Code Surge – Dynamic Content Discovery and Data Extraction

Project Overview:

In the evolving landscape of modern web development, serverless architectures, and React-based frontends have become increasingly prevalent. As a developer, having quick access to relevant documentation is crucial. Your challenge is to develop a sophisticated web scraping system that will extract and structure documentation from both AWS Lambda and React, preparing it for use in a local Retrieval-Augmented Generation (RAG) system.

Your Challenge:

Imagine you're part of a development team building a serverless application using AWS Lambda for the backend and React for the frontend. To streamline the development process, you need to create a local knowledge base that team members can query using an LLM-powered system (to be implemented later). Your task is to build the foundation by scraping and structuring the relevant documentation.

Scope: Scrape the following sections under each documentation's side menu:

1. **React Documentation Extraction**
    - Source URL: https://react.dev/learn
    - Target Sections: Quick Start
        i. Installation
        ii. Describing the UI
        iii. Adding Interactivity
        iv. Managing State
        v. Escape Hatches

2. **AWS Lambda Documentation Extraction**
    - Source URL: https://docs.aws.amazon.com/lambda/latest/dg/welcome.html
    - Target Sections:
        i. What is AWS Lambda?
        ii. Example apps
        iii. Building with TypeScript
        iv. Integrating other services
        v. Code examples

Task Requirements:

1. **Scraping Implementation:**
    - Navigation through documentation hierarchies
    - Handling of dynamic content and JavaScript-rendered pages
    - Processing of code snippets while preserving formatting
    - Management of relative links and resources
    - Respectful scraping practices
2. **Content Processing:**
    - Extract meaningful chunks of documentation content
    - Creative structured formatting of content
3. **Output Format:**
    - Combined one JSON file for both documentation

```
[{
  "title": "page_title",
  "source": "aws_lambda|react",
  "url": "original_url",
  "sections": [],
}]
```

4. **Code Quality**
    - Write clean, well-documented, and modular code following Python best practices.
    - Include error handling to manage potential issues such as network errors, missing elements, or unexpected content structures.
    - Provide a README file with clear instructions on how to run the script, including any dependencies and setup steps.

Marking Criteria

1. **Code Functionality & Accuracy (25%):**
    - The script successfully scrapes all specified sections and subsections.
    - Extracted content accurately reflects the original documentation without omissions or errors.

2. **Content Processing & Output Quality (20%):**
    - Unnecessary elements are effectively removed, resulting in clean and readable Markdown files.
    - The logical structure and hierarchy of the original documentation are preserved.

3. **Efficiency & Performance (20%):**
    - The script performs efficiently, minimizing runtime and resource usage.

4. **Code Quality & Best Practices (15%):**
    - Appropriate use of functions, classes, and modules enhances readability and maintainability.
    - Comprehensive error handling is implemented.

5. **Documentation & Usability (10%):**
    - Appropriate use of functions, classes, and modules enhances readability and maintainability.
    - Comprehensive error handling is implemented.

6. **Presentation Video(10%)**
    - The video should contain a small part of a web scrape. (You can scrape a few pages around 10 seconds) and need to add some code explanation in the video, the video should be less than 1 minute