I want to use Cursor with CodeGPT, using my Locally hosted Deepseek R1 LLM, to write me a website. For this I need your help. Please write me the ideal prompt for the AI coding agent that will help make this an easy process. I am a novice, you are my expert advisor. You will guide me through the process and ask questions, while always making suggestions that seem appropriate to you. Here is what I have so far as a prompt for the coding agent. Please help me make this more concise and understandable to minimize rework.

1. Overview

I want to programme a website that will allow me and my collegues to create daily press clippings newsletter to be shared with the team. It will allow me to copy and paste URLs from various German, South African and major International newspapers and outlets as input. The app will the parse the data from the website and copy it onto the newsletter. This will happen once daily.

1. Architecture

The architecture should be as light as possible while still providing the usuability i require. First, you will create the architecture for the programme. It will be hosted on the internet on a server. You will use common, easily understandable components and reduce complexity to a minimum while still providing all the capablilities. You will write the documentation for both backend and frontend in plain and simple language that I can understand. You will use a lightweight database such as SQLite. For now, no user authentication will be needed. The database will be cleansed of all data after 24hrs as the news will be outdated by then. The user can chose to manually also purge the database manually once he is done with the creation of the newsletter.

1. INPUT

This will be URLs from various German (Sueddeutsche, FAZ, Spiegel, Tagesspiegel, DW, ARD, ZDF), South African (Daily Maverick, The Post) and major International newspapers and news outlets (such as BBC, CNN, AP, Washington Post, NYT) as input. The data will be parsed and will include

1. Headline
2. Excerpt, usually the first paragraph
3. Date
4. Source incl. direct link to the website
5. Category: The categories will be: International Politics, National Politics, Economics & Environment, Other.

When the data is being parsed, it will be displayed in a preview to the user and be editable. Once the category has been selected and the input data has been reviewed, the user will post the article to be included in the newsletter.

4) OUTPUT

The output will be a newsletter of article snippets, is a simple, clean, modern newspaper style format as either PDF or word document. The news snippets will be categorized into the above categories (International Politics, National Politics, Economics & Environment, Other). The user will have the option to add more categories. The user can then move the articles up and down to rearrange them as he sees fit, and also delete them.

Claude 3.5 Coding guideline

Create a web-based press clippings newsletter system following these specific implementation phases:

PHASE 1: CORE INFRASTRUCTURE Requirements: 1. HTMX Frontend Setup - Create single-page application structure - Implement minimal, clean UI - Setup URL input field with validation - Add category dropdown matching template categories - Create basic preview panel 2. Backend Foundation using Go - Setup SQLite database - Create core tables: \* articles (id, headline, content, source\_url, pub\_date, category, image\_url[optional]) \* daily\_collections (id, creation\_date, status) - Implement 24-hour data purge mechanism - Add manual purge endpoint 3. Basic Article Parser - URL validation system - Multi-language parser support (German, English, Afrikaans) - Extract: \* Headlines \* First paragraph \* Publication date \* Source URL - Basic error handling - Support for major news sources specified

PHASE 2: CONTENT MANAGEMENT Requirements: 1. Article Management - Preview interface for parsed articles - Edit capability for: \* Headlines \* Content \* Categories - Drag-and-drop article arrangement - Article deletion 2. Category System - Implement fixed categories: \* Foreign Politics \* Domestic Politics \* Economy, Energy, Climate & Agriculture \* Verschiedenes (Other) - Custom category addition - Category reordering

PHASE 3: NEWSLETTER GENERATION Requirements: 1. Template Integration - Implement provided template structure - Maintain consistent spacing - Format dates according to template - Category-based article grouping 2. Export System - Preview mode before export - PDF generation - Word document generation - Maintain template formatting in exports 3. Multi-language Support - UTF-8 encoding - Right-to-left text support - Language detection - Proper character rendering

PHASE 4: OPTIMIZATION & POLISH Requirements: 1. Performance Optimization - Image optimization - Database query optimization - Frontend rendering optimization - Caching implementation 2. Error Handling - Comprehensive error messages - Failed URL parsing recovery - Database error handling - Export error handling 3. UI/UX Improvements - Loading states - Success/error notifications - Responsive design - Accessibility features Technical Specifications: - Frontend: HTMX - Backend: Go - Database: SQLite - Image Processing: Standard web-safe formats - Export Formats: PDF, DOCX - Character Encoding: UTF-8 - Browser Support: Modern browsers only Development Guidelines: - Keep code modular and well-documented - Implement error handling at each step - Follow Go best practices - Use async/await for asynchronous operations - Implement proper input sanitization - Follow security best practices

**New prompt with more architecture:**

I want to create a lightweight web application to generate a daily press clippings newsletter for my team. The app will allow me to input URLs from various German, South African, and major international news outlets. It will parse the data from these URLs and organize it into a newsletter format. Please guide me through the process, ask clarifying questions, and make suggestions where appropriate.

Requirements

1. Overview

The app will parse data from URLs and extract:

Headline

Excerpt (first paragraph)

Date

Source (with a direct link)

Category (e.g., International Politics, National Politics, Economics & Environment, Other)

Parsed data will be displayed in a preview for user review and editing.

Users can assign categories, rearrange articles, and delete unwanted ones before finalizing the newsletter.

2. Architecture

Use a lightweight, simple architecture hosted on a server.

Backend: Python based

Database: Use a lightweight database like SQLite to store parsed data temporarily (cleansed automatically after 24 hours or manually by the user).

Frontend: Vue.js

No user authentication is required at this stage.

Provide clear, beginner-friendly documentation for both backend and frontend.

3. Input

URLs from the following sources:

German outlets: Süddeutsche, FAZ, Spiegel, Tagesspiegel, DW, ARD, ZDF

South African outlets: Daily Maverick, The Post

International outlets: BBC, CNN, AP, Washington Post, NYT

The app will parse and display the following fields:

Headline

Excerpt (first paragraph)

Date

Source (with a direct link)

Category (user-selectable, with the ability to add custom categories)

4. Output

Article will be parsed and then a preview window will appear where the user can finalze the pre-populated data before saving and posting the article to the press clipping frontend.

The final newsletter will be generated in a clean, modern newspaper-style format as a PDF and Word document. I will also upload the sample output file for reference.

Articles will be grouped by category and can be rearranged via drag and drop or deleted by the user

**Proposed Architecture**

**1. Frontend**

* **Framework/Library**: Use a lightweight framework like **React** or Vue.js for building the user interface. These are widely used, beginner-friendly, and have excellent documentation.
* **Features**:
  + Input form for pasting URLs.
  + Preview section for parsed articles (editable fields for headline, excerpt, category, etc.).
  + Drag-and-drop functionality for rearranging articles.
  + Simple buttons for deleting articles or finalizing the newsletter.
* **Styling**: Use a CSS framework like **Tailwind CSS** or **Bootstrap** for a clean, modern design with minimal effort.

**2. Backend**

* **Framework**: Use **Flask** (Python) for the backend. Both are lightweight, easy to set up, and well-suited for small projects.
* **Responsibilities**:
  + Handle URL parsing (fetching and extracting data from the provided URLs).
  + Store parsed data temporarily in the database.
  + Provide APIs for the frontend to interact with (e.g., fetching, updating, and deleting articles).
  + Generate the final newsletter in PDF or Word format.

**3. Database**

* **Database**: Use **SQLite** for simplicity and ease of use.
  + Stores parsed article data temporarily (cleansed automatically after 24 hours or manually by the user).
  + No need for a complex database system since the data is short-lived and lightweight.

**4. URL Parsing**

* **Library**: Use a web scraping library like **BeautifulSoup** (Python) to extract data from the provided URLs.
  + Extract headline, excerpt (first paragraph), date, and source link.
  + Use a fallback mechanism if some fields are missing or cannot be parsed.

**5. Newsletter Generation**

* **PDF/Word Generation**:
  + Use **WeasyPrint** (Python) for generating PDFs.
  + Use **python-docx** (Python) for generating Word documents.

**6. Hosting**

* **Server**: Host the application on a cloud platform like:
  + **Heroku** (beginner-friendly, free tier available).
* **Domain**: Use a custom domain for easy access (optional).

**7. Workflow**

1. **Input**: User pastes URLs into the frontend.
2. **Backend Processing**:
   * Fetch the webpage content.
   * Parse the required fields (headline, excerpt, date, source).
   * Store the parsed data in the SQLite database.
3. **Preview**: Display parsed data in the frontend for user review and editing.
4. **Newsletter Creation**:
   * User finalizes the articles (assigns categories, rearranges, deletes).
   * Backend generates the newsletter in the selected format (PDF or Word).
5. **Output**: User downloads the newsletter.

**Why This Architecture?**

1. **Lightweight**: The combination of React/Vue.js, Flask/Express.js, and SQLite keeps the architecture simple and efficient.
2. **Scalable**: While lightweight, this architecture can be scaled up if needed (e.g., adding user authentication or more complex features in the future).
3. **Beginner-Friendly**: The tools and frameworks suggested are well-documented and widely used, making it easier for you to learn and maintain the application.
4. **Cost-Effective**: Hosting on platforms like Heroku or Vercel is free or low-cost for small projects.