Aditya Gadagandla 22WU0104139

Report: Documentation Analyzer Agent

Github link: https://github.com/adiii1701/MoEngage

Overview

This project delivers a production-ready **AI-powered Documentation Analyzer Agent** designed to evaluate documentation against four key improvement criteria:

1. **Readability**
2. **Structure & Flow**
3. **Completeness**
4. **Style Guidelines Compliance**

The implementation includes:

- Automated HTML scraping
- Hybrid NLP + LLM evaluation
- Actionable JSON-based output
- Error and rate-limit handling
- Integration with the Anthropic Claude API

Core Functionalities & Analysis Criteria

1. Readability for Non-Technical Marketers

- **Methodology**:
  - Uses **Flesch-Kincaid** and **Gunning Fog** readability scores
  - Augments analysis with **Claude LLM assessment** focused on the "non-technical marketer" persona
- **LLM Insights**:
  - Identifies jargon
  - Flags long or convoluted sentences
  - Provides rewrite suggestions
- **Output Example**:

```
"readability": {
  "flesch_kincaid_grade": 10.2,
  "assessment": "Some technical terms need simplification...",
  "suggestions": [
    "Break down the 89-word paragraph in section 3...",
    "Replace 'SDK integration methodology' with simpler
phrasing..."
  ]
}
```

2. Structure & Flow

- **Evaluates**:
  - Heading hierarchy (h1-h4)
  - Paragraph count and length
  - List usage
  - Navigation ease
- **Analysis Engine**:
  - Extracts content blocks using fallback selectors for HTML parsing
  - Calculates average paragraph length, heading depth, and structure clarity
- **LLM Role**:
  - Comments on information flow and whether sections are logically ordered
- **Output Example**:

```
"structure": {
  "heading_count": 8,
  "paragraph_count": 15,
  "avg_paragraph_length": 67.3,
  "suggestions": [
    "Add a 'Troubleshooting' section at the end...",
    "Include a 'Prerequisites' section before implementation
steps"
  ]
}
```

3. Completeness of Information & Examples

- **Heuristics Used**:
  - Presence of code blocks
  - Example count
  - Section titles indicating explanation depth
- **LLM Evaluation**:
  - Identifies gaps in implementation details or edge case handling
  - Suggests where additional examples or clarifications would help
- **Output Example**:

```
"completeness": {
  "code_examples_count": 2,
  "assessment": "Lacks explanation on error handling and edge
cases.",
  "suggestions": [
    "Add response format examples for each endpoint",
    "Include error handling use case with sample output"
  ]
}
```

4. Style Guidelines Compliance

- **Reference**: Focused implementation of the **Microsoft Style Guide**, particularly:
    - **Voice & Tone**: Customer-centric, active voice
    - **Clarity & Conciseness**: Eliminate redundancy
    - **Action-Oriented Language**: Guide readers directly
- **LLM Role**:
    - Flags passive voice, convoluted phrasing, or vague titles
    - Suggests specific rewrites
- **Output Example**:

```
"style_guidelines": {
  "assessment": "Several instances of passive voice and
unclear instructions.",
  "suggestions": [
    "Replace 'Events can be tracked' → 'Track events'",
    "Use clearer headings: 'API Usage' → 'Use the API to Send
Events'"
  ]
}
```

Technical Implementation Details

Component	Description
Language	Python 3.8+
LLM API	Anthropic Claude (Sonnet) – Optimized prompts for each criterion
Web Scraping	HTML parsing with fallback selectors for dynamic doc structure
Error Handling	Graceful exception management for scraping, API limits, network errors
Rate Limiting	Delay built-in between API calls (1-second sleep)
Output Format	Structured JSON with individual assessments, scores, and specific suggestions

Testing and Usage

- `example_usage.py`: Demonstrates standard usage of the tool
- `test_analyzer.py`: Includes mock tests to validate pipeline components
- `sample_outputs.json`: Contains example reports with metrics and LLM insights
- `setup.sh`: Automates environment setup and dependency installation

Setup Instructions

```
# Set API key
export ANTHROPIC_API_KEY="your-api-key-here"

# Install dependencies
pip install -r requirements.txt

# Run analysis
python doc_analyzer.py "https://help.moengage.com/hc/en-
us/articles/your-url"
```

Design Considerations

Decision	Rationale
Hybrid scoring + LLM	Combines quantitative and qualitative strengths
Marketer Persona	Custom LLM prompts ensure relevance for non-technical users
Output structure	JSON enables integration with downstream revision agents
Focused Style Guide Use	Prioritized practicality over full adherence

Challenges Faced & Solutions

Challenge 1: Dynamic Content Extraction

Problem: MoEngage documentation may have varying HTML structures across different pages.

Solution: Implemented multiple fallback selectors and robust parsing logic to handle different page layouts.

Challenge 2: Balancing Depth vs. Performance

Problem: Comprehensive analysis requires multiple LLM calls, which can be slow and expensive.

Solution: Optimized prompt design for efficient token usage and implemented rate limiting for stability.

Challenge 3: Actionable Suggestion Generation

Problem: LLM responses can be verbose and generic rather than specific and actionable.

Solution: Designed focused prompts that request specific improvements and implemented suggestion extraction logic.

Challenge 4: Readability for Specific Persona

Problem: Standard readability metrics don't account for the specific "non-technical marketer" persona.

Solution: Combined algorithmic scores with persona-specific LLM analysis for more relevant insights.

Future Improvements

1. Content Type Detection: Automatically adjust analysis criteria based on content type (tutorial, reference, overview)

2. Competitive Analysis: Compare against industry best practices and competitor documentation

3. Visual Element Analysis: Analyze images, diagrams, and formatting elements

4. Interactive Examples: Detect and evaluate interactive code examples or demos

5. Multi-language Support: Extend analysis to documentation in multiple languages

6. Historical Tracking: Track improvements over time and measure impact

7. Integration Capabilities: API endpoints for integration with documentation management systems

Ready for Future Extensions

- **Document Revision Agent (Task 2)** ready to build on modular JSON output
- Pluggable criteria modules allow for adding new checks (e.g., SEO, accessibility)
- Can be adapted to other domains with minimal changes

Deliverables Summary

File	Purpose
`doc_analyzer.py`	Core analyzer module
`requirements.txt`	Dependency list
`README.md`	Setup, usage, and methodology documentation
`example_usage.py`	Demonstration of usage
`test_analyzer.py`	Test cases and validation
`setup.sh`	Quickstart script
`sample_outputs.json`	Example JSON reports