOM)

3. Flask Backend Server
   - Receives scraped content and user question from the extension
   - Prepares a structured prompt for the LLM
   - Calls OpenRouter's API with:
     - Model: deepseek/deepseek-r1-0528-qwen3-8b:free
     - Prompt template:
       "Based only on the content below, answer this question in a short and clear way like you're talking to a friend…"
   - Returns the LLM response back to the frontend
4. OpenRouter LLM API
   - Cloud-hosted large language model
   - Accepts prompt via /chat/completions endpoint
   - Returns short, grounded answers in real time
5. Frontend Feedback Layer
   - Once the answer is displayed, users can rate it via Like / Dislike buttons

Data Flow Diagram

[User Input in popup.js]

↓

[Send message to contentScript.js]

↓

[Scrape page content from DOM]

↓

[Send content + question to Flask backend]

↓

[Flask formats prompt → sends to LLM API]

↓

[LLM responds → Flask returns response]

↓

[Answer shown in popup + feedback buttons enabled]