Project 2: Web Content Analyzer

Objective:

Build a web application that extracts content from any website URL and generates a comprehensive analysis report using LLM in just 2 days. This accelerated timeline leverages your Project 1 experience for rapid development. You will practice:

- Web Scraping: Extracting content from websites using Python
- Content Processing: Cleaning and structuring scraped data
- LLM Integration: Using AI to analyze and summarize content
- API Development: Building more complex backend endpoints

Core Requirements (Must-have)

Layer	Requirement
Backend	Python 3.11 + FastAPI POST /analyze endpoint: {"url": "https://example.com"} → {"analysis": "detailed report"} Web scraping with requests + BeautifulSoup Content extraction: title, headings, main text, links LLM integration for content analysis Error handling for invalid URLs and scraping failures
Frontend	Streamlit - Perfect for data analysis and reporting Use st.text_input() for URL input Use st.button() for analysis trigger Use st.expander() and st.columns() For report layout Use st.progress() for processing indicators
Content Processing	Extract meaningful content (skip navigation, ads, etc.) Limit content size to prevent API limits Structure data for LLM analysis Handle different website layouts

Development Approach: Milestone-Based Progression

Focus on deliverable quality and comprehensive review compliance. Each milestone must pass all relevant review templates.

Milestone 1: Web Scraping Foundation & Data Extraction

Deliverables:

- Working development environment with web scraping dependencies
- BeautifulSoup-based content extraction service
- URL validation and error handling system
- Basic Streamlit interface for URL input and results display
- Content preprocessing and cleaning pipeline

Review Requirements (Must Pass to Proceed):

- Security Review: Input validation, URL sanitization, no SSRF vulnerabilities
- Code Quality Review: Clean separation of scraping logic and UI
- Performance Review: Efficient content extraction and memory management

Milestone 2: LLM Integration & Analysis Engine

Deliverables:

- LLM integration for content analysis and report generation
- Structured analysis pipeline with content preprocessing
- Comprehensive error handling for analysis failures
- Enhanced UI with progress indicators and analysis results
- Content size optimization and intelligent truncation

Review Requirements (Must Pass to Proceed):

- AI Integration Review: Optimal prompt engineering and content analysis
- Performance Review: Response times and content processing efficiency
- Security Review: Content sanitization and safe processing

Milestone 3: Production Features & Advanced Analysis

Deliverables:

- Advanced report formatting and data visualization
- Batch processing capabilities for multiple URLs
- Export functionality (PDF, JSON, CSV)
- Comprehensive documentation and testing procedures
- Production-ready deployment preparation

Review Requirements (Must Pass for Project Completion):

- Architecture Review: Scalable design and component separation
- Security Review: Complete security assessment including SSRF prevention
- AI Integration Review: Production-ready analysis pipeline
- Code Quality Review: Final code quality and documentation standards

Milestone 4: Context-Aware Analysis with RAG (Optional Extension)

Deliverables:

- Vector Database Setup: Implement and configure a vector database (e.g., ChromaDB, FAISS).
- Knowledge Base Ingestion: Create a pipeline to populate the knowledge base with relevant documents (e.g., industry reports, competitor data).
- RAG Retrieval Service: Build a service that retrieves relevant context from the knowledge base based on the URL being analyzed.
- Augmented Analysis Pipeline: Integrate the RAG service into the main analysis workflow to provide enriched context to the LLM.
- Comparative Reporting: Enhance the UI to display comparative insights generated by the RAG-powered analysis.

Review Requirements (Must Pass for RAG Completion):

- AI Integration Review: Assess retrieval relevance, context quality, and enriched prompt effectiveness.
- Architecture Review: Evaluate the RAG pipeline design and its integration with the existing system.
- Performance Review: Measure the performance overhead of the retrieval step.

Technical Specifications

API Endpoint Structure

- Build a POST /analyze endpoint that accepts URLs and returns structured analysis reports
- Implement web scraping logic to extract meaningful content from various website layouts

Content Extraction Strategy

- Use BeautifulSoup to target main content elements while filtering out navigation, ads, and boilerplate
- Implement content size limits and intelligent truncation to handle large websites

LLM Analysis Integration

- Design prompts that generate structured reports covering company overview, services, and key highlights
- Handle API rate limits and content size restrictions for optimal LLM processing

Project Structure

- Organize code with separate modules for Streamlit UI, web scraping, and LLM analysis

- Use clean separation between data extraction, processing, and presentation layers

Stretch Goals (Nice-to-have)

- Multiple URL Analysis: Analyze and compare multiple websites
- Export Functionality: Download analysis as PDF or Word document
- Analysis History: Store and retrieve previous analyses
- Advanced Scraping: Handle JavaScript-heavy sites with Selenium
- Content Categories: Detect and categorize different types of websites

Deliverables

1. GitHub Repository Link (public or invite @mentor)
2. Live Demo with 3+ different website analyses
3. ANALYSIS_SAMPLES.md - Include:
    - Sample analyses of different website types (company, blog, news)
    - Screenshots of the application
    - List of tested URLs and results
4. Technical_Learnings.md

Measurable Goals & Review Template Compliance

Primary Objectives

- Web Scraping Excellence on diverse website scraping
- Security Compliance: Pass Security Review (SSRF prevention critical)
- AI Integration Quality: Pass AI Integration Review
- Performance Standards: Process websites within 30 seconds, handle large content efficiently
- Code Quality Standards: Pass Code Quality Review with 8/10+ score

Review Template Integration (All Must Pass)

➢ Security Review Requirements (Critical for Web Scraping)

- ➢ AI Integration Review Requirements
- ➢ Performance Review Requirements

Performance Standards

- Scraping Success on diverse websites (news, blogs, corporate sites)
- Processing Time: Average < 30 seconds for standard web pages
- Content Quality: Extract meaningful content while filtering noise
- Memory Efficiency: Handle large websites without memory issues
- Error Rate: < 10% failed scraping attempts due to code issues

Task Tracking & Project Management Integration

Project 2 - Web Content Analyzer

Epic ID: P2-ANALYZER
Priority: High
Dependencies: Project 1 completion

Milestone 1: Web Scraping Foundation & Data Extraction

Feature 1.1: Web Scraping Infrastructure

Task ID: P2-M1-SCRAPING
Priority: Critical
Dependencies: None

Sub-tasks:

- P2-M1-SCRAPING-01: Setup BeautifulSoup scraping service
  - Description: Create web scraping service with proper error handling
  - Acceptance Criteria:
    - BeautifulSoup integration working
    - URL validation implemented

        - Basic content extraction functional
- P2-M1-SCRAPING-02: Implement content cleaning pipeline
    - Description: Extract meaningful content while filtering noise
    - Acceptance Criteria:
        - Remove navigation, ads, boilerplate
        - Extract title, headings, main content
        - Handle different website layouts
- P2-M1-SCRAPING-03: Add security and validation
    - Description: Implement SSRF prevention and input validation
    - Acceptance Criteria:
        - URL whitelist/blacklist functionality
        - Private IP address blocking
        - Content size limits

Feature 1.2: Basic Analysis Interface

Task ID: P2-M1-UI
Priority: High
Dependencies: P2-M1-SCRAPING

Sub-tasks:

- P2-M1-UI-01: Create URL input interface
    - Description: Streamlit interface for URL input and basic results
    - Acceptance Criteria:
        - URL input field with validation
        - Progress indicators during scraping
        - Basic results display
- P2-M1-UI-02: Implement error handling UI
    - Description: User-friendly error messages and recovery
    - Acceptance Criteria:
        - Clear error messages for failed scrapes
        - Retry mechanisms
        - Input validation feedback

Milestone 2: LLM Integration & Analysis Engine

Feature 2.1: LLM Analysis Service

Task ID: P2-M2-LLM-SERVICE
Priority: Critical
Dependencies: P2-M1-SCRAPING

Sub-tasks:

- P2-M2-LLM-01: Integrate LLM for content analysis
  - Description: Connect to an LLM API to generate analysis reports from scraped text.
  - Acceptance Criteria:
    - LLM API returns a structured analysis.
    - Proper error handling for API failures.
- P2-M2-LLM-02: Develop prompt engineering strategies
  - Description: Design and refine prompts to guide the LLM for high-quality, structured output.
  - Acceptance Criteria:
    - Prompts produce consistent reports covering key areas.
    - Content is summarized effectively.
- P2-M2-LLM-03: Implement content size management
  - Description: Add logic to truncate or chunk content to fit within LLM context limits.
  - Acceptance Criteria:
    - Large content is handled without API errors.
    - Truncation strategy preserves key information.

Feature 2.2: Enhanced Frontend for Analysis

Task ID: P2-M2-UI
Priority: High
Dependencies: P2-M2-LLM-SERVICE

Sub-tasks:

- P2-M2-UI-01: Display LLM analysis report
    - Description: Integrate the analysis report into the Streamlit UI.
    - Acceptance Criteria:
        - Report is displayed in a user-friendly format (e.g., st.expander).
        - Loading indicators are shown during analysis.
- P2-M2-UI-02: Add error handling for analysis failures
    - Description: Show clear error messages if the LLM analysis fails.
    - Acceptance Criteria:
        - User is notified of API errors or timeouts.
        - Guidance for retrying is provided.

Milestone 3: Production Features & Advanced Analysis

Feature 3.1: Advanced Reporting and Export

Task ID: P2-M3-REPORTING
Priority: High
Dependencies: P2-M2-UI

Sub-tasks:

- P2-M3-REPORTING-01: Implement advanced report formatting
    - Description: Use markdown and charts to visualize the analysis.
    - Acceptance Criteria:
        - Reports are well-structured and visually appealing.
- P2-M3-REPORTING-02: Add report export functionality
    - Description: Allow users to download the analysis report as a PDF or JSON file.
    - Acceptance Criteria:
        - Export buttons for PDF and JSON are functional.

Feature 3.2: Production Readiness

Task ID: P2-M3-PROD
Priority: Critical
Dependencies: P2-M3-REPORTING

Sub-tasks:

- P2-M3-PROD-01: Write comprehensive tests
  - Description: Implement unit and integration tests for the application.
  - Acceptance Criteria:
    - Core logic is covered by tests.
    - CI pipeline passes.
- P2-M3-PROD-02: Create project documentation
  - Description: Write a detailed README and deployment guide.
  - Acceptance Criteria:
    - README includes setup, run, and deployment instructions.

Milestone 4: Context-Aware Analysis with RAG

Feature 4.1: RAG Pipeline Implementation

Task ID: P2-M4-RAG-PIPELINE
Priority: High
Dependencies: P2-M2-LLM-SERVICE

Sub-tasks:

- P2-M4-RAG-01: Setup vector database
  - Description: Configure and integrate a vector database like ChromaDB.
  - Acceptance Criteria:
    - Vector database is connected and accessible.
- P2-M4-RAG-02: Build knowledge base ingestion pipeline
  - Description: Create a script to process and load documents into the vector DB.
  - Acceptance Criteria:
    - Documents are successfully converted to embeddings and stored.
- P2-M4-RAG-03: Create context retrieval service
  - Description: Build a service to query the vector DB for relevant context.

- Acceptance Criteria:
  - Service returns relevant document chunks based on input queries.

Feature 4.2: Augmented Analysis and UI

Task ID: P2-M4-RAG-INTEGRATION
Priority: High
Dependencies: P2-M4-RAG-PIPELINE

Sub-tasks:

- P2-M4-RAG-04: Integrate RAG into analysis workflow
  - Description: Augment the LLM prompt with context retrieved from the RAG service.
  - Acceptance Criteria:
    - LLM receives and uses the additional context.
- P2-M4-RAG-05: Enhance UI for comparative analysis
  - Description: Update the UI to display insights from the RAG-powered analysis.
  - Acceptance Criteria:
    - UI clearly distinguishes between standard and RAG-augmented results.

Testing Websites

Test your analyzer with these different types of websites:

- Corporate: https://www.amzur.com, https://www.microsoft.com , https://www.apple.com
- E-commerce: https://www.amazon.com , https://www.shopify.com
- News: https://www.bbc.com , https://techcrunch.com
- Blog: https://medium.com , https://dev.to
- Educational: https://www.coursera.org , https://www.edx.org
- Google news: https://www.technologyreview.com/feed

- TechCrunch: https://techcrunch.com/feed/
- MIT Technology Review: https://www.wired.com/feed/rss
- VentureBeat: https://venturebeat.com/category/ai/feed/
- ZDNet: https://www.zdnet.com/topic/artificial-intelligence/rss.xml

Quick Start Resources

- BeautifulSoup Documentation: https://www.crummy.com/software/BeautifulSoup/bs4/doc/
- Requests Library: https://docs.python-requests.org/
- Web Scraping Guide: https://realpython.com/beautiful-soup-web-scraper-python/
- FastAPI Background Tasks: https://fastapi.tiangolo.com/tutorial/background-tasks/

Success Criteria Checklist

- Can successfully scrape content from 80%+ of tested websites
- Generates meaningful analysis reports using LLM
- Handles errors gracefully (network issues, invalid URLs)
- User-friendly interface with loading states
- Clean, documented code with proper structure
- Comprehensive testing with various website types