Newsaic

A Personalized News Application

Mehdi Abid Mohamed Amor
Ahmed Louay Araour Mohamed Aziz Belkahla
Donia Ben Othmen Ilef Hcine Rania Souei

November 11, 2025

Table of contents

Executive Summary

Newsaic is a personalized news application designed to improve engagement, accessibility, and relevance in news consumption. Its main attraction is delivering daily story briefings, crafted from related articles and tailored to each user's preferences, leveraging LLMs to produce coherent, readable stories.

Users are presented a default "For You" feed (based on their chosen sections) and can also access the separate Daily Highlights briefing that distills the top stories of the day.

Key technical components include **Kimi K2 Instruct 0905** for story generation (inferred via **Groq**), **Qwen-3 Embedding 0.6B** (via **Ollama**) for semantic understanding, and **LangChain** for model orchestration and text processing. **Kokoro-82M** provides high-quality audio narration.

The platform also offers semantic search, a Q&A module for deeper article comprehension, audio narration for accessibility, and bookmarking for saving content. Screenshots illustrate these functionalities in action. Future improvements could include persona-based TTS, additional customization for top stories, and a more intelligent, personalized news feed.

Introduction

News consumption can be challenging due to the sheer volume of information, difficulties in identifying relevant content, and accessibility barriers. There is considerable potential to improve how readers engage with news, helping them find and understand stories more effectively.

Recognizing these challenges, our team chose to explore ways to make news more engaging, relevant, and accessible. We focused on understanding the difficulties readers face, experimenting with new presentation formats, and exploring tools that could help surface important stories more effectively, while keeping personalization and clarity in mind.

1 Design Thinking Approach

To guide our development of Newsaic, we followed a design thinking methodology. This allowed us to focus on understanding users' needs, clearly defining the problems, generating creative ideas, testing potential solutions, and iterating based on feedback. Each phase helped ensure that the application would be both useful and engaging, while addressing real challenges faced by news readers.

1.1 Empathize

During the empathy phase, we focused on understanding the needs and frustrations of news readers. We found that many users experienced emotional fatigue or negative feelings from reading news, struggled with information overload, and often found articles irrelevant or too complex. Users also expressed distrust in news sources and a desire for clarity, simplicity, and personal relevance. To better understand these challenges, we considered example users such as a young person interested in scientific news who lacked familiarity with technical terms, a user trying to follow local politics in a foreign country, and a busy university professor needing concise, reliable news. These insights guided our focus on making news both personalized and accessible.

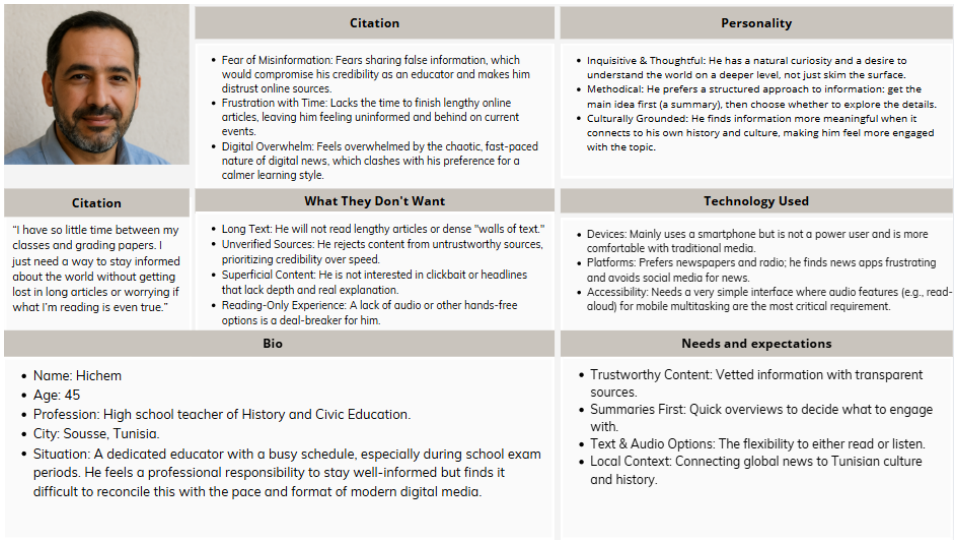

Figure 1.1: Example author persona used for story rewriting: Hichem

1.2 Define

From our empathizing work, we defined the main problem as a combination of content overload, lack of personal relevance, and difficulty understanding or engaging with traditional news formats. Our goal became creating a news application that delivered content in a way that was engaging, simplified, and customized to the user's preferences, while also addressing accessibility concerns.

1.3 Ideate

Our initial brainstorming generated a wide range of ideas. We considered delivering news in alternative formats, such as podcasts, short-form content style stories, or informal chat-style summaries. We also explored providing multiple author personas for users to select from, comment sections with opposing viewpoints, customizable algorithms based on keywords, and conversational interfaces for questions about the news. Audio narration for accessibility and busy users was also proposed.

After evaluating feasibility and alignment with our goals, we decided to exclude the alternative content formats and focus solely on the persona-based story

rewriting. This decision, made during the ideation phase, helped simplify the scope while keeping the core personalization feature central.

1.4 Prototype

As we moved to prototyping, we tested technical and design assumptions. Early prototypes highlighted several challenges: allowing multiple personas and generating every article for each persona would be costly and create interface complexity, while rewriting every article for every user was too resource-intensive due to LLM usage limits. We experimented with showing one active persona at a time and providing the ability to switch personas globally, but ultimately simplified to a single persona to reduce cognitive overload and computational cost.

During this phase, we also developed the concept of combining multiple related articles into a single story. This approach condensed information, reduced the number of articles users needed to read, and addressed the overload problem. Prototyping revealed that limiting the number of generated stories per day would not only manage technical constraints but also improve user experience by emphasizing the most relevant content.

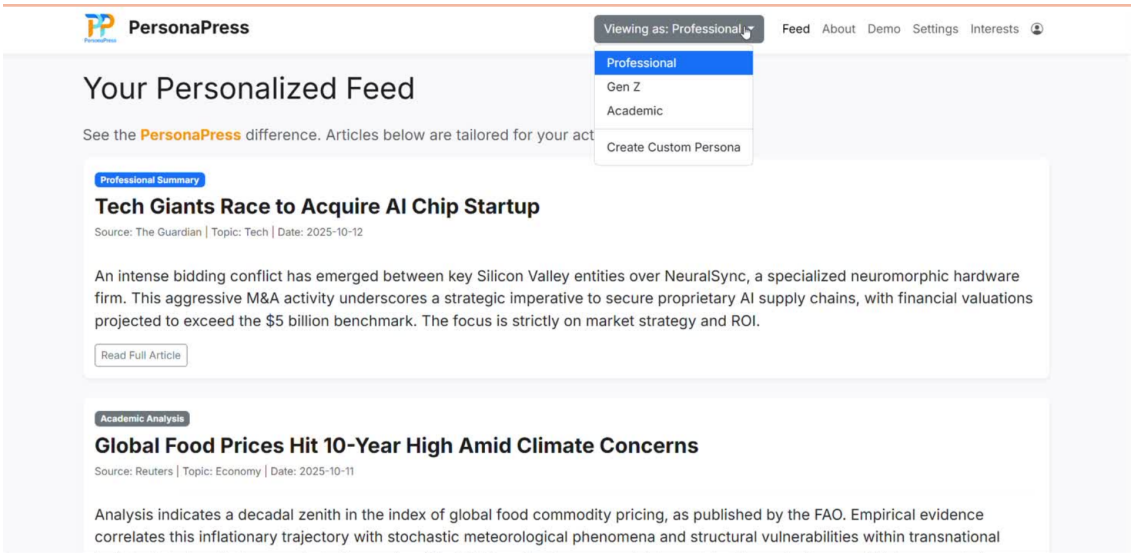

Figure 1.2: Initial prototype exploring a persona toggle and story briefings

1.5 Test

Through testing, we validated the top stories feature by generating a limited number of combined articles rewritten in a consistent author persona, confirming it reduced overload, improved clarity, and addressed key pain points from the empathy phase.

2 Functionalities

2.1 For You Feed

The For You feed is the primary view for users: it surfaces articles from the user's preferred sections and ranks them for relevance. From the feed users can bookmark items, open article detail pages, run semantic search across the archive, view related-article recommendations, and use the Q&A module on any opened article.

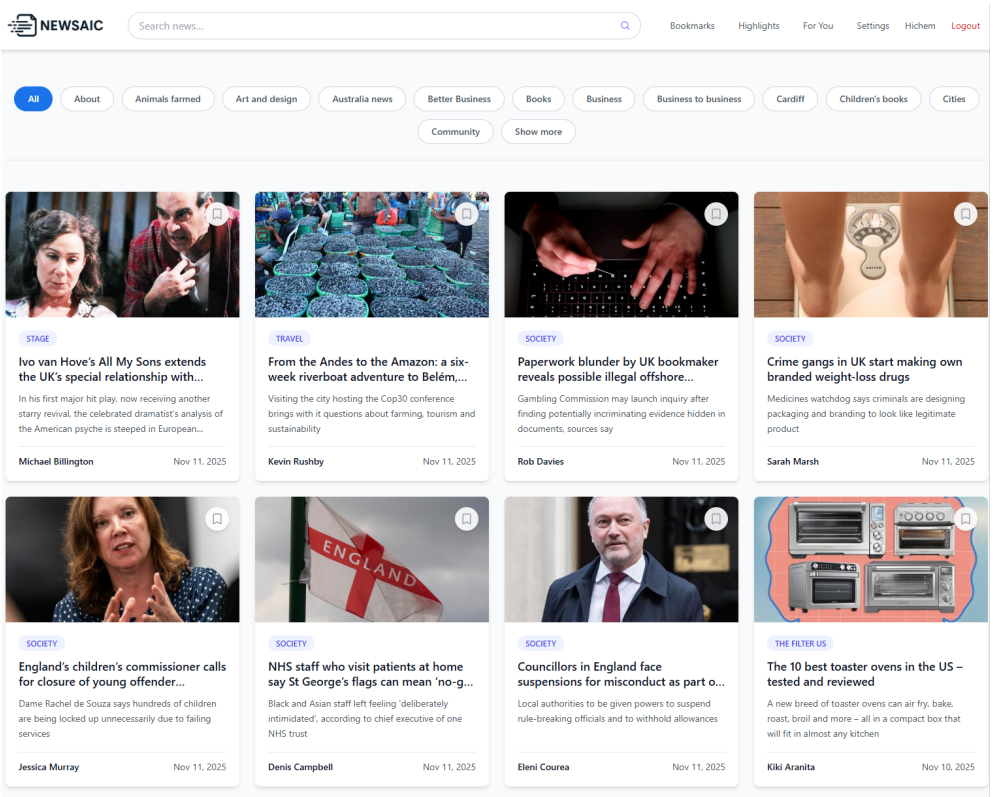

Figure 2.1: For You feed showing personalized articles from user's preferred sections

2.2 Highlights / Top Stories

This feature provides a separate Daily Highlights briefing: a curated set of story briefings written from related articles and tailored to the user's preferences. The briefing is distinct from the main For You feed and is intended as a concise way to catch up on the most important items.

The highlights generation process uses a sophisticated two-stage LLM pipeline. First, an LLM analyzes 30-50 candidate articles (prioritizing the user's preferred sections) and identifies up to 10 story ideas, grouping related articles thematically. Then, for each story idea, a second LLM generates a complete rewritten story that synthesizes information from source articles while adhering to the user's author persona (tone, style, length) and considering their demographics (age, gender) and section preferences. The system ensures all source articles are represented

in multi-article stories and maintains a persona snapshot at generation time for consistency. Each story can optionally include audio narration generated using Kokoro TTS.

2.3 Semantic Search

Users can search for relevant content beyond their personalized feed using semantic search, which leverages embeddings to match articles based on meaning rather than just keywords. Semantic search is available from the feed and the search UI to find any article across the dataset.

The semantic search implementation uses chunk-level embeddings rather than article-level embeddings for finer granularity. When a user enters a query, it is embedded using the same Qwen-3 Embedding model, then compared against all article chunks using cosine similarity. The system retrieves the top 20 matching chunks, deduplicates by article (keeping only the highest-scoring chunk per article), and returns ranked articles. This approach ensures that even if only a portion of an article is relevant to the query, the article can still be discovered.

2.4 Related Articles

Under each article, the top three most similar stories are displayed, helping users explore topics in greater depth and discover additional perspectives. Related-article recommendations appear both in the briefing and in article detail views in the For You feed.

The related articles feature uses article-level embeddings (1024-dimensional vectors) to find semantically similar content. For each article, the system performs a vector similarity search excluding the article itself, retrieves the top 4 candidates, and displays the top 3. This enables users to discover content on related topics even when keywords don't overlap.

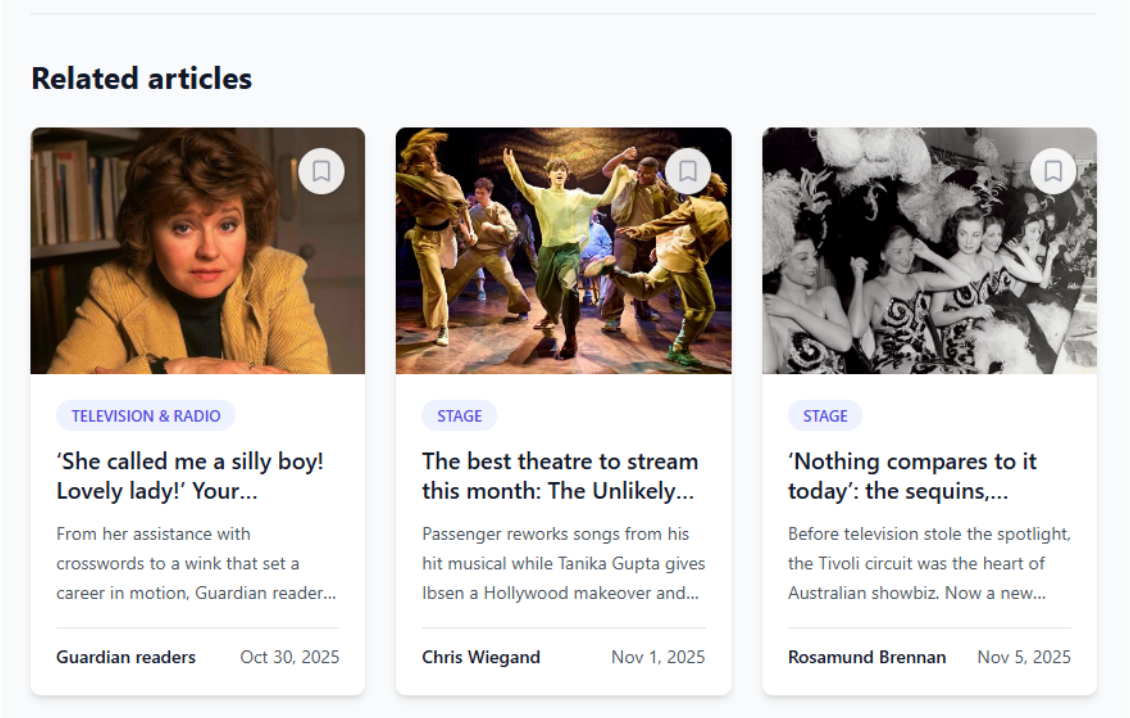

Figure 2.2: Related articles recommendations displayed below an article

2.5 Q&A Module

The Q&A module enables users to ask questions about individual articles. Using a retrieval-augmented generation approach, the system provides answers drawn from the text of the source articles; this is available when a user opens an article from the feed or the briefing.

The Q&A system uses a three-stage pipeline: (1) query refinement, where the user's question is optimized for semantic search using article context; (2) vector search across article chunks to find the most relevant passages; and (3) answer generation that synthesizes information from retrieved chunks. The system returns not only the answer but also the source chunks used, providing transparency and allowing users to verify information.

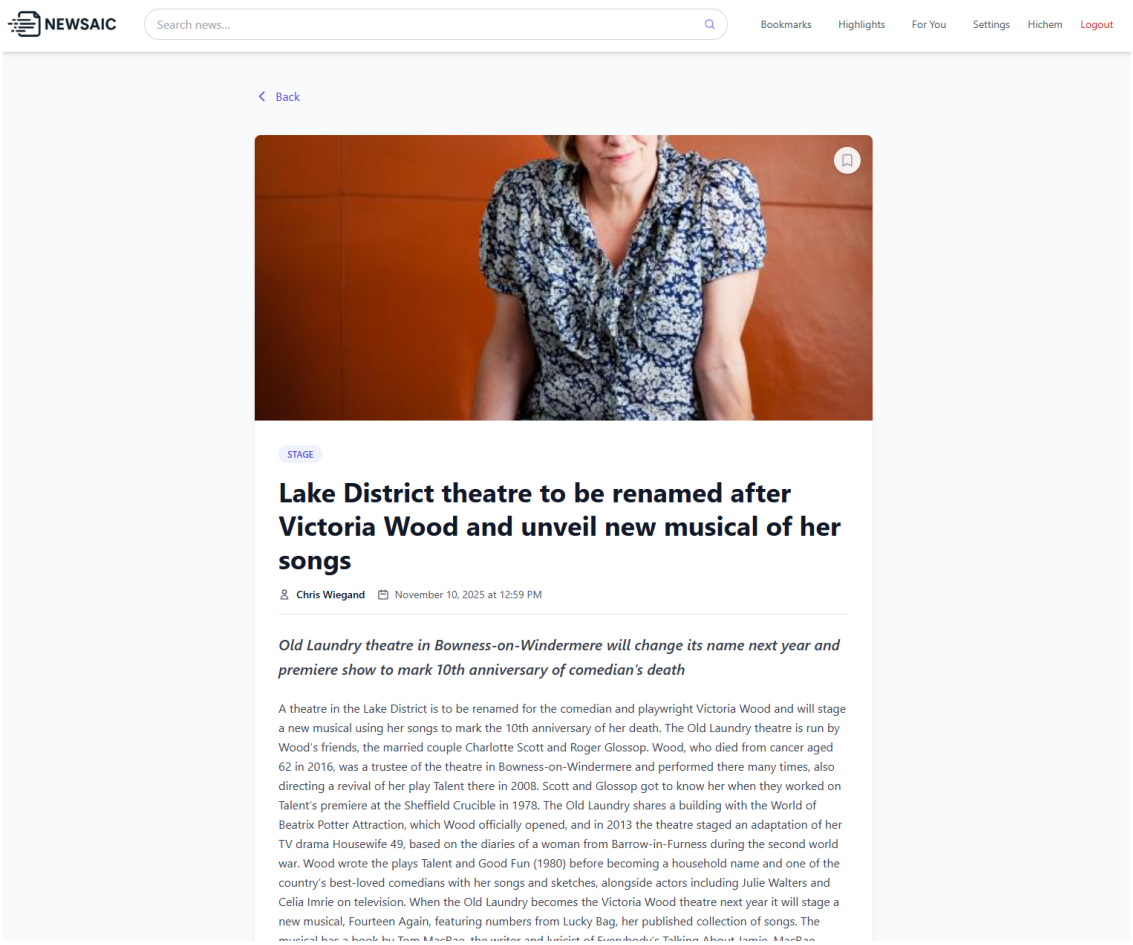

Figure 2.3: Article detail page with Q&A module showing question input and answer with source chunks

2.6 Audio Narration

Articles can be converted into high-quality spoken form for users who prefer listening or have visual impairments, providing an accessibility feature that keeps the application inclusive.

Audio narration is generated automatically during Daily Highlights creation using Kokoro-82M TTS. Each story's body text is processed through Kokoro's KPipeline with a configurable voice (default: 'af_heart'), producing 24kHz WAV audio files that are stored as Django FileField attachments. The narration generation can be skipped for faster development cycles, but is enabled by default

for production use. Users can play narrations directly in the interface through an integrated audio player component.

2.7 Bookmarking

Users can save articles for later reference; bookmarks are accessible from the For You feed and a saved-items view.

2.8 User Management and Personalization

The application includes comprehensive user management features that enable deep personalization:

- **User Profiles**: Users can manage personal information including first name, last name, gender, and birthday, which are used to tailor content generation.
- **Author Persona Configuration**: Each user can customize their content preferences through an author persona system, specifying:
  - **Tone**: friendly, professional, casual, or formal
  - **Style**: concise, detailed, or balanced
  - **Length**: short, medium, or long
  - **Extra Instructions**: free-form text for additional customization
- **Section Preferences**: Users select their preferred news sections (at least one required), which determines both the For You feed content and the articles considered for Daily Highlights generation.
- **User Types**: The system distinguishes between News Readers and Admins, with different permissions and access levels.

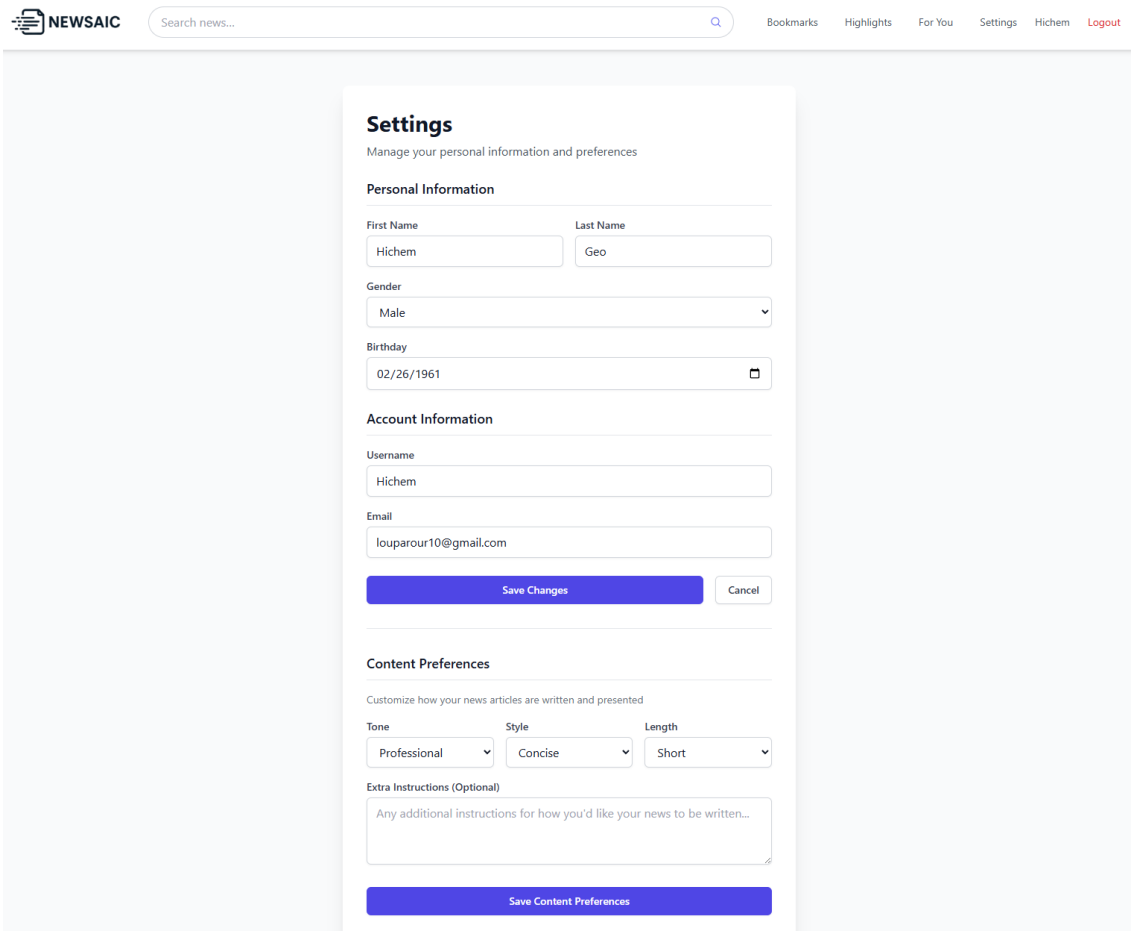

Figure 2.4: Settings page showing user profile, persona configuration, and section preferences

3 Technology Stack

Our technology choices were guided by the need to integrate LLMs, embeddings, and news content into a unified, efficient application.

3.1 Backend Framework and Database

We built the backend in **Python**, primarily due to its strong ecosystem for machine learning and LLM libraries. For the framework, we considered **FastAPI** and **Django**, and for the database, we evaluated **MariaDB**, **PostgreSQL with pgVector**, **sqlite-vss (SQLite Vector Similarity Search)**, and **MongoDB**. In the end, we chose **Django paired with MongoDB**, as it provided a strong combination with easy setup, native support for vector storage operations, and a single system for managing both content and embeddings.

The Django MongoDB backend provides several key advantages:

- **Embedded Models**: Support for nested document structures, allowing us to store tags, contributors, section preferences, and author personas as embedded documents within user and article records.
- **Vector Search Indexes**: Native support for vector similarity search with cosine similarity, enabling efficient semantic search across article embeddings and chunk embeddings.
- **Hybrid Data Model**: Combination of relational models (for articles, users, highlights) with embedded models (for tags, preferences) provides flexibility while maintaining data integrity.

3.2 Task Queue System

We implemented **Celery** for asynchronous task processing, enabling background execution of resource-intensive operations:

- **Article Fetching**: Periodic fetching of articles from The Guardian API runs asynchronously, with configurable batch sizes and date filtering to avoid duplicate fetches.
- **Embedding Generation**: Both article-level and chunk-level embeddings are generated in background tasks, processed in batches to optimize performance and manage API rate limits.
- **Task Scheduling**: Celery Beat is configured for scheduled tasks, allowing automated article fetching and embedding generation on a regular schedule.

3.3 Data Models and Schema

The application uses a carefully designed data model:

- **Article Model**: Stores article metadata, full text, embeddings (1024-dimensional vectors), and embedded tags and contributors. Includes indexes on section IDs, publication dates, and vector search indexes for semantic operations.
- **Chunk Model**: Articles are split into chunks (512 tokens with 50-token overlap) for RAG operations. Each chunk has its own embedding and vector search index, enabling fine-grained semantic search for Q&A.
- **User Model**: Extends Django's AbstractUser with embedded persona and section preferences, user type (Reader/Admin), and demographic information used for personalization.
- **DailyHighlight Model**: Stores a snapshot of the user's persona at generation time, ensuring consistency even if user preferences change.
- **Story Model**: Generated stories with title, body text, order, narration file, and many-to-many relationships to source articles.
- **Bookmark Model**: Simple many-to-one relationships between users and articles with unique constraints to prevent duplicates.

3.4 Frontend

The frontend was implemented in **React**, providing a responsive and modern interface, though it is not central to the report.

3.5 News Sources

Articles were fetched from **The Guardian Open Platform**, chosen for its generous quota, well-structured plain-text articles, and ability to filter content by section and tags, which facilitated user-specific customization.

3.6 LLM Integration

To interface with LLMs, we used **LangChain**, providing a reliable framework for model interaction and text processing for embeddings. For model inference, we relied on **Groq**, which offered generous quotas and access to multiple LLM providers, including **OpenAI**, **Moonshot AI**, and **Meta**, allowing us to compare and select the model best suited for generating persona-based stories. The LLM we ultimately used was **Kimi K2 Instruct 0905**, chosen for its ability to maintain persona-specific style and tone.

3.7 Embeddings

Embeddings were generated using **Ollama**, a local inference engine that allowed us to run embeddings without API limitations. We selected **Qwen-3 Embedding 0.6B** from the MTEB leaderboard for its efficiency on our hardware while providing reliable embedding quality.

Rank (Bor…	Model	Zero-shot	Memory U…	Number of P…	Embedding D…	Max Tokens
1	gemini-embedding-001	99%	Unknown	Unknown	3072	2048
2	Qwen3-Embedding-8B	99%	28866	7B	4096	32768
3	Qwen3-Embedding-4B	99%	15341	4B	2560	32768
4	Qwen3-Embedding-0.6B	99%	2272	595M	1024	32768

Figure 3.1: Qwen-3 Embedding 0.6B performance reference (MTEB)

3.8 Text-to-Speech

For audio narration, we used **Kokoro-82M**, a compact TTS model capable of running locally, producing clear and natural-sounding speech without relying on external services or restrictive quotas. The system uses Kokoro's KPipeline with configurable voice options (default: 'af_heart'), generating WAV audio files at 24kHz sample rate that are stored as Django FileField attachments to story objects.

4 System Architecture

4.1 Story Generation Pipeline

The Daily Highlights feature uses a sophisticated two-stage LLM pipeline to generate personalized stories:

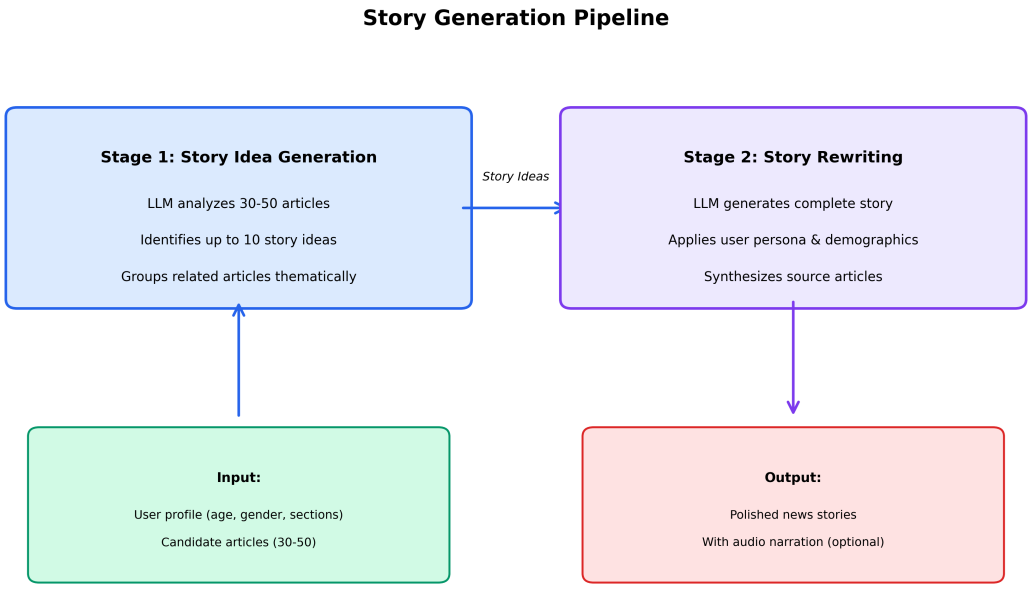

Figure 4.1: Story Generation Pipeline Architecture

4.1.1 Stage 1: Story Idea Generation

The system first analyzes a curated set of articles (30-50 articles from the user's preferred sections, supplemented with other recent articles if needed) and uses an LLM to identify and group related articles into story ideas:

- **Input**: User profile (age, gender, preferred sections), candidate articles with metadata
- **Process**: LLM analyzes articles and suggests up to 10 story ideas, each with:
  - A suggested title
  - A concept description (max 500 characters)
  - Source article IDs that inspired the idea
- **Output**: Structured list of story ideas, filtered for quality and relevance

The LLM is instructed to prioritize quality over quantity, avoid repetitive topics, and only group articles when there's a clear thematic link.

4.1.2 Stage 2: Story Rewriting

For each story idea, the system generates a complete rewritten story:

- **Personalization Context**: The rewriting process considers:
  - User's author persona (tone, style, length, extra instructions)
  - User demographics (age, gender)
  - Preferred sections
  - Source articles' content, titles, sections, and keywords
- **Multi-Article Synthesis**: When multiple source articles are provided, the LLM is instructed to:
  - Ensure all source articles are represented
  - Highlight the most relevant information from each
  - Create a coherent narrative that synthesizes the content
- **Output**: A polished news article with a title and body text, written in the user's preferred style

The entire process is wrapped in a database transaction to ensure atomicity: if story generation fails for all stories, the DailyHighlight is not created.

4.1.3 Narration Generation

After story creation, audio narrations are generated using Kokoro TTS:

- Stories are processed sequentially
- Each story's body text is converted to speech using the configured voice
- Audio is saved as WAV files and attached to story objects
- The process can be skipped with a flag for faster generation during development

4.2 Q&A Pipeline Architecture

The Q&A module implements a three-stage retrieval-augmented generation (RAG) pipeline:

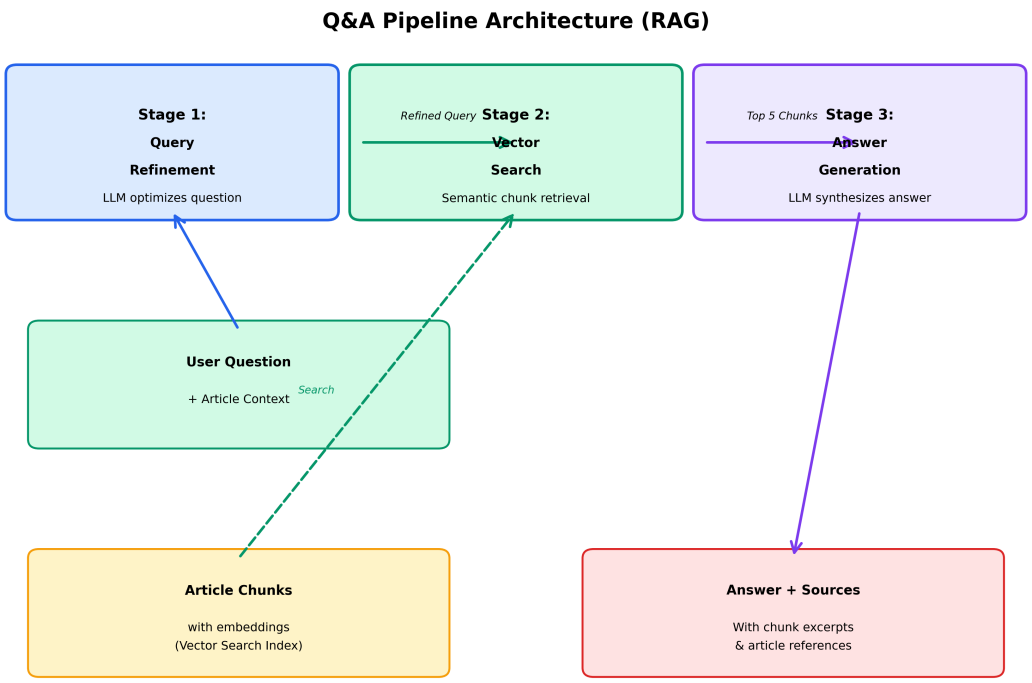

Figure 4.2: Q&A Pipeline Architecture

4.2.1 Stage 1: Query Refinement

Before performing vector search, the user's question is refined by an LLM:

- **Context**: The system provides the article title, section, and a preview of the article body
- **Purpose**: Transform the user's natural language question into an optimized query for semantic search
- **Output**: A concise, context-aware query string

4.2.2 Stage 2: Vector Search

The refined query is embedded and used for semantic search:

- **Search Target**: Article chunks (not full articles), enabling fine-grained retrieval
- **Method**: Cosine similarity search using MongoDB's VectorSearchIndex
- **Parameters**: Top 5 most similar chunks are retrieved, with 100 candidates considered
- **Result**: Ranked list of relevant chunks with their source article information

4.2.3 Stage 3: Answer Generation

The retrieved chunks are used to generate an answer:

- **Input**: Original question, article context, and retrieved chunks
- **Process**: LLM generates a factual answer using only the provided chunks and article content
- **Safety**: The LLM is instructed to say "I don't have enough information" if the answer cannot be found in the provided content
- **Output**: Structured response including:
  - The answer text
  - List of used chunks with excerpts and source article information
  - The refined query that was used

This pipeline ensures answers are grounded in the source material while being responsive to user questions.

4.3 Article Processing Pipeline

Articles undergo several processing stages:

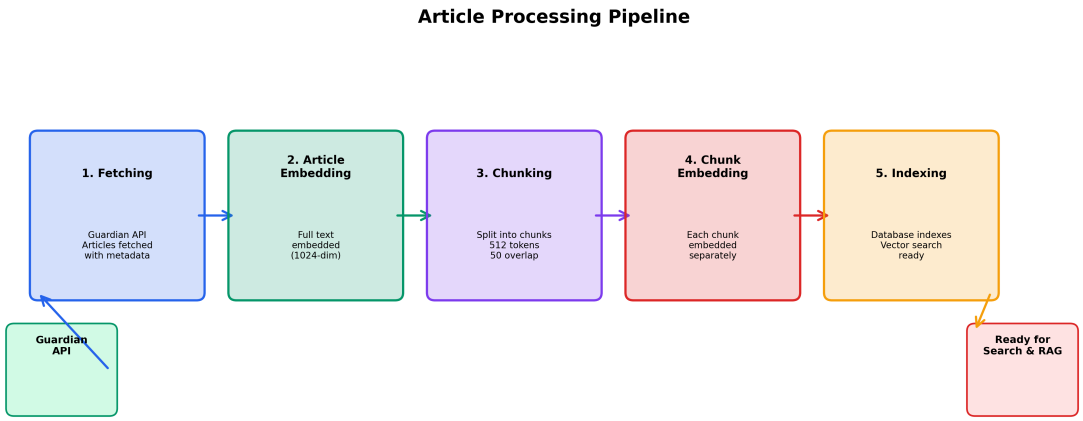

Figure 4.3: Article Processing Pipeline

1. **Fetching**: Articles are fetched from The Guardian API with configurable date ranges and batch sizes. The system tracks already-fetched articles to avoid duplicates.

2. **Article Embedding**: Full article text is embedded using Qwen-3 Embedding 0.6B, generating 1024-dimensional vectors stored in the article model. These embeddings are used for related article recommendations.

3. **Chunking**: Articles are split into chunks using RecursiveCharacterTextSplitter:

   - Chunk size: 512 tokens
   - Overlap: 50 tokens
   - Preserves article metadata for traceability

4. **Chunk Embedding**: Each chunk is embedded separately, creating fine-grained vectors for RAG operations. Chunks have their own vector search index.

5. **Indexing**: Database indexes are created on:
    - Section IDs (for filtering)
    - Publication dates (for time-based queries)
    - Vector search indexes (for semantic operations)

4.4 Management Commands

The system includes several Django management commands for data processing:

- `fetch_sections`: Populates the database with available news sections from The Guardian API
- `fetch_articles`: Fetches articles from The Guardian API with options for:
    - Date range filtering
    - Batch size control
    - Maximum pages to fetch
    - Automatic duplicate detection
- `embed_articles`: Generates embeddings for articles that don't have them yet, processing in configurable batch sizes
- `embed_article_chunks`: Splits articles into chunks and generates embeddings for each chunk, with options for:
    - Batch size control
    - Maximum articles to process
    - Re-embedding existing chunks
- `generate_highlights`: Complete pipeline for generating daily highlights:
    - Can target specific users or all readers
    - Configurable voice for narration
    - Option to skip narration
    - Repeat mode for scheduled generation (every N minutes)

5 Results

Implemented features include a personalized For You feed (section-based), a separate Daily Highlights briefing of story-based summaries, semantic search, related-article recommendations, per-article Q&A, audio narration, and bookmarking.

5.1 Technical Implementation Summary

The system successfully integrates multiple components:

- **Backend**: Django REST Framework API with MongoDB backend, supporting embedded models for flexible data structures and native vector search capabilities.
- **Asynchronous Processing**: Celery task queue handles article fetching and embedding generation in the background, enabling scalable processing of large article volumes.
- **Vector Search**: Dual-level embedding system (article-level for recommendations, chunk-level for Q&A) provides both coarse and fine-grained semantic search capabilities.
- **LLM Integration**: Structured output using Pydantic models ensures type safety and reliable parsing of LLM responses across the story generation and Q&A pipelines.
- **Caching**: Response caching for highlights (2-hour TTL with custom cache keys) reduces redundant processing and improves response times.
- **Data Processing**: Comprehensive management commands enable efficient data pipeline operations, from initial setup through ongoing maintenance.

The architecture supports personalization at multiple levels: user demographics influence story generation, section preferences filter content, and author personas shape writing style, creating a truly customized news experience.

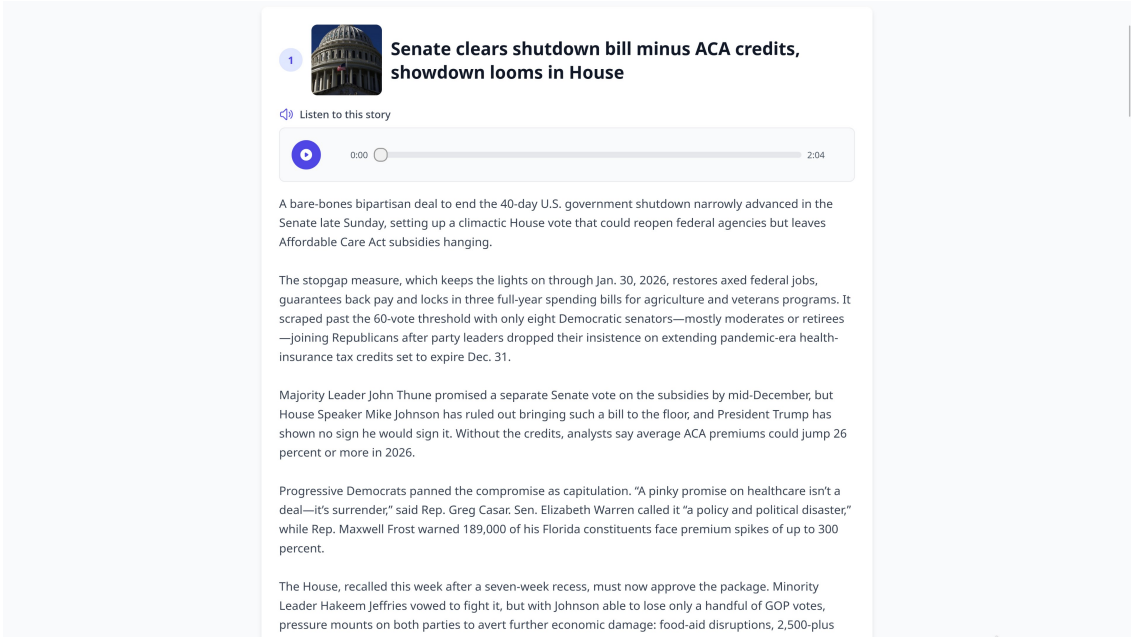

Figure 5.1: Daily Highlights view with audio narration toggle

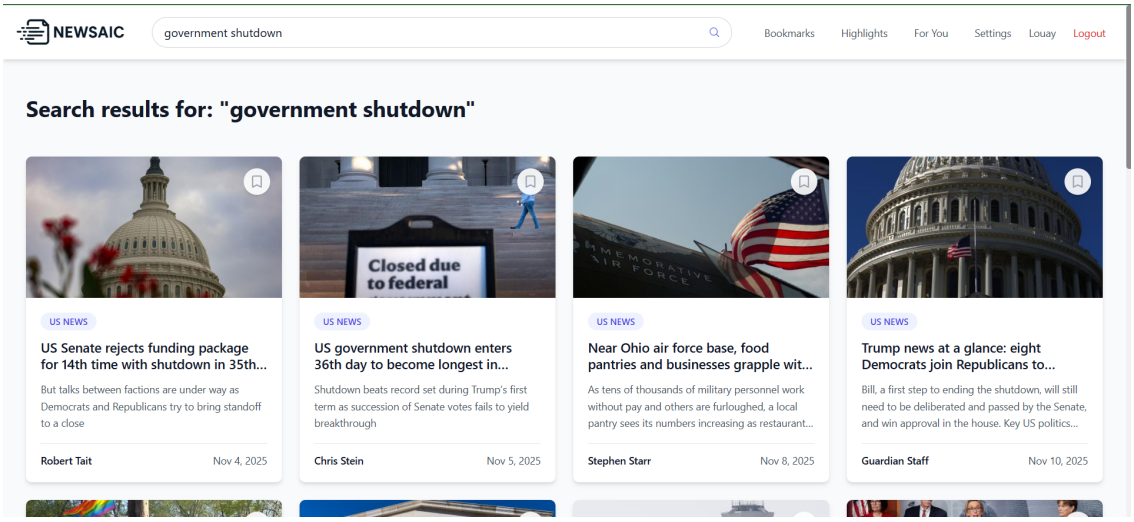

Figure 5.2: Semantic Search interface screenshot

Conclusion

Newsaic demonstrates how news engagement can be enhanced through building stories from related articles, story generation, semantic search, Q&A, audio narration, and bookmarking. Thanks to advances in LLMs, embeddings, and local inference engines, the application can generate stories, surface related content, answer questions about articles, and provide audio narration. Looking ahead, the system could be extended to support persona-based TTS, allow further customization such as the number of top stories, integrate additional functionalities, and provide a more intelligent, personalized news feed.