Content Generation Workflow

When a user clicks "Generate Content" on the dashboard, the following process occurs:

1. Input Collection & Validation

- User selects an **AI model** (GPT-4, Deepseek, etc.)
- User selects an **API key** from their saved keys
- User selects a **prompt template** from their organization
- User enters a **batch name** for organization
- User toggles **web scraping** on/off
- User enters **word count** (if web scraping is disabled)
- User can enter **custom outline** and **section prompts** (optional)
- User enters **keywords** (via text area or CSV upload)

2. Form Processing

- For CSV input: Parses primary and secondary keywords
- For text area: Splits keywords by new lines
- Constructs payload for each keyword

3. API Request Processing

- POST request to /api/articles endpoint
- Creates new Batch document in database
- For each keyword, creates Keyword document with initial status "not started"
- Determines prompt type (single-step vs multi-step)

4. Queue Job Creation (for multi-step prompts)

- Retrieves prompt details from database
- Prepares outline and section prompts with keyword substitution
- Adds job to longArticlesQueue (REDIS QUEUE) with parameters:

    - outlinePrompt
    - sectionPrompt
    - API (key)
    - GPT (model)
    - secondaryKeywords
    - keyword
    - customOutlinePrompt
    - customSectionPrompt
    - _id (of the Keyword document)

5. Queue Processing

- Worker picks up job from Redis queue
- Updates Keyword status to "in progress"
- Initiates content generation process

Steps 6 – 10 is essentially the business logic behind silent content right now in producing a single article using keywords, combining scraping (google search api + cheerios), content analysis (done using NLP techniques), Outline + Section Prompting (using LLM) and then Assembling all the content in html tags.

6. Web Scraping Process

The web scraping stage forms the research foundation for AI-generated content:

- Search Engine Integration: The system queries Google's Custom Search API with the target keyword
- Result Collection: Top 10 search results are retrieved for comprehensive topical coverage
- Targets Competitive Content: By analyzing top-ranking pages for the target keyword
- Content Extraction: For each search result:
    - HTML is parsed using Cheerio
    - Structural elements (headings, paragraphs, lists) are identified
    - Content is extracted and categorized by element type

    - Three separate collections are created:
        - Raw article content with HTML structure
        - Clean text content without markup
        - Heading-specific content for outline inspiration

This stage effectively collects competitive content that's already ranking for the target keyword, providing a research base for the AI.

7. Content Analysis Process

The analysis stage transforms raw scraped content into actionable insights:

- Text Processing:
    - Content is tokenized into individual words
    - Extensive filtering removes noise (stopwords, short words, numbers)
    - Remaining words are lemmatized (reduced to base forms)
- Keyword Extraction:
    - TF-IDF algorithm identifies statistically significant terms
    - Terms are ranked by relevance to the topic
    - Top 20 terms become primary keywords
    - Next 30 terms become secondary keywords
- Context Building:
    - Keywords are consolidated across all analyzed articles
    - This creates a comprehensive semantic map of the topic

This analytical approach ensures the generated content addresses the same semantic territory as top-ranking content.

Actual NLP processing at this stage: (programmatically) (easy comments on each line)

```
    const preprocessedTokens = tokens
      .map(token => token.toLowerCase()) // Lowercase all tokens
      .filter(token => token.length >= minWordLength) // Remove short words
      .filter(token => !stopwords.has(token.toLowerCase())) // Remove stopwords
      .filter(token => !numericRegex.test(token)) // Remove numerics
      .filter(token => !excludedKeywordRegex.test(token)) // Remove keywords
with numbers
```

```
        .map(token => token.replace(/[.,\/#!$%\^&\*;:{}=\-_`~()]/g, '')); //
Remove punctuation



        // Custom lemmatization function
        const customLemmatizer = token => {
          if (token.endsWith('ing') || token.endsWith('ed')) {
            // Likely a verb
            return winkLemmatizer.verb(token);
          } else if (token.endsWith('s')) {
            // Likely a noun
            return winkLemmatizer.noun(token);
          } else {
            // Default to verb as a general case
            return winkLemmatizer.verb(token);
          }
        };

        // Lemmatize tokens after preprocessing
        const lemmatizedTokens = preprocessedTokens.map(customLemmatizer);


    // TF-IDF keyword extraction
    const tfidf = new TfIdf();
    tfidf.addDocument(lemmatizedTokens);

    // Get the list of unique words
    const uniqueWords = Array.from(new Set(lemmatizedTokens));

    // Calculate the TF-IDF scores for each word
    const keywordScores = {};
    uniqueWords.forEach(word => {
      keywordScores[word] = tfidf.tfidf(word, 0); // Use 0 as the document index
    });

      // Sort keywords by TF-IDF score and select primary and secondary keywords
      const sortedKeywords = Object.entries(keywordScores)
        .sort((a, b) => b[1] - a[1]);

        // console.log('sortedKeywords', sortedKeywords);

      // const allKeywords = sortedKeywords.slice(0, 100).map(([word, _]) =>
word);
```

```
      const primaryKeywords = sortedKeywords.slice(0, 20).map(([word, _]) =>
word);
      const secondaryKeywords = sortedKeywords.slice(20, 50).map(([word, _]) =>
word);
```

8. Outline Generation Process

The outline generation stage creates the content structure:

- AI Prompt Construction:
    - Scraped heading data is formatted as JSON
    - Target keyword is included for context
    - Extracted keywords provide topical guidance
    - Custom outline prompt is incorporated if provided
- AI Response Processing:
    - AI generates a structured JSON outline
    - Each section includes:
        - Strategic heading names based on competitive analysis
        - Appropriate heading levels (h2-h4) for content hierarchy
        - Detail rating (1-5) to control section depth

This ensures the content structure mirrors successful competitor content while maintaining SEO optimization.

Actual prompt at this stage given to LLM:

```
jsonObjectString += `\n
These JSON objects detail the structure of multiple articles. Use this
information, along with the provided keywords ${allKeywordsStr}, to generate a
comprehensive outline for an article about ${keyword}. Ensure the content is
engaging and adheres to current SEO best practices.

  Format the response as:
  {
    "word_count": [optimal total word count],
    "sections": [
      {
```

```
      "heading_name": "[Derived from JSON data]",
      "heading_level": "h2",
      "details": [detail level between 1 and 5]
    },
    ...
  ]
} ${customOutlinePrompt}`;
```

9. Section Generation Process

The section generation stage builds the actual content:

- Incremental Content Creation:
  - Each heading becomes its own generation task
  - Section prompts include:
    - The heading title
    - HTML heading level for proper formatting
    - Detail level specification (determines section length)
    - Primary and secondary keywords for topical relevance
    - Any custom section prompt provided by the user
- Resilient Processing:
  - Robust error handling with exponential backoff retry logic
  - Each section is added to the growing article
  - HTML formatting is preserved for proper presentation

Actual Prompt at this stage given to LLM:

```
for (let [index, heading] of headings.entries()) {
  const wrapperPrompt = `Write a section about '${heading.heading_name}' using
<h${heading.heading_level}> for the title. Ensure the content is thorough and
reflects a detail level of '${heading.details}' out of 5, formatted with
appropriate HTML tags. Use both primary and secondary keywords from the provided
list.

  ${sectionPrompt}. For more secondary keywords you can use ${secondaryKeywords
|| ""}, only using those which are necessary.`;
```

10. Content Assembly

- Combines all generated sections
- Formats with proper HTML tags
- Ensures keyword distribution throughout the article

11. Database Update

- Updates Keyword document with generated content
- Sets status to "completed"
- Stores image URL if generated

12. Error Handling

- Updates Keyword status to "failed" with error message
- Implements retry logic with exponential backoff

13. Completion

- Job is removed from queue
- User can view completed article in the batches section
- Content is available for WordPress publishing