Lebanese University

Faculty of Economics and Business administration

Branch 3

News AI Advance Project

M2 Graduation Project

Major: Business Computer

Prepared by:

Nacir Chahine

Committee Members

Supervisor: Dr Dalia Battikh

Academic year

2024-2025

Table of Contents

Table of Figure

Abstract

This memoir documents the development of News Advance, an AI-powered news credibility analyzer built with Django 5.2. The platform aggregates news articles from diverse, reliable sources, then applies an AI-enhanced analysis pipeline to assess credibility, detect bias, evaluate sentiment, and summarize content.

At the core of News Advance is a custom fine-tuned BART summarization model, trained on the BBC News Summary dataset for fast, domain-specific summaries, with seamless fallback to Ollama-powered local LLMs for more nuanced or creative analysis. The system also integrates political bias detection, sentiment scoring, and misinformation alerts, presented through a user-friendly web interface.

Beyond technical innovation, News Advance represents a personal journey into responsible AI, balancing the precision of machine learning with the flexibility of local LLM integration to empower readers in navigating today's complex media ecosystem.

Acknowledgments

I extend my deepest gratitude to everyone who supported me throughout the creation of News Advance.

To my mentors and peers who offered guidance, encouragement, and constructive criticism at every stage — your insights were invaluable in transforming an ambitious idea into a working solution.

To the open-source community — whose contributions to Django, newspaper3k, spaCy, transformers, and countless other libraries made this project possible — I owe a debt of thanks. Your collective innovation set the foundation for my own work.

Finally, to my friends and family — thank you for your patience, especially during the late nights spent debugging, retraining models, and poring over API responses. Your belief in me kept the momentum alive even when the challenges felt overwhelming.

Dedication

To everyone who believes that technology can be a force for truth.

This work is dedicated to the journalists who strive for accuracy, the developers who build tools for transparency, and the readers who seek to understand the world with an open and critical mind.

And to those who reminded me that persistence is more than just debugging — it's the courage to keep building even when the vision feels far away.

We also dedicate this project to all the open-source developers who have contributed to the development of the tools and libraries we used to create this project. We appreciate their efforts in making software development accessible and collaborative, and we hope to contribute back to the community in our own small way.

Finally, we would like to dedicate this project to our loved ones who have supported us throughout the development process. Their encouragement and patience have helped us to overcome challenges and push forward to completion.

General Introduction

The digital age has transformed how we consume news, but it has also blurred the lines between fact, opinion, and misinformation. With the speed at which content spreads, traditional fact-checking methods struggle to keep pace. The result is an information ecosystem where trust can be fragile and critical thinking more important than ever.

News Advance was conceived as a response to this challenge — a system that not only collects and organizes news from multiple sources but also applies AI-driven analysis to evaluate its credibility. Built on Django 5.2, the platform brings together automated news aggregation, political bias detection, sentiment analysis, summarization, and source reliability scoring in one unified environment.

The project's architecture reflects a deliberate focus on modularity and scalability, with separate apps for aggregation, analysis, and user management. Under the hood, it integrates

both traditional NLP methods (such as VADER sentiment scoring) and modern transformer-based models (including a fine-tuned BART summarizer) to ensure robust performance.

This memoir explores the journey from concept to implementation, detailing the technical, ethical, and practical considerations involved in creating a system that aims to help users navigate today's complex media landscape.

Chapter 0: The Spark of an Idea - Dealing With the Information Problem

Today's digital world has a strange problem. We have more information than ever, but finding the truth is getting harder. We see a constant flood of news, opinions, and stories, and it's tough to tell the difference between real journalism and fake news. This problem gets worse on social media, where algorithms show us what we already agree with, creating "filter bubbles" that keep us from seeing different points of view. The "News Advance" project started as a way to fix this.

Our mission was clear, but also very ambitious: to help the modern news reader. We know that being able to understand the media is a key skill today, so we wanted to build a tool that was more than just a news feed. We imagined a smart helper that would give people the confidence to look past the headlines and understand the complex world of online news. The main goal was to build a web app that uses Artificial Intelligence to help people check the articles they read, understand media bias, and spot potential fake news. By making things more transparent, we hoped to help create smarter readers who could have more meaningful conversations about important topics.

At its core, News Advance was designed to be a powerful analysis tool. The plan was for the system to collect articles from all kinds of news sites—from big, old newspapers to new online blogs—and run them through a set of tough AI checks. This included checking for political bias by looking for loaded words, and analyzing the emotional tone to see if an article was neutral or trying to make you feel a certain way. It would also create short, accurate summaries so people could get the main points quickly. Finally, we wanted to build a foundation for a fact-checking system to check claims against trusted sources. We didn't want to just build

another news app; we wanted to build a new lens for people to see the news through, one that shows the structure, purpose, and quality of the content.

This memoir is the story of how we turned that idea into a real thing—a story about the tech we used, the code we wrote, the unexpected problems we faced, and the smart wins that made it all happen.

Chapter1: Business Requirement

1.1-Introduction

News Advance was designed to meet a clear and urgent need: empowering users to assess the credibility of the news they consume. To achieve this, the system incorporates the following core functions and features:

- News aggregation: Fetch and parse articles from vetted sources via newspaper3k; normalize metadata and extract main images.
- Political bias analysis: LLM-driven leaning classification (left, center-left, center, center-right, right) with numeric bias_score and confidence.
- Sentiment analysis: VADER baseline with AI-enhanced sentiment (classification, score, explanation) when Ollama is available.
- Summarization: Primary ML path using a fine-tuned BART model (BBC News Summary dataset); automatic fallback to local LLMs via Ollama when the ML model isn't available.
- Key insights extraction: AI-generated bullet points stored per article and shown in a collapsible panel.
- Logical fallacies: Curated catalog with per-article detections and deep links to fallacy detail pages.
- Fact-checking pipeline: Claim extraction + LLM verification → rating, confidence, explanation, sources.
- Source reliability scoring: Weighted aggregate of fact-check outcomes (~60%), bias consistency (~20%), and fallacy frequency (~20%).
- User system: Authentication, saved articles, and preferences to toggle summary, key insights, and fact-checks visibility.
- Misinformation alerts: Admin-managed alerts linked to articles, email notifications.

1.2-Tools and libraries requirement

The system integrates a wide range of tools and libraries to support its AI-driven analysis pipeline and web framework infrastructure:

- Core: Django 5.2; SQLite (dev) / PostgreSQL (prod); Bootstrap 5.
- Aggregation: newspaper3k, requests, BeautifulSoup5.
- NLP: NLTK (vader_lexicon, punkt, stopwords), spaCy (en_core_web_sm).
- AI/LLM (local via Ollama): llama3, deepseek-r1:8b, mistral, qwen2:1.5b, phi.
- ML summarization (optional): transformers + torch for fine-tuned BART.
- Utilities: Faker (test data), python-dotenv, pillow, lxml.

1.3-Overview of News Advance

At its core, News Advance operates as a three-layered system:

1. Aggregation layer: newspaper3k-driven ingestion and normalization from diverse sources.
2. Analysis layer: sentiment, bias, ML summarization with Ollama fallback, key insights, fallacy detection, fact-checking, and source reliability scoring.
3. User layer: templates and views for browsing sources and articles, toggling AI features, and viewing analysis results and alerts.

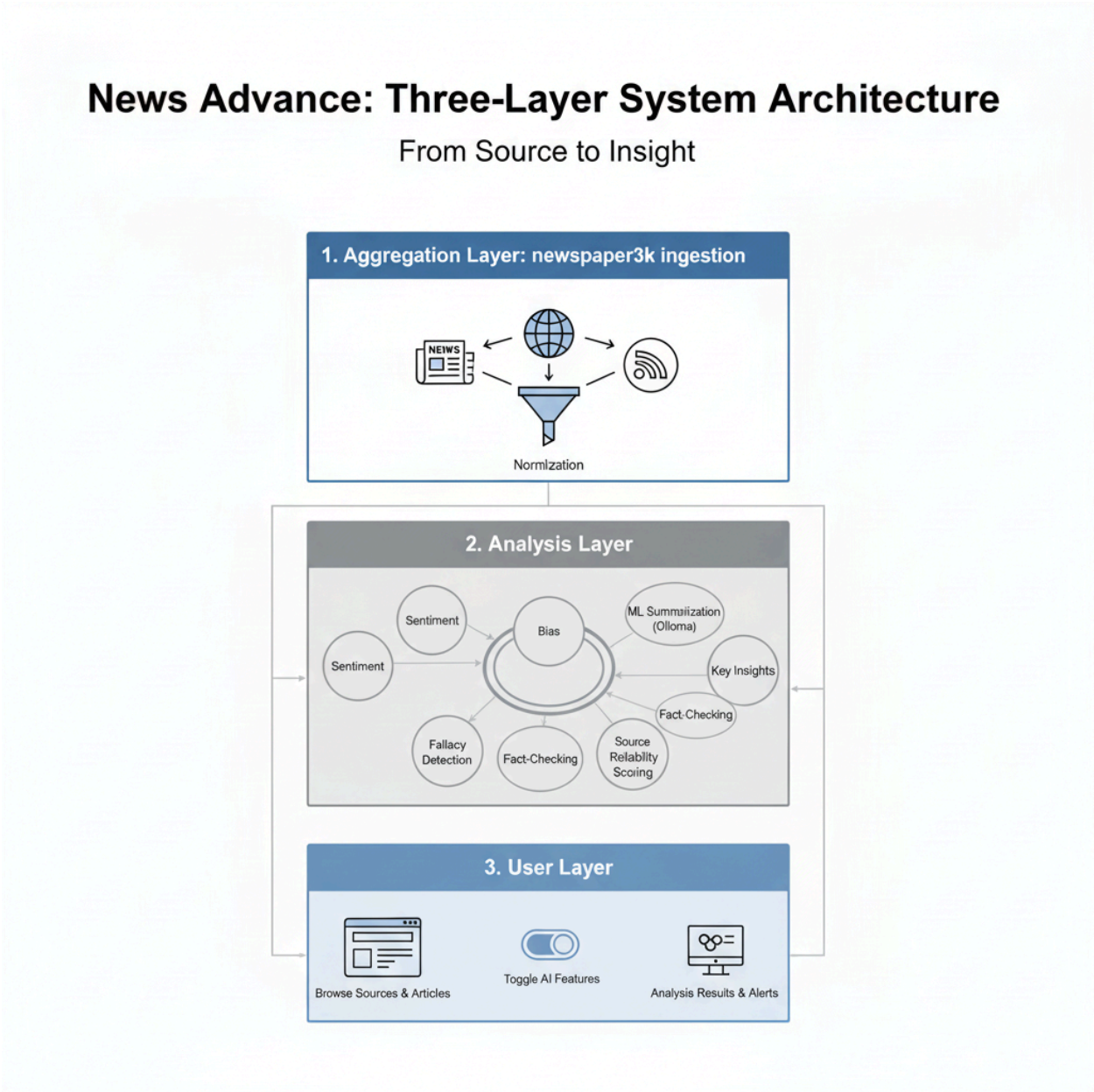


Figure 1: System Layers

Dual summarization is supported: a news-optimized BART model for speed/consistency and an Ollama LLM path for flexible fallback. The architecture is modular across apps (news_aggregator, news_analysis, accounts) for maintainability and future expansion.

1.4-Overview Django

Django is a high-level Python web framework that promotes rapid development and clean, pragmatic design. For News Advance, Django provides:

News Advance applies Django's MVT across three apps with project-level settings for Ollama and ML summarization.

ORM models: articles, sources, analyses (bias, sentiment, summary), fact checks, logical fallacies, alerts.

Auth + admin: curation of fallacy catalog, managing alerts and fact checks, and quick source/article administration.

Templates/static: Bootstrap-based UI with accessible components and toggles for AI features.

Django provides the maintainable backbone that lets AI-driven analysis coexist with a clean, user-friendly web experience.

1.5-Conclusion

In this chapter, we outlined the essential business requirements that shaped the vision of News Advance. Starting with the core functions and features, we identified the system's responsibility to aggregate news from diverse sources, analyze sentiment and bias, generate concise summaries, detect logical fallacies, and provide fact-checking mechanisms. These features, combined with user preferences and alerting capabilities, ensure that the platform addresses both the technical challenge of large-scale information processing and the human need for transparency and trust.

The tools and libraries presented — from Django and PostgreSQL to transformers, spaCy, and Ollama — demonstrate a deliberate alignment between reliable backend infrastructure and advanced natural language processing capabilities. Similarly, the system overview and Django's MVT pattern highlighted how the project's architecture supports modularity, scalability, and maintainability.

By defining these requirements clearly, this chapter set the groundwork for the following design and modeling phase. The functional vision, supported by the technical toolkit, forms the blueprint upon which the system's conceptual models and eventual implementation will be built. In short, Chapter 1 provided the "what" and the "why," paving the way for Chapter 2 to focus on the "how."

Chapter 2: Development Tools

2.1-Introduction

The successful implementation of News Advance depended not only on a clear vision of its functions but also on the careful selection of development tools. Each tool was chosen for a specific purpose, balancing reliability, performance, and ease of integration. From the development environment to the programming language and AI frameworks, the tools provided the foundation for building a system that is both technically sound and user-oriented.

In this chapter, we present the main tools and technologies that supported the development of the project. PyCharm was selected as the primary Integrated Development Environment (IDE), offering robust features for productivity and debugging. Python, as the core programming language, provided an extensive ecosystem of libraries crucial for natural language processing, machine learning, and web development. Ollama enabled the integration of local large language models (LLMs), such as llama3, ensuring flexibility in analysis tasks while reducing reliance on cloud-based APIs. Finally, a custom summarization model, fine-tuned on BBC news data, was developed to generate fast, domain-specific summaries tailored to the needs of modern news readers.

Together, these tools formed the backbone of the project, transforming the abstract business requirements defined in Chapter 1 into a practical and scalable implementation.

2.2-PyCharm

PyCharm, developed by JetBrains, was chosen as the primary IDE for building News Advance. It provided a professional environment that streamlined the entire development workflow. Features such as intelligent code completion, real-time error detection, integrated version control, and built-in Django support significantly enhanced productivity.

The debugging tools in PyCharm proved invaluable during the integration of complex modules like the aggregation pipeline and the analysis system. Instead of relying solely on print statements, the

debugger allowed for step-by-step inspection of code execution, variable states, and error traces. This was particularly useful when troubleshooting Django views, API endpoints, and the AI model integration.

Moreover, PyCharm's project management features made it easy to maintain a modular architecture, keeping the news_aggregator, news_analysis, and accounts apps organized. Combined with virtual environment management and plugin support, PyCharm provided a stable and efficient foundation for both rapid prototyping and long-term maintainability.

Figure 2: jetbrains pycharm logo

2.3-Python

Python served as the backbone of the *News Advance* project. Its readability, extensive libraries, and strong community support made it the ideal choice for combining web development with natural language processing. Django, the chosen web framework, leveraged Python's strengths to provide a secure and scalable foundation for the system.

Beyond Django, Python's rich ecosystem of libraries played a crucial role in the project:

- **Natural Language Processing:** NLTK (VADER), spaCy, and transformers were used for tasks ranging from sentiment detection to advanced summarization.

- **Web Scraping:** newspaper3k and BeautifulSoup4 allowed seamless aggregation of news articles from diverse sources.

- **Machine Learning:** PyTorch and Hugging Face Transformers powered the fine-tuned summarization model.

- **Utilities:** Python libraries such as Faker, dotenv, and Pillow simplified testing, configuration, and image processing.

The flexibility of Python ensured that *News Advance* could integrate traditional rule-based NLP with state-of-the-art transformer models, while also supporting robust database operations, APIs, and user interfaces.

2.4-Ollama

Ollama was integrated into the system to enable the use of local large language models (LLMs). Its primary advantage was offering AI-powered text analysis without relying on cloud services, thereby ensuring greater privacy, lower latency, and reduced dependency on external APIs.

Within *News Advance*, Ollama was used as a fallback mechanism and an enhancement layer in the analysis pipeline. Whenever the fine-tuned summarization model was unavailable, or when a more nuanced response was required, Ollama-powered models could step in to handle sentiment analysis, bias detection, and claim verification.

The modular configuration of Ollama allowed multiple models to be tested and integrated efficiently. Among these, *llama3* served as a general-purpose language model, while specialized models were applied for summarization and reasoning tasks.

Figure 3: Ollama logo

2.4.1-Llama3

Llama3 was one of the primary models deployed through Ollama for *News Advance*. It was chosen for its balance between performance and efficiency on local hardware. The model supported key features such as political bias classification, sentiment interpretation, and logical fallacy detection.

One of the advantages of llama3 was its ability to generate context-aware insights quickly, making it well-suited for interactive analysis in the user interface. Although not specifically fine-tuned for journalism, it provided robust general-purpose language understanding that complemented the more specialized summarization model.

2.4.2-Custom Summary Model Trained on BBC Data

To achieve fast and domain-relevant news summarization, a custom model was developed using the facebook/bart-base architecture. This model was fine-tuned on the BBC News Summary dataset, which contains thousands of professionally written news articles and summaries.

The fine-tuned model delivered concise, high-quality summaries tailored to the structure of news writing, ensuring users could grasp the key points of an article without reading the full text. This reduced information overload while maintaining factual accuracy and readability.

In production, the summarization pipeline prioritized this fine-tuned BART model. If unavailable, it fell back to Ollama-powered LLMs, which provided more flexible and creative outputs at the cost of consistency. This dual-path strategy ensured both reliability and adaptability in news analysis.

2.5-Conclusion

The tools described in this chapter were more than just technical choices — they defined the identity and capabilities of News Advance. PyCharm provided the environment for efficient development and debugging, while Python offered a versatile programming foundation supported by a powerful ecosystem of libraries. Ollama extended the system’s flexibility,

enabling the integration of advanced LLMs such as llama3, while the custom summarization model ensured fast, domain-specific outputs tailored for journalism.

Together, these tools transformed the project's business requirements into a concrete implementation strategy. They enabled seamless interaction between web development, natural language processing, and artificial intelligence, creating a platform that is both innovative and practical.

This chapter highlighted the "with what" of the project's development. The next chapter will shift the focus to the "how," exploring the modeling and conceptual design that structured the system's architecture.

Chapter3: Modeling and conception

3.1-Introduction

Modeling and conception represent a critical stage in the development of News Advance. After defining the business requirements and selecting the development tools, the next step was to translate these abstract needs into a coherent system design. The goal of this chapter is to present the structural blueprint of the project, showing how its components interact and how the system's complexity is managed.

Unified Modeling Language (UML) diagrams were chosen to formalize this design. UML offers an expressive, standardized visual language that allows developers to communicate, validate, and refine the architecture before implementation. By constructing use case diagrams, class diagrams, and other design artifacts, we ensured that the system would not only meet functional requirements but also remain modular, extensible, and maintainable.

Through modeling, the project moved from what the system should do toward how the system will achieve it. This transition bridges the gap between theoretical requirements and practical coding, providing a solid foundation for the implementation phase that follows.

3.2-What is Unified Modeling Language (UML)

The UML is a very important part of developing object-oriented software and the software development process. The UML uses mostly graphical notations to express the design of software projects. Using the UML helps project teams communicate, explore potential designs, and validate the architectural design of the software. In this article, we will give you detailed ideas about what UML is, the history of UML and a description of each UML diagram type, along with UML examples.

Why UML?

As the strategic value of software increases for many companies, the industry looks for techniques to automate the production of software and to improve quality and reduce cost and time-to market. These techniques include component technology, visual programming, patterns, and frameworks. Businesses also seek techniques to manage the complexity of systems as they increase in scope and scale.

Goals of UML:

1. Provide users with a ready-to-use, expressive visual modeling language so they can develop and exchange meaningful models.
2. Provide extensibility and specialization mechanisms to extend the core concepts.
3. Be independent of particular programming languages and development processes.
4. Provide a formal basis for understanding the modeling language.
5. Encourage the growth of the OO tools market.
6. Support higher-level development concepts such as collaborations, frameworks, patterns, and components.
7. Integrate best practices

3.3-Use Case Diagram

A use case diagram is used to clarify the interactions between system actors and the functionalities they can access. For News Advance, two primary actors were identified: Visitor/User and Admin.

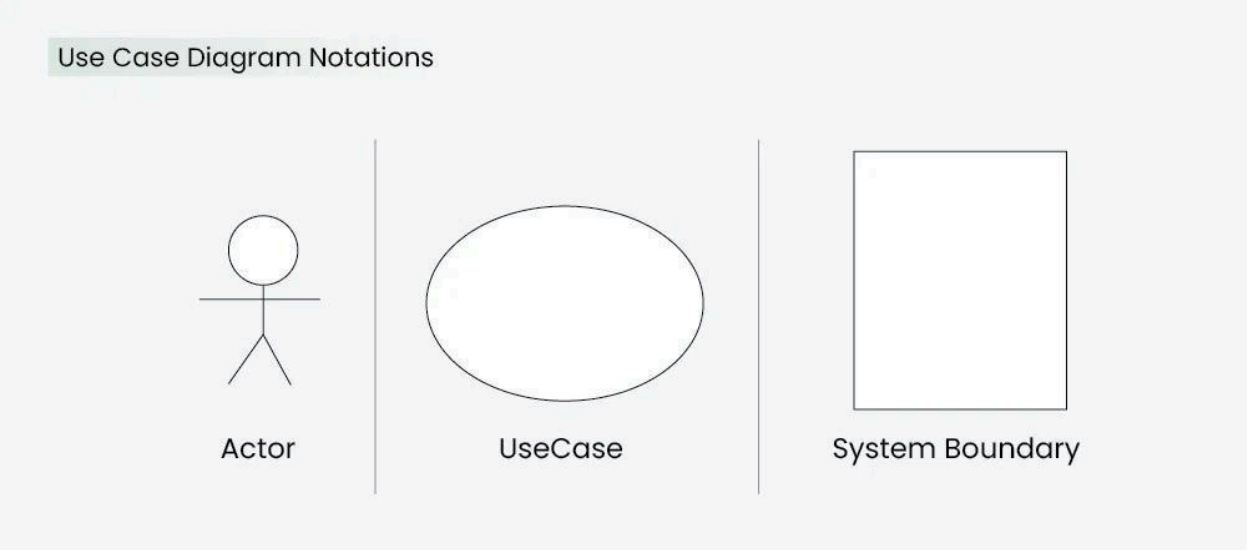
Use Case Diagram Main actors and goals:

- Visitor/User can register or log in, edit their profile, and customize preferences such as showing or hiding summaries, insights, or fact-checks. They can browse news sources, read articles, and save items of interest. When opening an article, the system provides multiple AI-driven analyses:

    - Political Bias – classifying the article's leaning with confidence scores.
    - Emotional Tone (Sentiment Analysis) – detecting neutrality, positivity, or negativity, with optional Ollama-enhanced explanations.

- Logical Fallacies – highlighting flawed arguments with references to the fallacy catalog.
- Fact Checks – extracting claims and verifying them against trusted sources.
- Summarization – providing concise, domain-specific summaries from the custom fine-tuned BART model (or Ollama fallback).
- Key Insights – bullet-point extractions of essential information.

- Admin manages the system's integrity by curating sources and articles, reviewing analyses, and managing logical fallacies. They also have the ability to trigger re-analysis, ensuring that results remain consistent when models or detection rules evolve. CRUD (Create, Update, Delete) operations apply both to sources and logical fallacies.



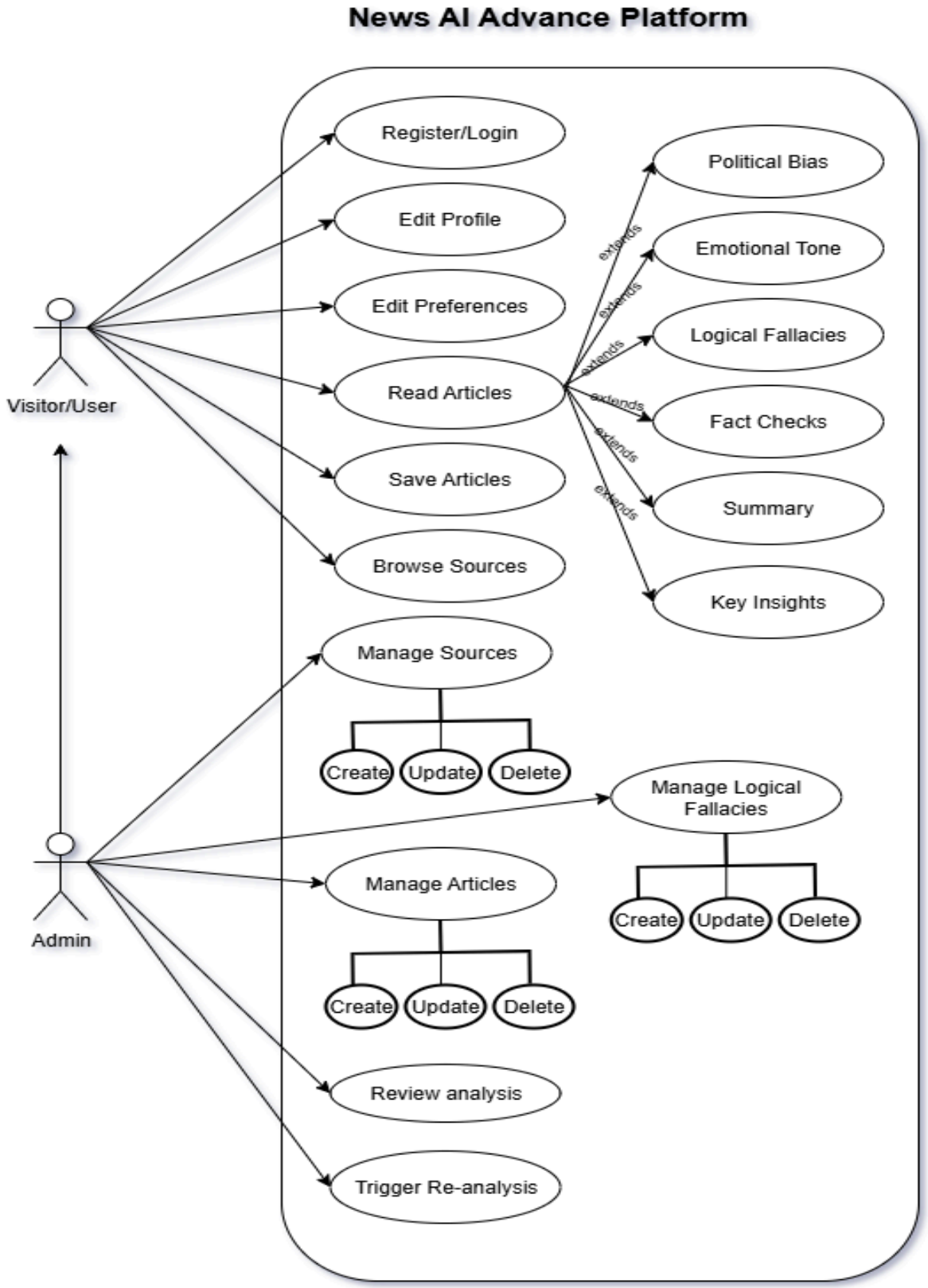


Figure 4: Use Case Diagram

This diagram makes the dual role of the system explicit: on one side, empowering users with transparency tools to navigate news critically; on the other, equipping administrators with the oversight and control necessary to maintain system reliability and trustworthiness.

Narrative

A visitor begins by browsing available sources or searching for specific topics. The system fetches and displays stored articles, which can be selected for detailed review. Upon opening an article, the analysis pipeline is triggered, surfacing multiple layers of AI-driven insights:

- Summary – a concise representation of the article, generated by the custom fine-tuned BART summarizer with fallback to Ollama models.
- Sentiment Analysis – emotional tone detection (positive, negative, or neutral), enhanced with explanatory feedback when available.
- Political Bias Analysis – classification of the article's leaning along the spectrum from left to right, quantified with a numeric bias score and confidence level.
- Key Insights – AI-generated bullet points that highlight the core information contained in the article.
- Logical Fallacies – detections of flawed arguments, linked to a curated catalog for deeper understanding.
- Fact Checks – extracted claims verified against trusted sources, with ratings, confidence scores, and evidence citations.

Users may save articles for later review, adjust visibility preferences (e.g., showing or hiding summaries or fact-checks), and manage their profiles.

On the other side, Administrators play a supervisory role. They manage sources and articles through CRUD (Create, Update, Delete) operations, curate the logical fallacy catalog, and review system-generated analyses. When model updates or new detection methods are introduced, admins can trigger re-analysis to ensure results remain accurate and up to date.

Entity Mapping

The use case diagram also maps directly onto the system's main entities and relationships, as modeled in the class diagram:

- User – represents individuals interacting with the system. Each user has a secure profile and can save multiple articles.
- NewsArticle – the central data model, storing article content, metadata, and analysis results.
- BiasAnalysis – one-to-one with each article, storing political leaning, bias score, and confidence.
- SentimentAnalysis – one-to-one with each article, storing sentiment classification and subjectivity score.
- Summary – linked to each article, providing concise AI-generated summaries.
- ArticleInsight – one-to-many relationship with NewsArticle, representing extracted bullet points.
- FactCheckResult – one-to-many relationship with NewsArticle, storing claim text, verification rating, confidence, and evidence.
- LogicalFallacyDetection – one-to-many relationship with NewsArticle, referencing catalogued fallacies.
- MisinformationAlert – linked to articles when admins identify trending misinformation patterns.

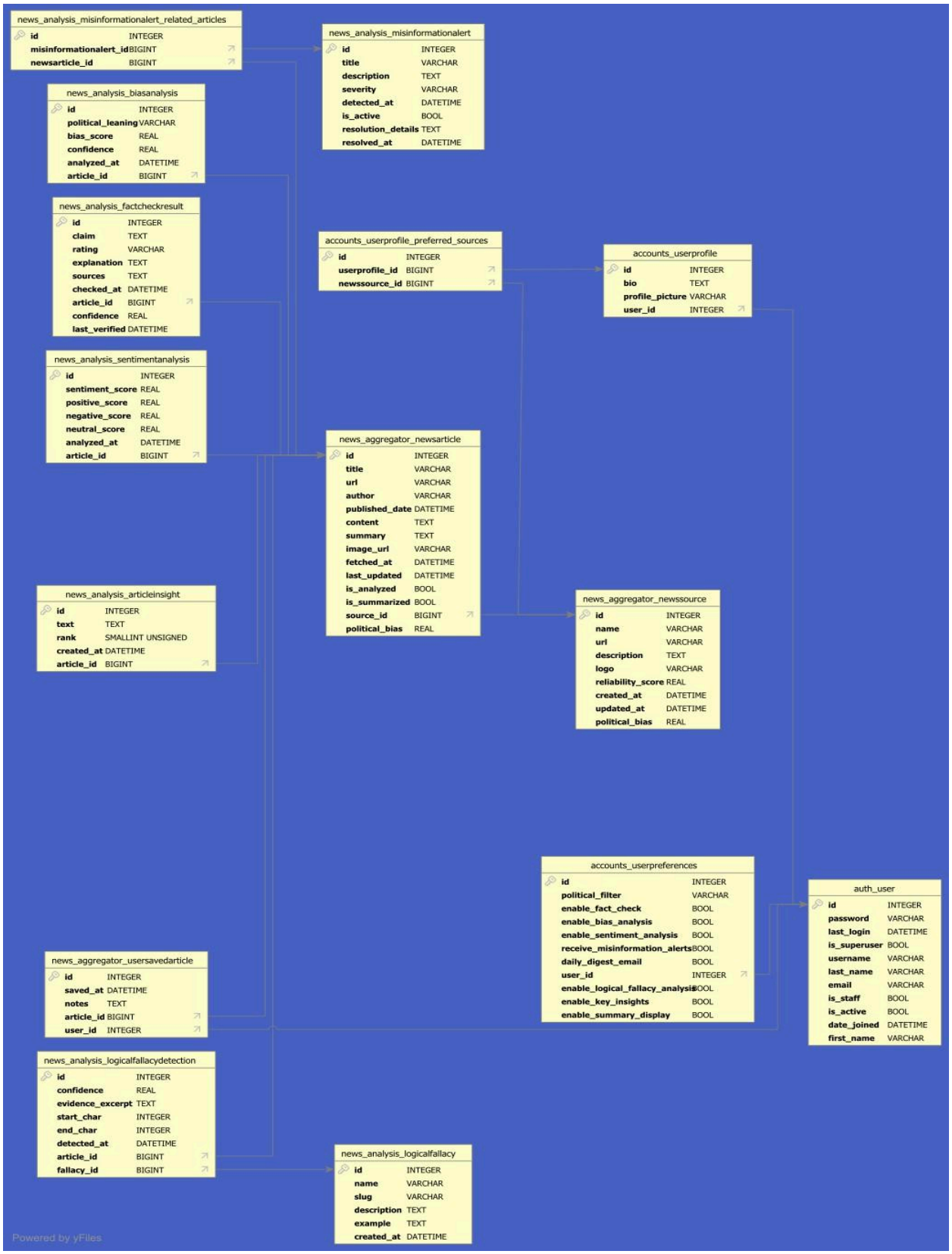


Figure 5: Tables structure and relation

Relationship Summary

- User → NewsArticle: One-to-many (a user can save multiple articles).
- NewsSource → NewsArticle: One-to-many (a source can have multiple articles).
- NewsArticle → BiasAnalysis: One-to-one (each article has one bias analysis).
- NewsArticle → SentimentAnalysis: One-to-one (each article has one sentiment analysis).
- NewsArticle → ArticleInsight: One-to-many (each article may have multiple extracted insights).
- NewsArticle → FactCheckResult: One-to-many (each article may have multiple claims).
- NewsArticle → LogicalFallacyDetection: One-to-many (each article may include several fallacy detections).
- LogicalFallacy → LogicalFallacyDetection: One-to-many (fallacy catalog entries link to many detections).

3.4-Class Diagram

The class diagram provides a static view of the system architecture, describing the structure of News Advance by showing its classes, attributes, and the relationships among them. It acts as the bridge between the functional requirements outlined in the use case diagram and the technical implementation that follows in the codebase.

Core Classes and Relationships

At the heart of the system, the **NewsSource** and **NewsArticle** classes represent the foundation of the aggregation pipeline:

- **NewsSource**: stores information about each news outlet, including its name, URL, and reliability score. Each source can provide multiple articles.

- **NewsArticle**: central model containing title, content, publication date, URL, and raw metadata. It is linked to a variety of analysis outputs, ensuring that articles are analyzed once and reused across modules.

The main relationships are summarized as follows:

- **NewsSource 1..*** → **NewsArticle**
- **NewsArticle 1** → **1 BiasAnalysis**
- **NewsArticle 1** → **1 SentimentAnalysis**
- **NewsArticle 1** → **1 Summary**
  NewsArticle 1.. → *ArticleInsight**
  NewsArticle 1.. → *FactCheckResult**
  NewsArticle 1.. → *LogicalFallacyDetection**

- *LogicalFallacy 1..* → *LogicalFallacyDetection**
  NewsArticle ↔ **MisinformationAlert** (optional link)
  User 1.. → *UserSavedArticle (join model with NewsArticle)**

Key Analytical Classes

- **BiasAnalysis**: stores fields such as `political_leaning`, `bias_score`, and `confidence`. A strict one-to-one mapping ensures each article has a single, authoritative bias classification.

- **SentimentAnalysis**: includes `sentiment_score`, `subjectivity_score`, and optional model explanations.

- **Summary**: keeps concise AI-generated abstracts, either from the custom BART model or Ollama fallback.

- **ArticleInsight**: stores bullet-point insights extracted from the article. Modeled as one-to-many to capture multiple independent insights.

- **FactCheckResult**: holds claim text, verification rating (true, false, unverified), confidence scores, and evidence citations.

- **LogicalFallacy**: serves as a catalog curated by administrators.

- **LogicalFallacyDetection**: links a NewsArticle to one or more LogicalFallacy entries, with optional evidence excerpts and character spans.

- **MisinformationAlert**: allows admins to flag articles linked to trending or harmful misinformation.

User-Related Classes

- **User**: represents system users with secure credentials and saved preferences (e.g., toggling summaries or fact-checks).

- **UserSavedArticle**: a join model that links users with the articles they choose to bookmark. This structure preserves flexibility, as a single article can be saved by many users without duplicating content.

Design Benefits

This class structure ensures:

- **Modularity**: each analysis type is independent, making it easy to add or improve models without redesigning the schema.

- **Extensibility**: new analysis classes (e.g., multilingual processing, advanced fallacy detection) can be added as separate modules.

- **Efficiency**: articles are stored only once but linked to multiple analyses, avoiding duplication.

- **Maintainability**: the clear separation of concerns allows administrators and developers to update, re-analyze, or extend the system with minimal disruption.

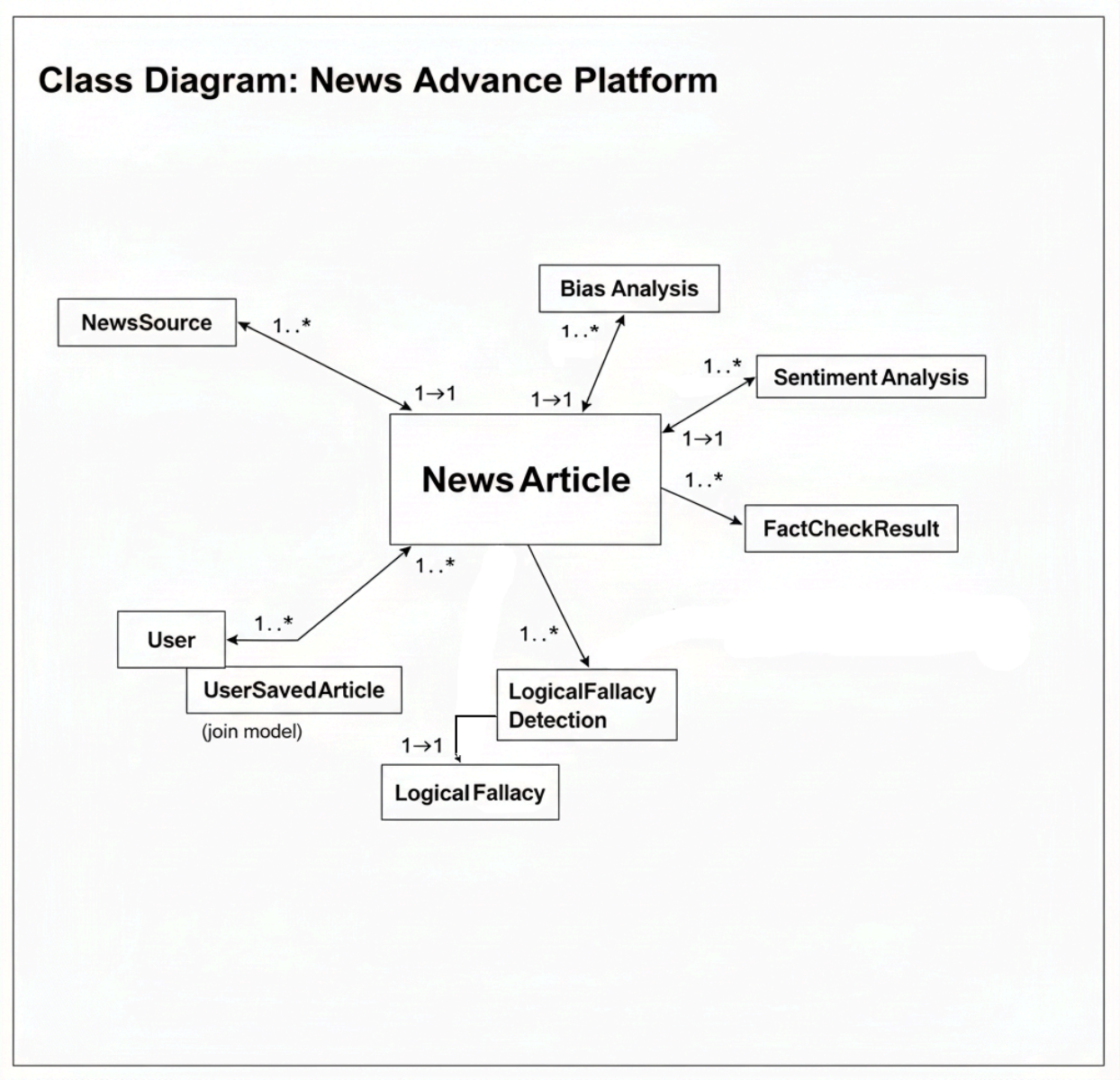


Figure 6: Class Diagram

Representative model excerpts:

```
class NewsArticle(models.Model):
    title = models.CharField(max_length=255)
    source = models.ForeignKey(NewsSource, on_delete=models.CASCADE)
    url = models.URLField(unique=True)
    published_date = models.DateTimeField(default=timezone.now)
    content = models.TextField()
    summary = models.TextField(blank=True)
```

```
class FactCheckResult(models.Model):
    article = models.ForeignKey(NewsArticle,
on_delete=models.CASCADE)
    claim = models.TextField()
    rating = models.CharField(max_length=20)
    explanation = models.TextField()
    confidence = models.FloatField(null=True, blank=True)
```

This structure keeps analyses independent, enables incremental additions (e.g., topic modeling), and avoids data duplication by anchoring all analysis artifacts to a single NewsArticle instance.

3.5-Conclusion

This chapter highlighted how modeling and conception transformed the business vision of News Advance into a concrete architectural framework. The use case diagram captured the interactions between users, administrators, and the system, while the class diagram defined the core entities and their relationships. Together, these models established clarity, reduced ambiguity, and provided a roadmap for efficient development.

By relying on UML as a design language, we created a structure that is not only technically sound but also adaptable to future enhancements such as advanced fact-checking or multilingual support. The modeling stage ensured that every subsequent coding effort would align with the overall system vision.

In short, this chapter defined the blueprint of News Advance. With the architecture validated, the project was ready to enter the implementation phase, where these models would be translated into working code and tangible functionality.

Chapter 4: From Code to Experience — Implementation and Demonstration

4.1-Introduction

This chapter presents the implementation and demonstration of *News Advance*, moving from conceptual models to a fully functioning system. After establishing the business requirements, selecting tools, and modeling the architecture in previous chapters, the focus here shifts to the practical realization of those designs.

The implementation phase covers the platform and tools used, the Django MVT framework, backend customizations, and the APIs that enable communication across components. It also details the aggregation and analysis pipelines, including sentiment detection, political bias classification, logical fallacy identification, summarization, and fact-checking. Special emphasis is placed on the fallback strategies that ensure robustness even when AI models are unavailable.

Alongside backend processes, the chapter demonstrates the user-facing experience. Screenshots illustrate key interactions such as registration, login, profile management, and article exploration. User interface design emphasizes accessibility, preference persistence, and responsive layouts, while administrative functions showcase system oversight through curated sources, fallacy catalogs, and reliability scoring.

Finally, the development timeline, testing strategy, and deployment setup highlight the methodology and quality assurance measures taken to deliver a secure, maintainable, and production-ready platform. Together, these sections provide a comprehensive view of how *News Advance* was implemented and how it operates in practice.

4.2-Platform and Tools used

- Framework: Django 5.2 (MVT), Python 3.10+
- Databases: SQLite (dev), PostgreSQL (prod target)
- NLP/AI: NLTK (VADER), spaCy (en_core_web_sm), Transformers + Torch (BART summarization), Ollama (local LLMs)

- Scraping: newspaper3k, BeautifulSoup4
- UI: Bootstrap 5 static assets
- Dev: PyCharm/VS Code, python-dotenv

Configuration highlights:

```
OLLAMA_ENDPOINT = 'http://localhost:11434/api/generate'
SUMMARIZATION_MODEL_DIR = BASE_DIR / 'news_analysis' / 'ml_models' /
'summarization' / 'trained_model'
SUMMARIZATION_BASE_MODEL = 'facebook/bart-base'
USE_ML_SUMMARIZATION = True
```

Django community configuration:

1. Set up a virtual environment.
2. Install Django. (pip install Django)
3. Create a project (django-admin startproject project_name)
4. Create an app (python manage.py startapp app_name)

PostgreSQL configuration:

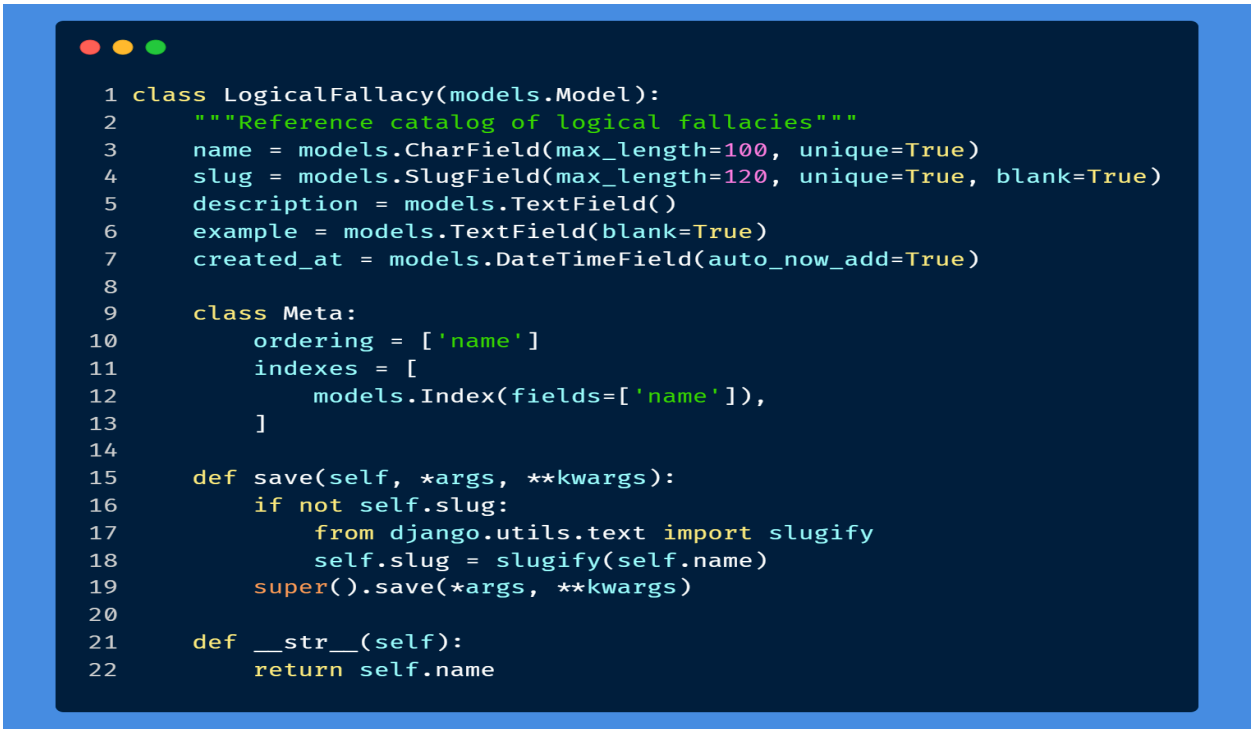
5. Download PgAdmin
6. Create Login/Group Role which is going to be the DB user and password
7. Create Database which is going to be the DB name

4.3-MVT: (Model-View-Template) in this project

Django follows the Model-View-Template (MVT) architectural pattern, which is similar to the Model-View-Controller (MVC) pattern.

In MVT, the Model represents the data and business logic of the application, the View handles user interaction and presentation logic, and the Template handles the visual representation of the data.

Model: The Model represents the database schema and the data stored in it. It defines the structure of the data and the relationships between different data entities. In Django, the Model is defined using a Python class that inherits from django.db.models.Model and defines the fields and their types.



```python
class LogicalFallacy(models.Model):
    """Reference catalog of logical fallacies"""
    name = models.CharField(max_length=100, unique=True)
    slug = models.SlugField(max_length=120, unique=True, blank=True)
    description = models.TextField()
    example = models.TextField(blank=True)
    created_at = models.DateTimeField(auto_now_add=True)

    class Meta:
        ordering = ['name']
        indexes = [
            models.Index(fields=['name']),
        ]

    def save(self, *args, **kwargs):
        if not self.slug:
            from django.utils.text import slugify
            self.slug = slugify(self.name)
        super().save(*args, **kwargs)

    def __str__(self):
        return self.name
```

Figure 7: Model

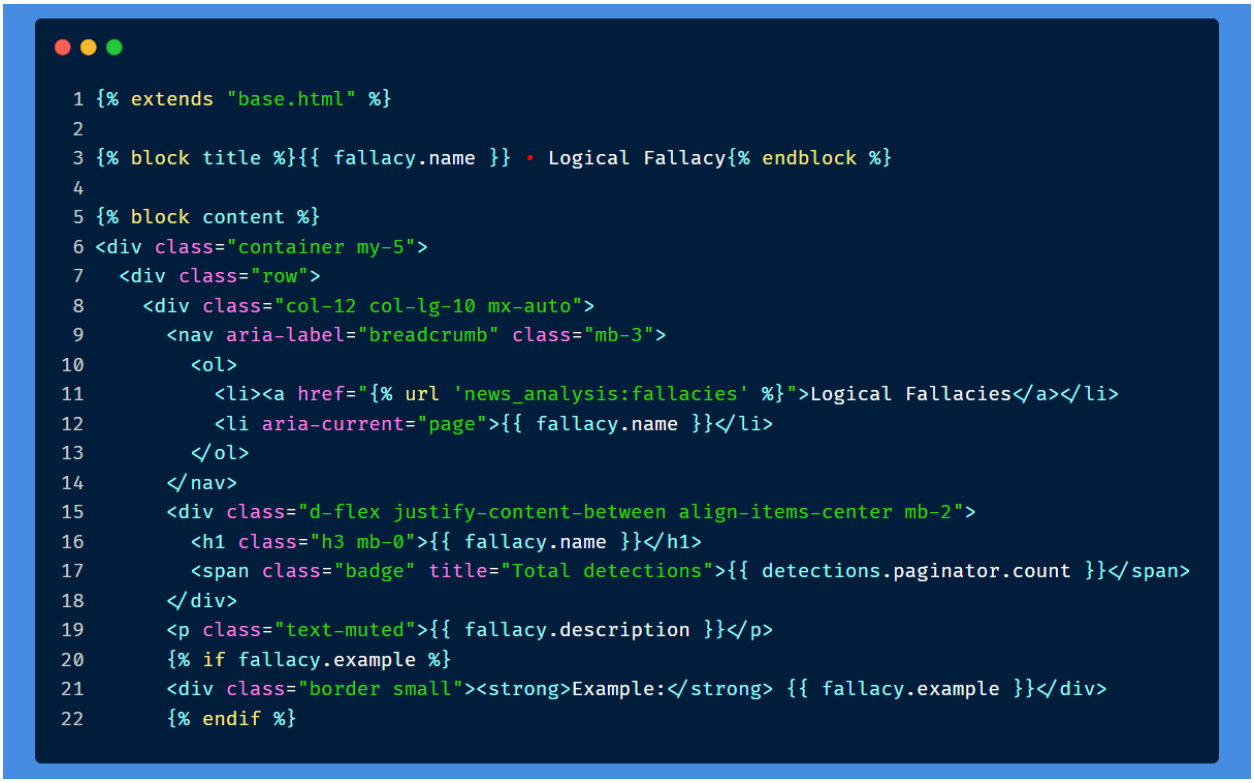
View: The View is responsible for handling user requests, interacting with the Model to retrieve or manipulate data, and returning an HTTP response. In Django, the View is implemented as a Python function or class-based view that receives an HTTP request and returns an HTTP response.

```python
def fallacy_detail(request, slug):
    """Detail page for a single logical fallacy with related detections/articles."""
    fallacy = get_object_or_404(LogicalFallacy, slug=slug)
    detections_qs = (
        LogicalFallacyDetection.objects
        .filter(fallacy=fallacy)
        .select_related('article', 'article__source')
        .order_by('-detected_at')
    )
    paginator = Paginator(detections_qs, 20)
    page_number = request.GET.get('page')
    detections = paginator.get_page(page_number)

    context = {
        'fallacy': fallacy,
        'detections': detections,
    }
    return render(request, 'news_analysis/fallacy_detail.html', context)
```

Figure 8: View

Template: The Template is responsible for rendering the data returned by the View into a visual format that can be presented to the user. In Django, the Template is implemented using HTML and other web development technologies, and it can include placeholders for dynamic data that is populated by the View.



```
{% extends "base.html" %}

{% block title %}{{ fallacy.name }} · Logical Fallacy{% endblock %}

{% block content %}
<div class="container my-5">
  <div class="row">
    <div class="col-12 col-lg-10 mx-auto">
      <nav aria-label="breadcrumb" class="mb-3">
        <ol>
          <li><a href="{% url 'news_analysis:fallacies' %}">Logical Fallacies</a></li>
          <li aria-current="page">{{ fallacy.name }}</li>
        </ol>
      </nav>
      <div class="d-flex justify-content-between align-items-center mb-2">
        <h1 class="h3 mb-0">{{ fallacy.name }}</h1>
        <span class="badge" title="Total detections">{{ detections.paginator.count }}</span>
      </div>
      <p class="text-muted">{{ fallacy.description }}</p>
      {% if fallacy.example %}
      <div class="border small"><strong>Example:</strong> {{ fallacy.example }}</div>
      {% endif %}
```

Figure 9: Template

The MVT pattern in Django emphasizes the separation of concerns between the Model, View, and Template, making it easier to maintain and update the application code. It also provides a modular structure that allows developers to reuse code and extend the application functionality as needed.

4.4-Backend admin customization and data management

In Django, the main backend dashboard is usually referred to as the Django Admin site. It is a built-in feature of Django that provides a powerful interface for managing the application's data models and performing administrative tasks.

The Django Admin site allows authorized users to log in and access various functionalities such as adding, editing, and deleting data objects. It also provides features for managing user authentication, creating custom views, and generating reports.

The Django Admin site is automatically generated based on the application's defined data models, which means that developers do not need to write any additional code to create the dashboard. The interface provides a user-friendly and customizable way to manage the

application's data and provides a quick way to access all the data models registered in the application.

The Django Admin site provides a high level of security, as it requires user authentication to access its functionalities. It also allows developers to control the access and permissions of different users to the dashboard and its functionalities.

Customizing the Backend Dashboard:

While the default backend dashboard provides a functional and straightforward interface for managing the database, it may not be suitable for all users or organizations. Therefore, the dashboard can be customized to better fit the needs of the users and the project.

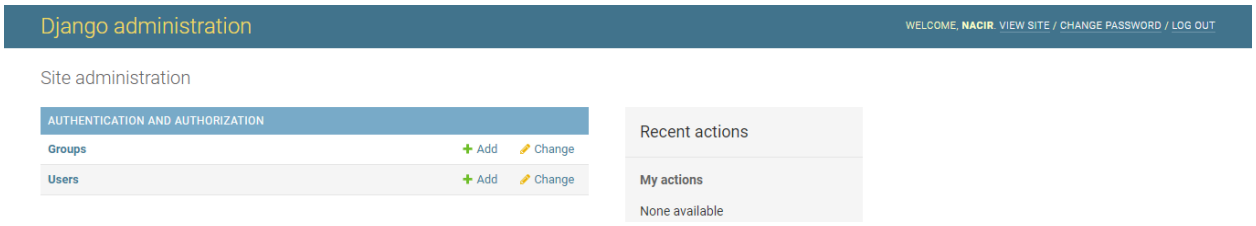


Figure 10: Uncustomized Main CMS Dashboard

In this example, we see the default backend dashboard with its standard look and feel. However, users can modify the dashboard's appearance, including changing the color scheme, typography, and layout, to make it more user-friendly and aligned with their brand identity.

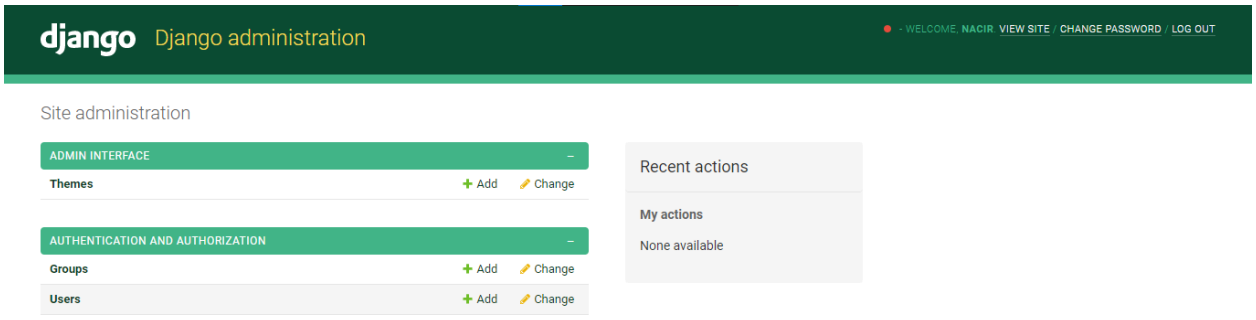


Figure 11: Customized CMS Dashboard

Here, we see an example of a customized dashboard with a distinct color scheme and layout, providing a unique and appealing look and feel.

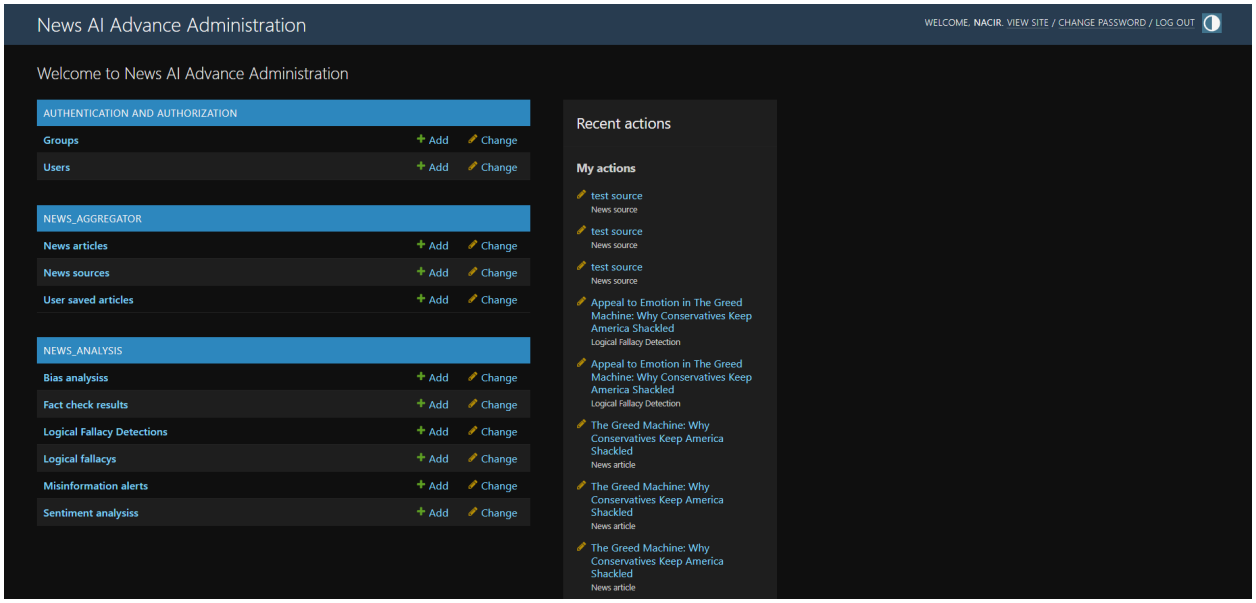


Figure 12: Main CMS Dashboard

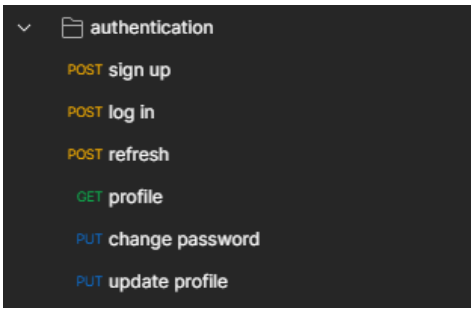
Another example of a customized dashboard, with a different color scheme and typography, making it more legible and easier to use.

Customizing the backend dashboard allows users to create a more personalized and functional experience when interacting with the system. However, it's important to ensure that the customizations do not compromise the system's security, functionality, or usability.

4.5-Application Programming Interface (APIs)

An API, or Application Programming Interface, is a set of rules and protocols that allow different software applications to communicate with each other. In the context of a Django project, an API is a way to expose your application's data and functionality to external clients, such as mobile apps or other web services.

APIs are typically built using a combination of web technologies, such as HTTP, XML, and JSON. They provide a standardized way for clients to request data and perform actions, using a set of well-defined endpoints and parameters.



In the case of your Django project, you might use APIs to allow external clients to perform tasks such as register new users, login, and get a profile, or get the processed articles and make a new ui for displaying them. By providing APIs, you can enable other developers to build on top of your application, creating new integrations and value-added services.

Figure 13: APIs requests

A compact JSON endpoint exposes misinformation alerts for a given article:

```
path('api/articles/<int:article_id>/misinformation-alerts/',
article_misinformation_alerts, name='article_misinformation_alerts')
```

Example response:

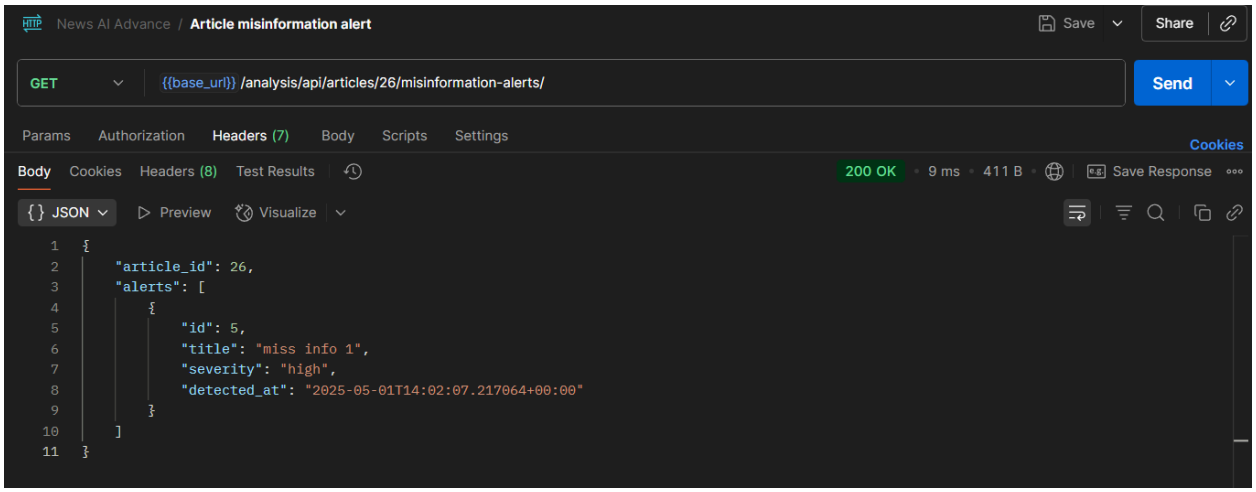


Figure 14: Misinformation Alerts API

The server validates the request and retrieves the appropriate information from the database before returning the response to the user. The specific fields and validation requirements for GET requests may depend on the business requirements and data model of the application.

While there may be many other APIs available in the system, that are used internally for better user experience like saving article api consumed by the AJAX calls in some pages. However, depending on the business requirements and use cases of the system, additional APIs may be necessary to provide more advanced functionality or integration with other systems.

4.6-Aggregation pipeline and management commands

- fetch_news: scrape sources with newspaper3k, normalize content, store NewsArticle and link to NewsSource.
- analyze_articles: run bias, sentiment, summarization (ML with Ollama fallback), key insights, logical fallacy detection, and fact-checking.
- generate_test_data: seed sources/articles for development.
- recalculate_reliability: recompute NewsSource.reliability_score from recent fact checks, fallacy frequency, and bias consistency.
- send_misinformation_alerts: compile and optionally notify admins/users of trending alerts.

4.7-Analysis pipeline overview

- Input: NewsArticle content
- Preprocess: clean HTML, normalize whitespace, tokenize

- Sentiment: VADER baseline; optional Ollama-enhanced sentiment with explanation
- Bias: LLM classification with score and confidence
- Summarization: BART model or Ollama fallback
- Insights: bullet-point extraction
- Logical fallacies: rule/LLM-assisted detection persisted per article
- Fact-checking: claim extraction + verification + rating/confidence

Claim extraction scoring (excerpt):

```
def score_sentence(s: str) -> float:
    if len(s) < 40 or len(s) > 300: return -1
    score = 0.0
    if re.search(r"[0-9]|%|\$", s): score += 1.5
    if '"' in s or "'" in s: score += 0.4
```

4.8-Fact-checking and logical fallacies

- FactCheckResult stores claim text, rating (true/mostly_true/.../unverified), confidence, explanation, and sources.

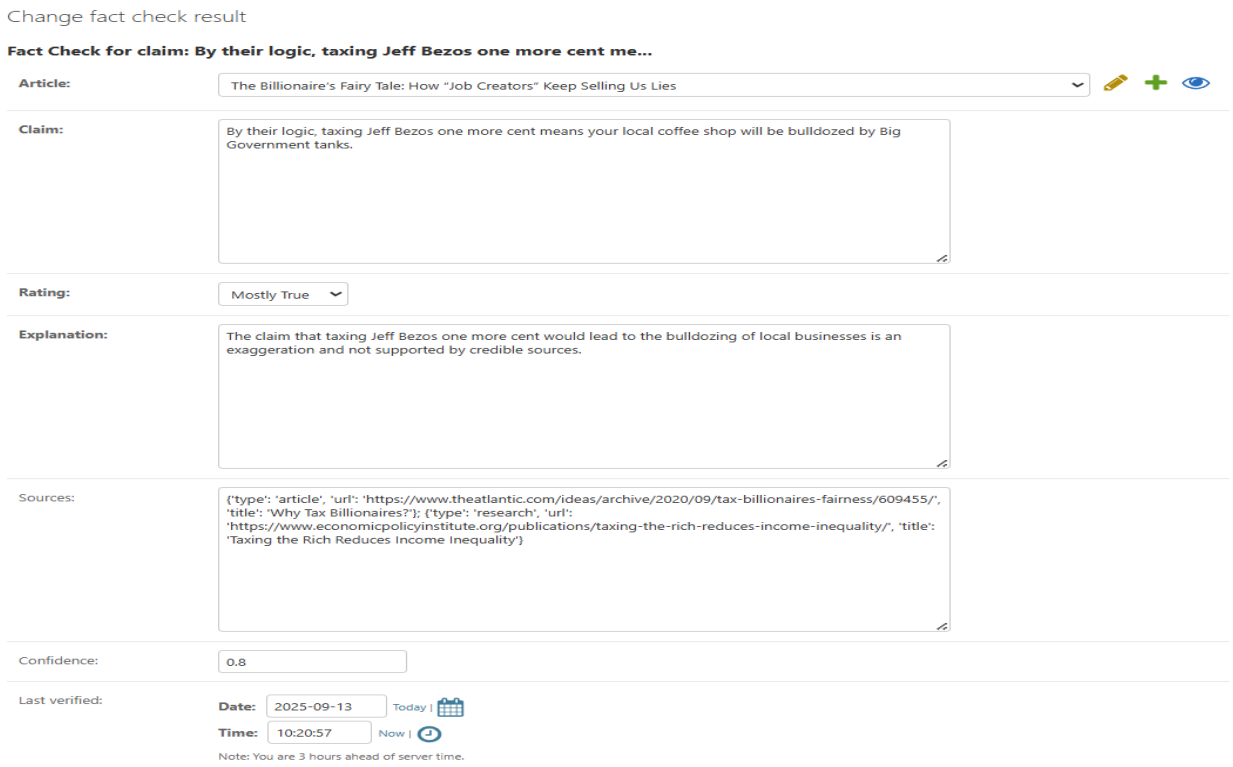


Figure 15: Fact check edit

- LogicalFallacy catalog is maintained in admin; detections are stored per article with optional evidence excerpt and character spans.
- Admins can review detections and fact checks, re-run verification, and attach alerts for trending misinformation.

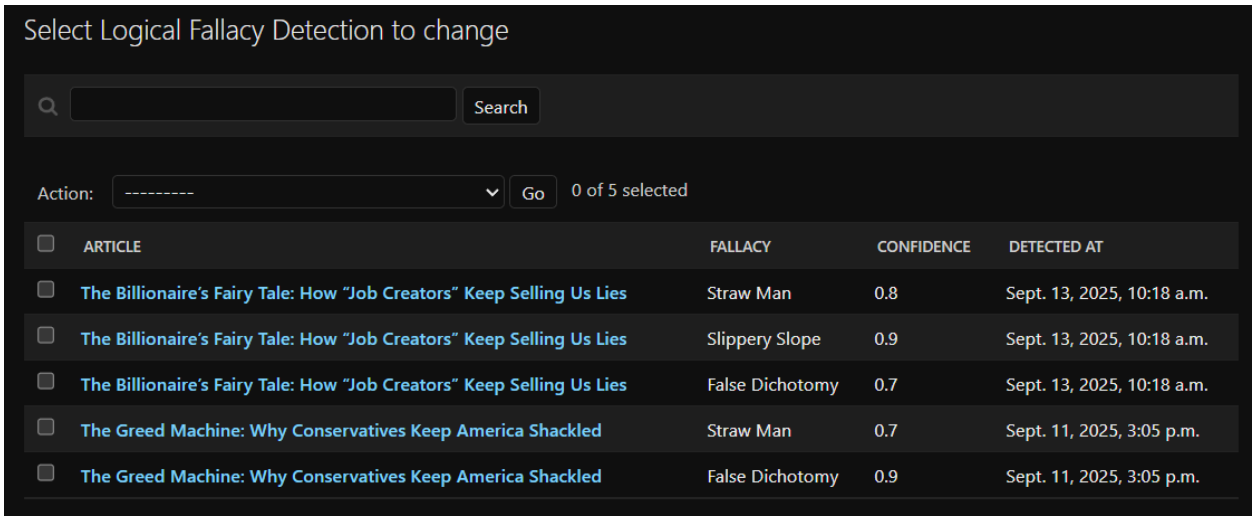


Figure 16: Logical fallacies detections for articles

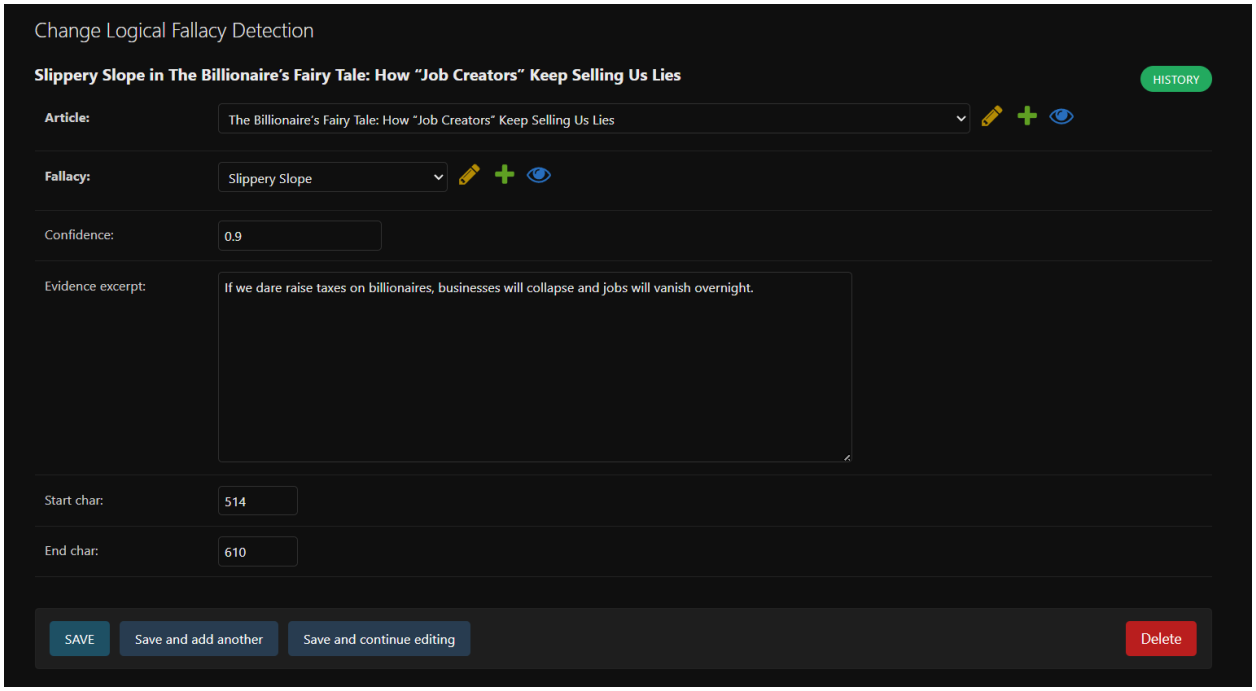


Figure 17: Logical fallacy detection edit

4.9-Summarization and fallback strategy

- Default: use fine-tuned BART from SUMMARIZATION_MODEL_DIR
- Fallback: if model missing or errors, send prompt to Ollama (at OLLAMA_ENDPOINT)
- Summaries are cached on the article record and refreshed if content changes

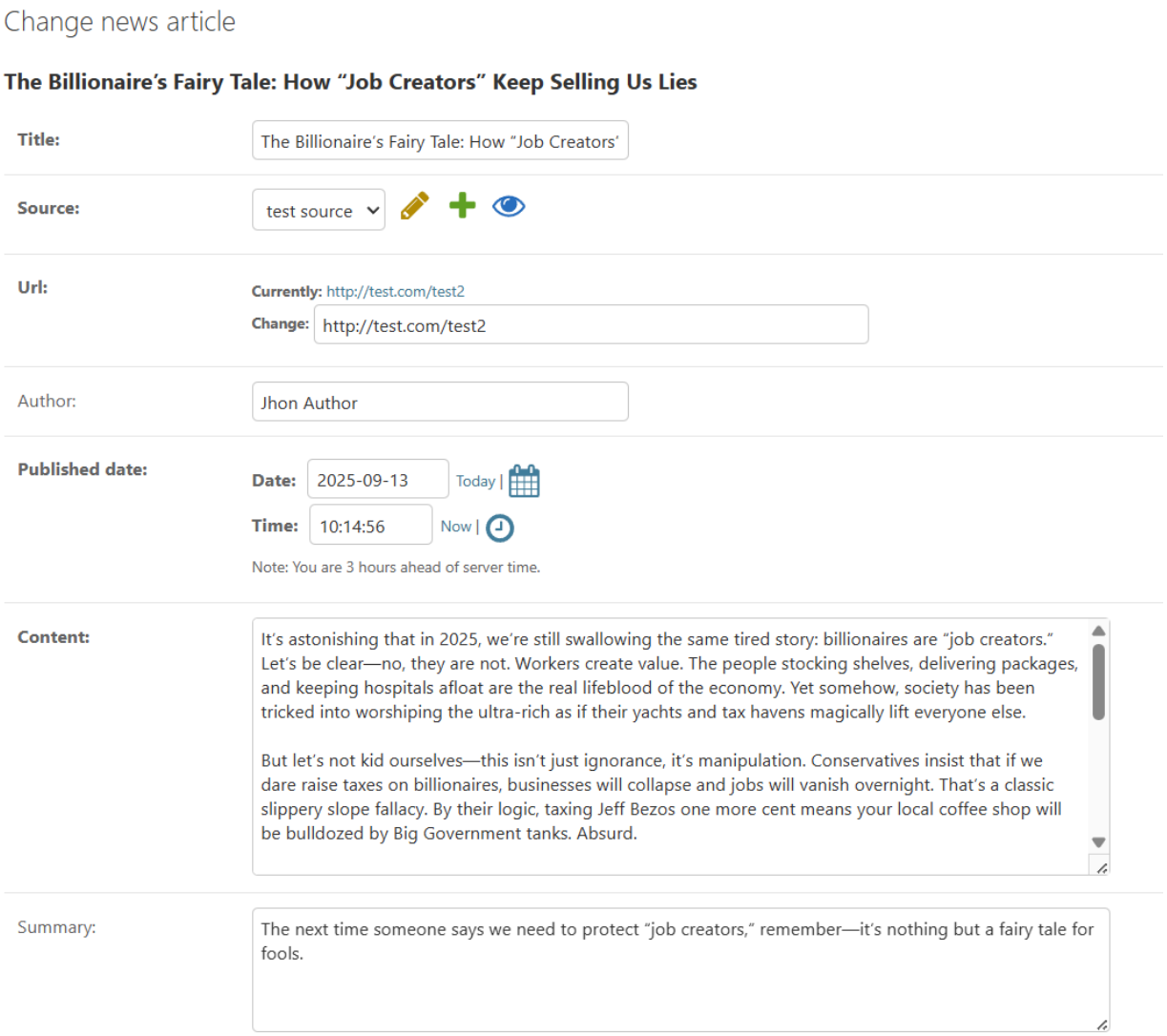


Figure 18: Article content and summary

4.10-UI/UX and user preferences

- Article page sections: AI summary, key insights, sentiment, bias, fallacies, fact checks, alerts

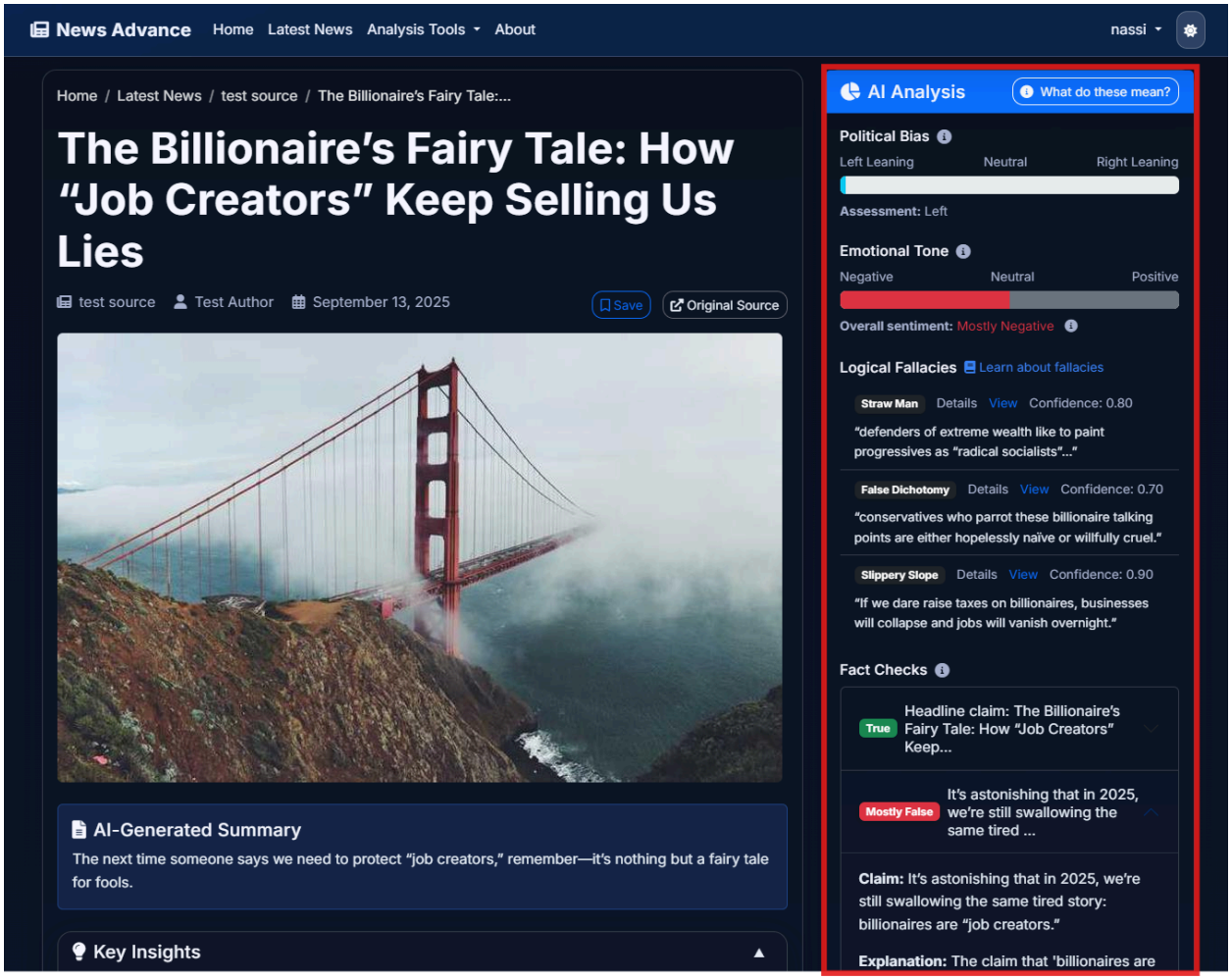


Figure 19: Article AI analysis section

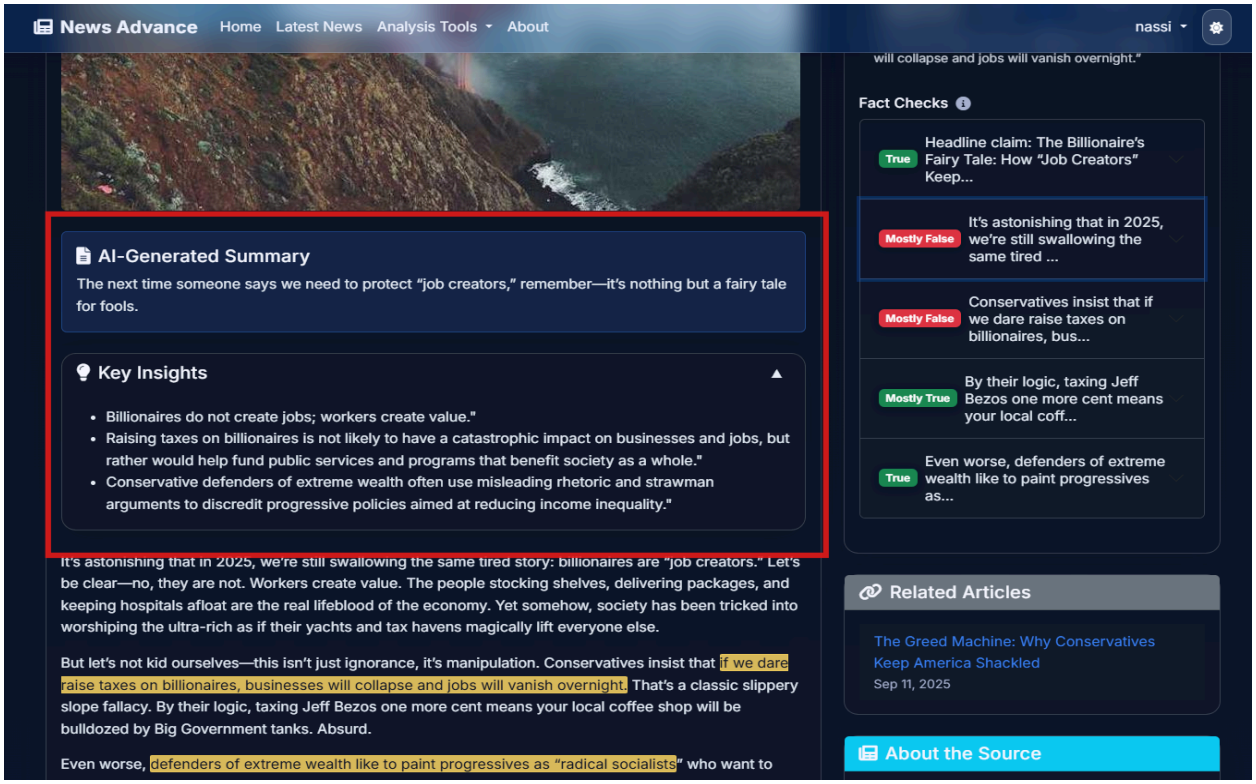


Figure 20: Article AI Summary and key insights section

- User toggles: show/hide summary, insights, and Analysis; persist preference

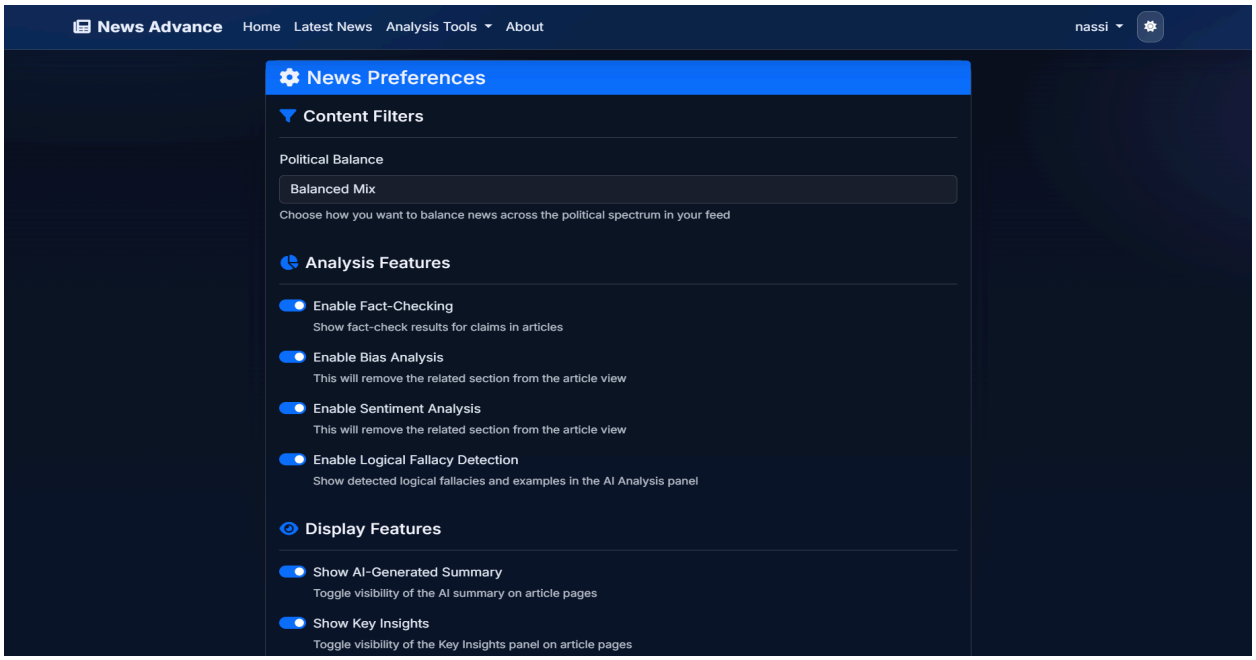


Figure 21: User preferences page

- Accessibility: semantic headings, keyboard navigation, and minimal color dependency

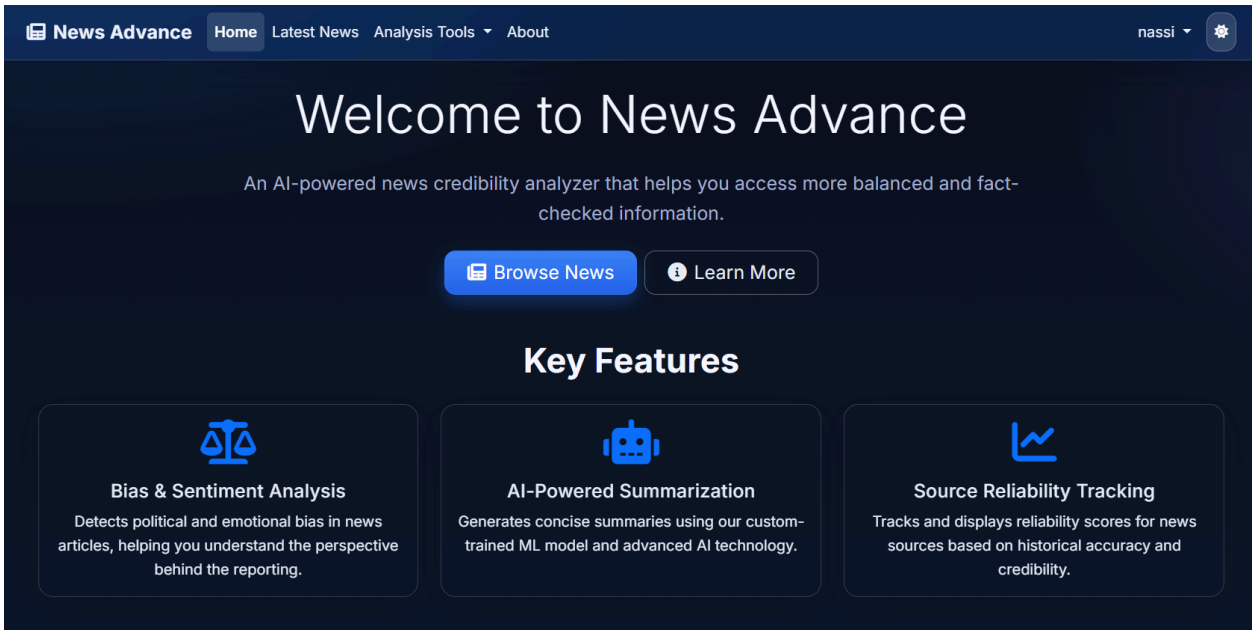


Figure 22: Home page

4.11-Source reliability scoring

- Inputs: recent FactCheckResult ratings distribution, LogicalFallacyDetection frequency, BiasAnalysis variance vs. claimed leaning
- Weights: fact checks (~60%), bias consistency (~20%), fallacy frequency (~20%)
- Output: 0–100 reliability_score on NewsSource; recalculated via management command

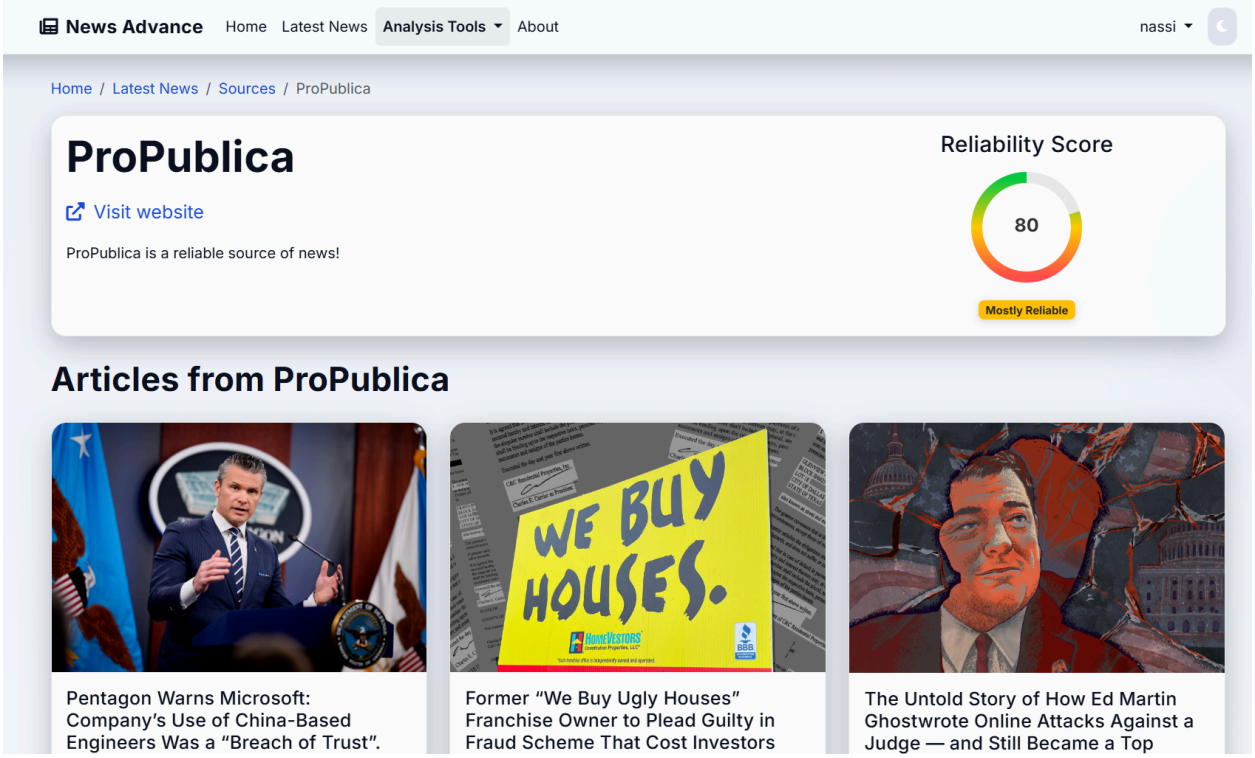
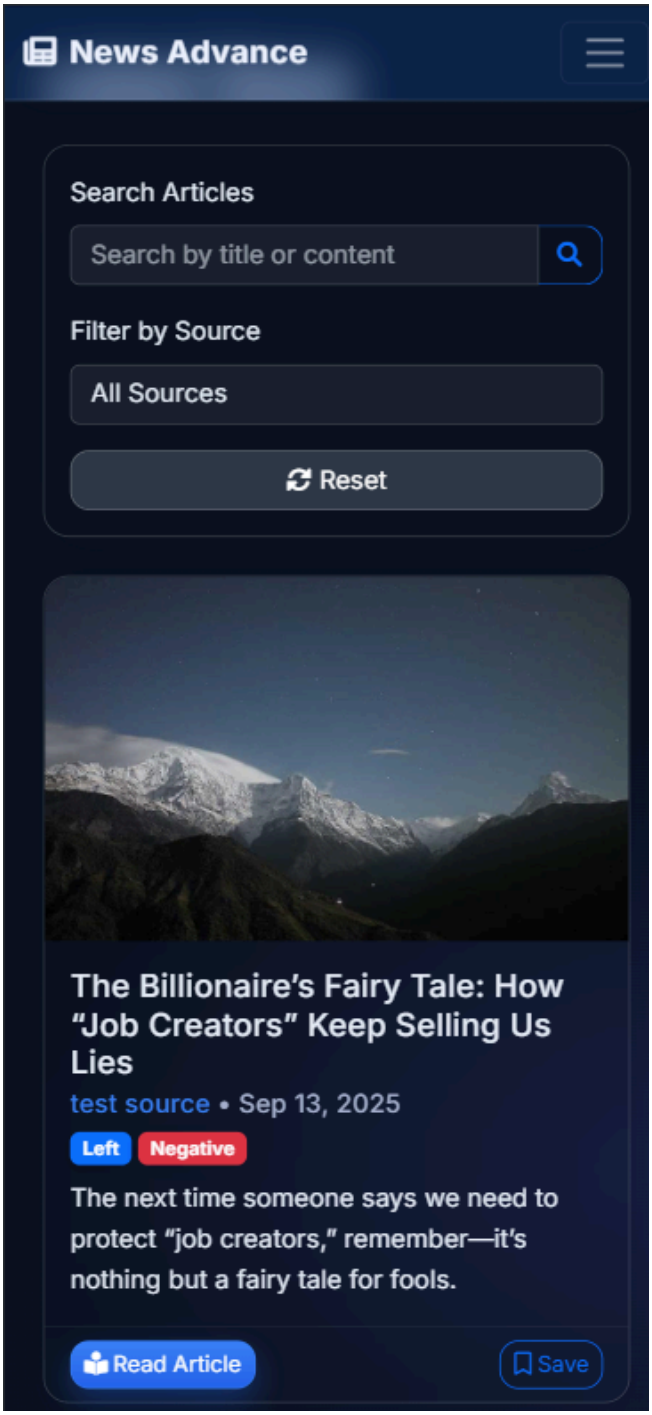


Figure 23: Source articles and reliability score

4.12-UI/UX: screens, interactions, and error handling

- Navigation and information architecture:
  - Landing ⟶ Sources ⟶ Article list ⟶ Article details with analysis panels (Summary, Insights, Sentiment, Bias, Fallacies, Fact checks, Alerts)
  - Admin routes for curation (sources, articles, fallacy catalog, fact checks)
- Article details layout:
  - Left: original article metadata (source, date, URL) and raw content toggle
  - Right: stacked analysis cards with clear status, timestamps, and confidences
- User preferences and state:
  - Per-user toggles to show/hide summary, insights, and fact checks; persisted and respected server-side
  - Sticky last-viewed section; deep links to specific panels via anchors
- Feedback and failure UX:
  - Graceful fallbacks: ML summary ⟶ Ollama; missing fact checks render a clear CTA to re-run
  - Error banners carry correlation IDs (server logs) for quick triage

- Accessibility and responsiveness:
    - Bootstrap 5 semantics, keyboard navigation, contrast-safe palette, mobile-first cards



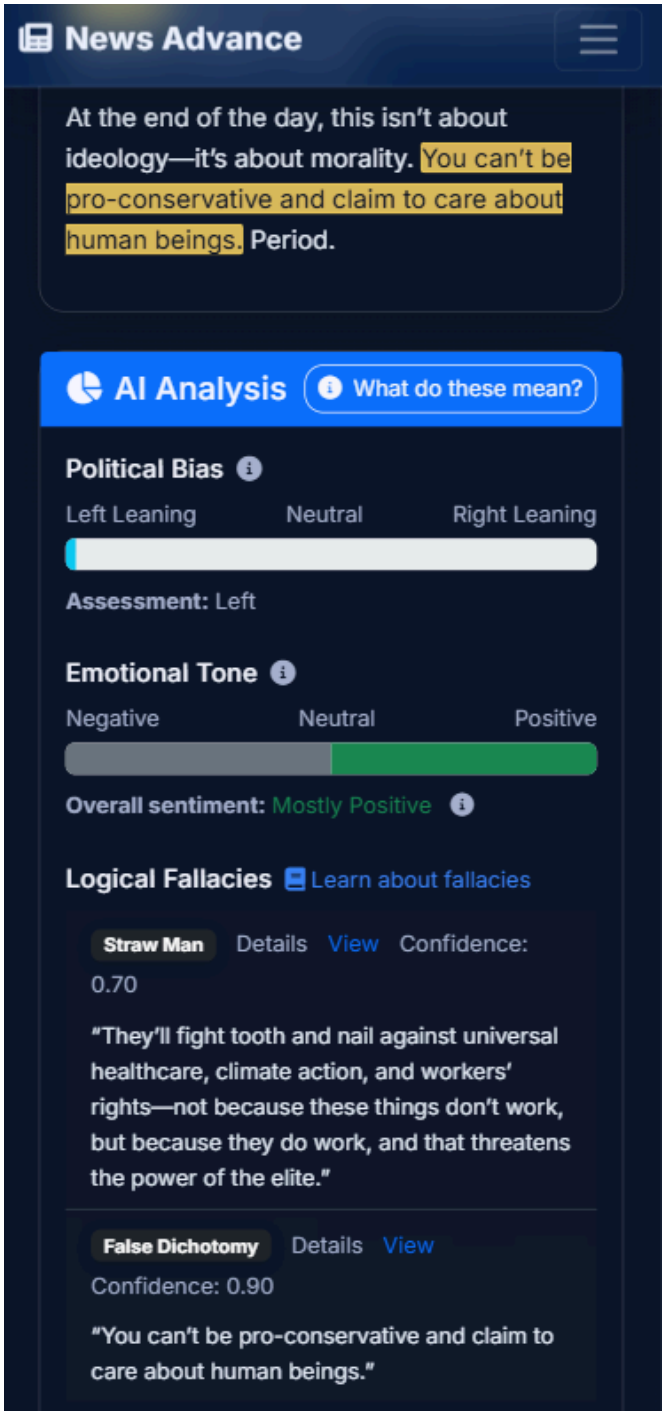


Figure 24: Responsive on mobile view

4.13-Development timeline and methodology

- Week 1–2: Project scaffolding (Django apps), settings, admin baselines, initial models
- Week 3–4: Aggregation pipeline with newspaper3k, normalization, and admin curation flows
- Week 5–6: Analysis pipeline v1 (sentiment, bias, summarization with ML+fallback); insights extraction
- Week 7–8: Logical fallacy detection, fact-checking (claims+verification), reliability scoring
- Week 9: API endpoint for misinformation alerts; UI/UX polish and preferences
- Week 10: Stabilization, documentation, and memoir alignment with the codebase
- Practice: small iterative commits, feature flags for ML usage, and reproducible seeds for demos

4.14-Testing strategy and QA

- Unit tests (targeted):
    - Claim extraction scoring, bias/sentiment classification wrappers, summarization fallback selection
    - Reliability score computation determinism on fixed inputs
- Integration checks (scripted):
    - End-to-end analyze_articles on a small fixture set; verify all analysis objects created once
    - API: GET /api/articles/misinformation-alerts returns structured JSON and correct severities
- Mocking and isolation:
    - Ollama calls stubbed in tests; deterministic fixtures for NLP models where practical
- Manual QA:
    - Admin smoke tests (create/edit/delete sources, articles, fallacy catalog)
    - UI spot checks for empty/loading/error panels and user preference persistence

4.15-Deployment and production setup

- Environment and configuration:
    - DEBUG=False, allowed hosts, secure cookies, CSRF/HTTPS, SECRET_KEY via env
    - Database: SQLite for dev; PostgreSQL for production
    - Static files: collectstatic with hashed filenames; whitenoise or CDN as needed
- Services:
    - Optional: Ollama service deployed on same host or isolated node; health checks before analysis

    - Planned: Celery + Redis for scheduled ingestion and async pipeline runs
- Observability and reliability:
    - Structured logs for each pipeline stage with article/source IDs
    - Correlation IDs surfaced in UI error banners for quick log lookup
- Data hygiene:
    - Guardrails for duplicate URLs, charset/encoding normalization, and content length limits

4.16-Conclusion

The implementation and demonstration phase translated the project's theoretical framework into a concrete and interactive platform. By leveraging Django's MVT architecture, robust NLP and AI libraries, and a modular analysis pipeline, *News Advance* successfully delivers its core functionalities: news aggregation, summarization, sentiment and bias analysis, logical fallacy detection, fact-checking, and source reliability scoring.

On the user side, the platform offers a transparent and customizable reading experience. Preferences such as showing or hiding summaries and fact-checks are respected, while the article interface integrates AI-generated insights in an accessible and responsive design. Administrative workflows complement this by ensuring data quality, managing sources, and overseeing detection outputs.

Beyond technical implementation, the project followed a disciplined timeline, incorporating testing strategies, fallback mechanisms, and production-ready configurations. These ensured not only functional correctness but also resilience, scalability, and usability.

In summary, this chapter demonstrated how *News Advance* evolved from an abstract concept into a tangible platform, validating the design choices made in earlier chapters. With the system fully implemented and tested, the foundation is now set for critical evaluation and reflection in the concluding stages of this memoir.

Conclusion

The development of News Advance has been a journey from abstract ideas about media transparency to a tangible platform that empowers users to engage with news more critically. Beginning with the identification of business requirements, the project outlined key functions such as aggregation, summarization, bias and sentiment analysis, fact-checking, logical fallacy detection, and reliability scoring. Each of these features was carefully mapped to technical requirements, supported by the deliberate selection of tools, frameworks, and natural language processing models.

Through modeling and conception, the system's complexity was distilled into a coherent architecture, ensuring modularity, extensibility, and maintainability. The subsequent implementation phase validated these designs, producing a working platform that not only integrates advanced AI techniques but also provides a clear and intuitive user experience. Screenshots of registration, article exploration, and analysis panels demonstrated how the platform balances functionality with accessibility, while administrative features ensured oversight and data integrity.

The project achieved its central goal: creating a system that enhances the reliability, clarity, and transparency of news consumption in an age of overwhelming information and misinformation. While technical challenges were encountered—such as ensuring fallback mechanisms, training summarization models, and integrating diverse NLP pipelines—they were addressed through disciplined development, modular design, and iterative testing.

Looking ahead, News Advance can be extended with multilingual support, real-time misinformation alerts, deeper integration of fact-checking databases, and broader scalability through distributed pipelines. These enhancements would further strengthen its role as a practical tool for critical media literacy and trustworthy journalism.

In conclusion, this project demonstrates that the combination of natural language processing, large language models, and thoughtful system design can provide meaningful solutions to one of today's most pressing challenges: enabling individuals to navigate the news landscape with confidence, clarity, and critical awareness.

Perspective

Immediate enhancements:

- Celery/Redis: async ingestion and background analysis; scheduled re-analysis windows
- REST API: expand endpoints for sources, articles, fallacies, fact checks, and reliability scores
- Fact-checking with RAG: retrieval over reputable sources to ground LLM verification
- UI/UX: per-panel refresh; cross-source comparison; richer filtering and search
- Model upgrades: multilingual pipelines, domain-tuned summarizers, larger local LLMs (resource-aware)

References

- Django: https://www.djangoproject.com/
- Ollama: https://ollama.com/
- Hugging Face Transformers: https://huggingface.co/transformers/
- PyTorch: https://pytorch.org/
- NLTK (VADER): https://www.nltk.org/
- spaCy: https://spacy.io/
- newspaper3k: https://newspaper.readthedocs.io/
- Bootstrap: https://getbootstrap.com/