LEBANESE UNIVERSITY

Faculty of Economic Sciences

And Business Administration

Branch III

The News Advance Project

M2 Graduation Project

Specialty: Business Computer

Prepared By

Nacir Chahine

Jury

Dr. Dalia Batikh, Supervisor

Dr. , Coordinator

Table of Contents

Abstract

…… ……..….. 4

Acknowledgments

…… …… 5

Dedication

…… …….…… 6

General Introduction

…….…… … 7

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

…..……. 8

chapter1: Business Requirement

…… 9

1.functions and features

…….…… 9

2.tools and libraries requirement

……..…… 10

3.overview of Web Scraping
……. 10
4.overview Django
…….……
. 12
chapter2: Modeling and conception
……..……. 13
1.Unified Modeling Language (UML)
……..….. 13
2.Why UML?
……..……
……..…. 13
3.Goals of UML
…….……..……
…..…. 13
4.Use Case diagram
……..……
… 14
5.Class diagram
……..……
……. 16
Chapter3: Implementation Phase
……..…… 19

1.Platform and Tools used
…… 19

a. Django community configuration
…… 19

b. PostgreSQL configuration
…… 20

2.MVT (Model-View-Template)
…… 20

3. Backend admin customization and data management
……23

4. Application Programming Interface (APIs)
…… 25

5. Aggregation pipeline and management commands
…… 26

6. Analysis pipeline overview
…… 27

7. Fact-checking and logical fallacies
…… 27

8. Summarization and fallback strategy
…… 27

9. UI/UX and user preferences
…… 27

10. Source reliability scoring
…… 28

11. UI/UX: screens, interactions, and error handling
…… 28
12. Development timeline and methodology
…… 28
13. Testing strategy and QA
…… 29
14. Deployment and production setup
…… 29
Conclusion
…… 30
Perspective
…… 30
References…… 31
Table of figures
…… 31

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.Functions and features:

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.

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, BeautifulSoup4.
- 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.

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.

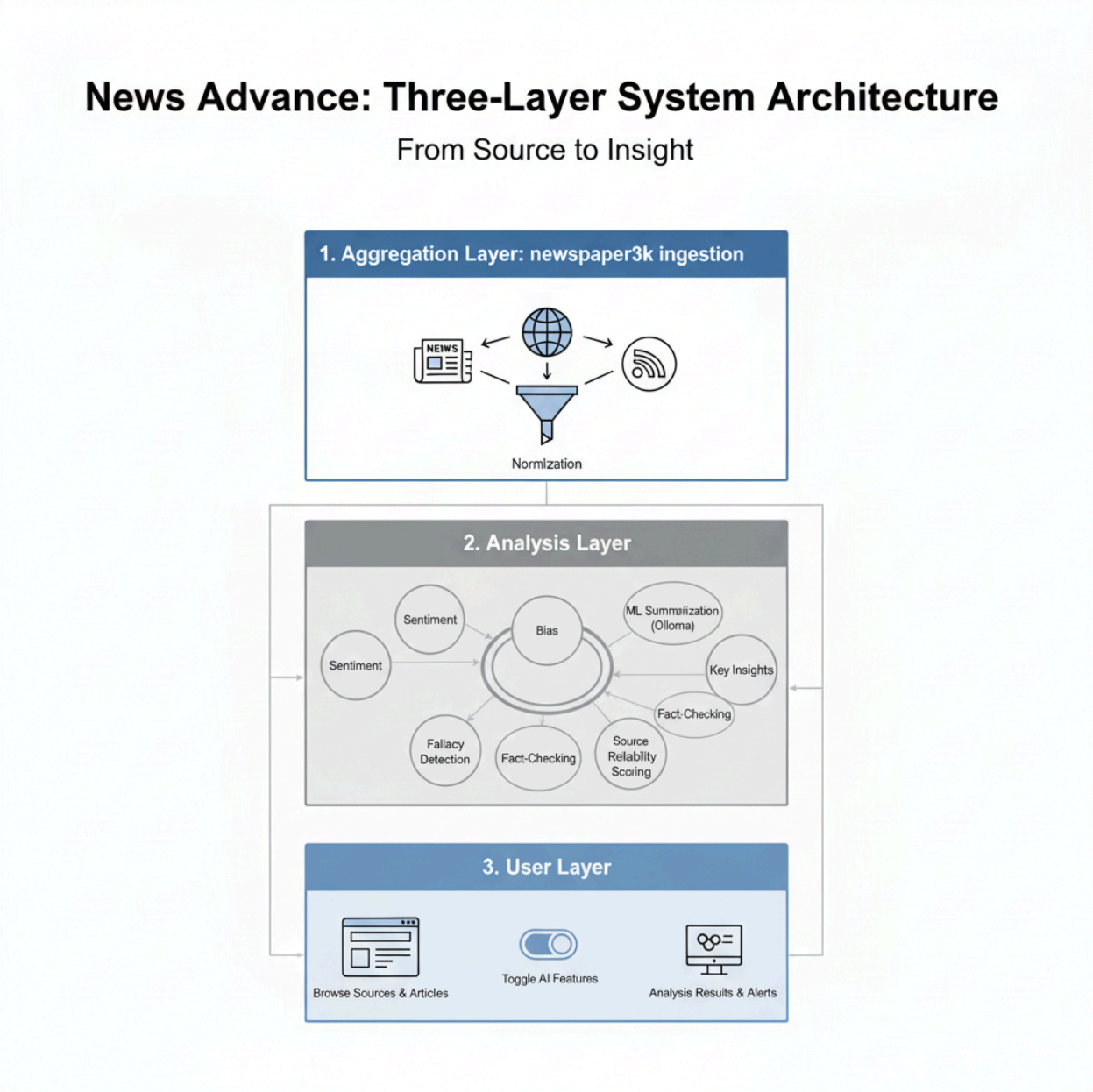


Figure 1: System Layers

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.

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.

Chapter2: Modeling and conception

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

Use Case Diagram:

A use case is a methodology used in system analysis to identify, clarify, and organize system requirements. The use case is made up of a set of possible sequences of interactions between systems and users in a particular environment and related to a particular goal.

Use Case Diagram Main actors and goals:

- Visitor/User: browse sources, read articles, view AI summary, sentiment, bias, fallacy detections, fact-check results; save articles.
- Admin: curate sources, manage logical fallacies, review fact checks, trigger re-analysis, manage misinformation alerts.

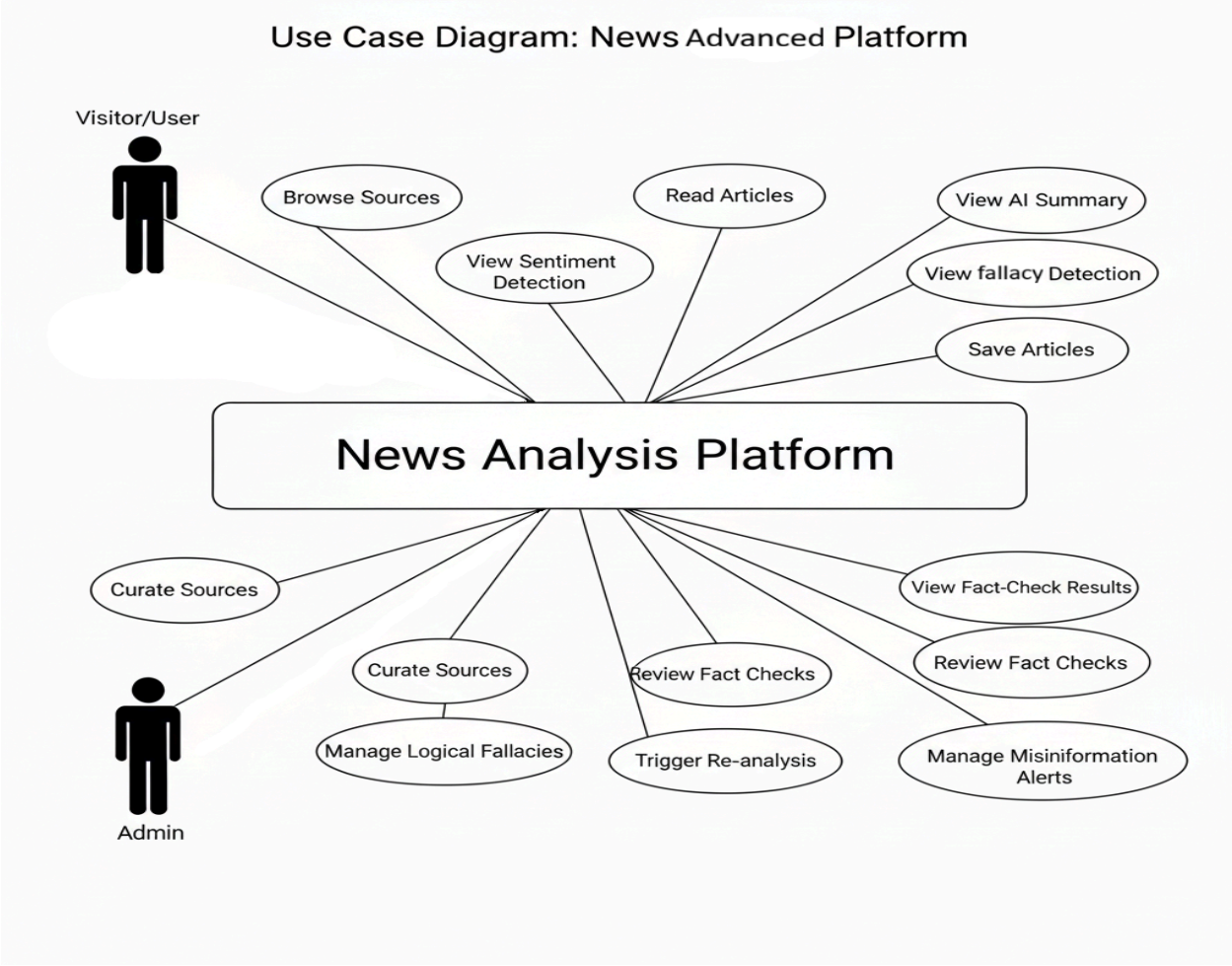


Figure 2: Use Case

Narrative: A user selects a source or searches; the system fetches or displays stored articles. On opening an article, the analysis pipeline surfaces summary, sentiment, bias, insights, fallacies, and any fact-checks; the user can save the article or adjust visibility preferences.

In our use case, there is a clear mapping between the system's real-world entities and the objects modeled in our class diagram. The diagram represents the main building blocks of News Advance and the relationships between them.

User

Represents individuals interacting with the system. Each user has an id, username, and password (securely stored). Users can save multiple news articles, which reflects a one-to-many relationship with the NewsArticle class.

NewsArticle

The central data model in the application. It contains fields like id, title, content, and published_date. Every article can be linked to both a BiasAnalysis and a SentimentAnalysis. This ensures that once an article is fetched and stored, it can be analyzed for multiple attributes without duplication of content.

BiasAnalysis

Stores the results of political bias detection. Fields like political_leaning and bias_score quantify the article's political orientation. A one-to-one relationship connects each BiasAnalysis to a single NewsArticle, ensuring analysis accuracy for that specific piece of content.

SentimentAnalysis

Holds the sentiment evaluation for an article. Attributes like sentiment_score and subjectivity_score describe the emotional tone and objectivity of the content. This class also maintains a one-to-one relationship with NewsArticle.

Relationship Summary:

User → NewsArticle: One-to-many (a user can save many articles).

NewsArticle → BiasAnalysis: One-to-one (each article has exactly one bias analysis).

NewsArticle → SentimentAnalysis: One-to-one (each article has exactly one sentiment analysis).

The diagram also indicates dependency between BiasAnalysis and SentimentAnalysis when combining results for advanced credibility scoring.

This structure ensures that the system is modular (each analysis type is separate), extensible (new analysis models like fact-checking can be added without redesign), and efficient (article content is stored only once but can be reused for multiple analysis processes).

Class Diagram:

A class diagram in the Unified Modeling Language (UML) is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among objects.

Class Diagram Core classes and relationships:

NewsSource 1..*→ NewsArticle

NewsArticle 1→1 BiasAnalysis, 1→1 SentimentAnalysis, 1..→ ArticleInsight, 1..→ FactCheckResult, 1..*→ LogicalFallacyDetection, ..↔ MisinformationAlert

User 1..*→ UserSavedArticle (join model)

LogicalFallacy 1..*→ LogicalFallacyDetection

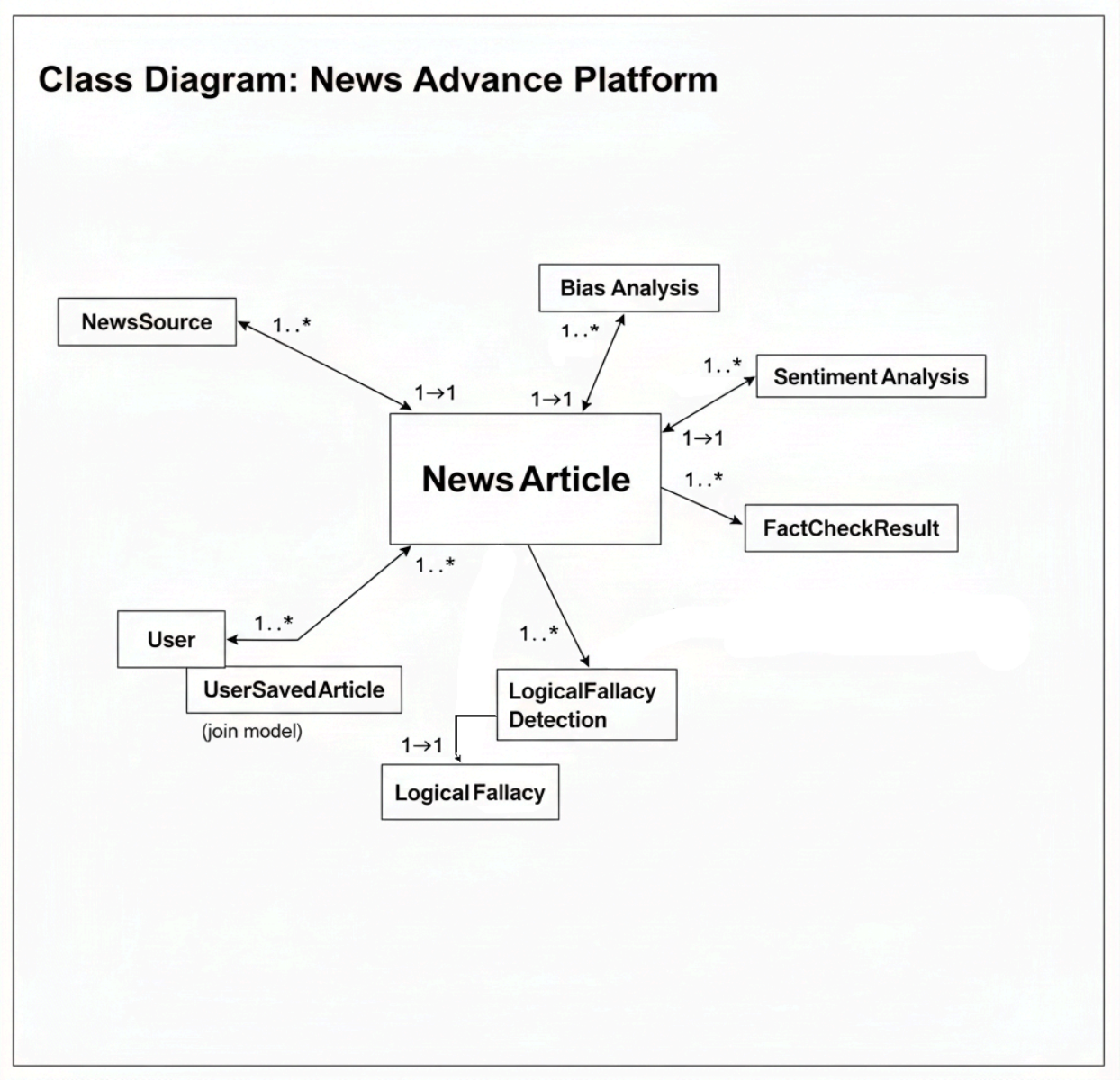


Figure 3: Class Diagram

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.

Chapter3: Implementation Phase

1.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

2.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 4: Model

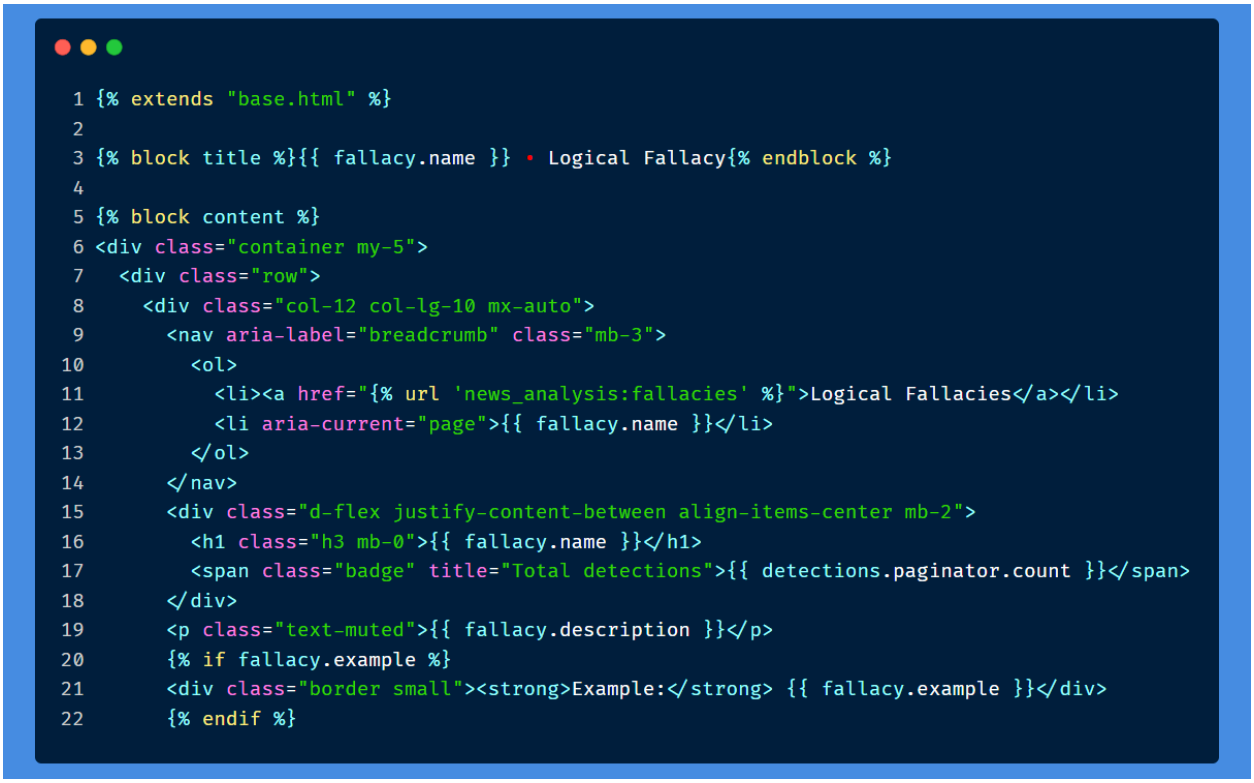
View: The View is responsible for handling user requests, interacting with the Model to retrieve or manipulate data, and returning an HTTP response. In Django, the View is implemented as a Python function or class-based view that receives an HTTP request and returns an HTTP response.

```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 5: 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 6: 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.

3. 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.

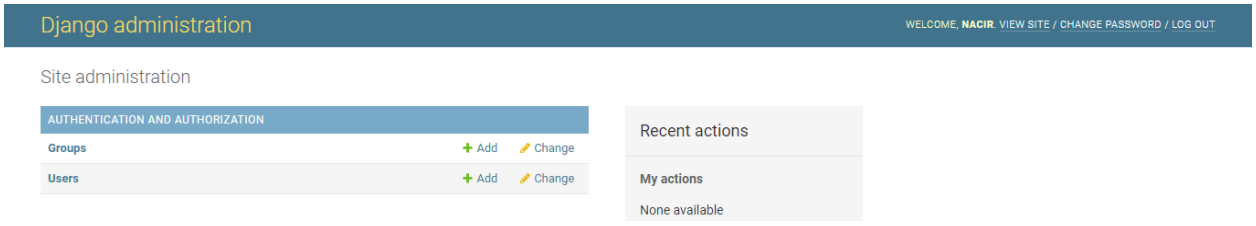


Figure 7: Main Dashboard

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.

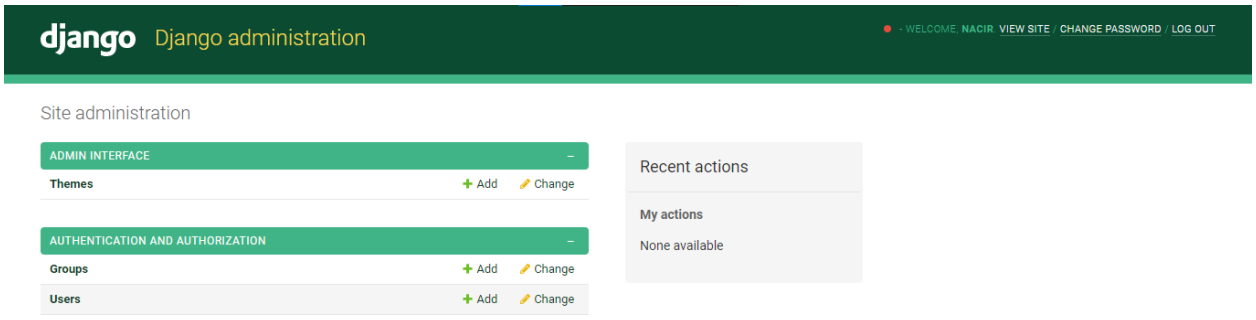


Figure 8: Customized Dashboard 1

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

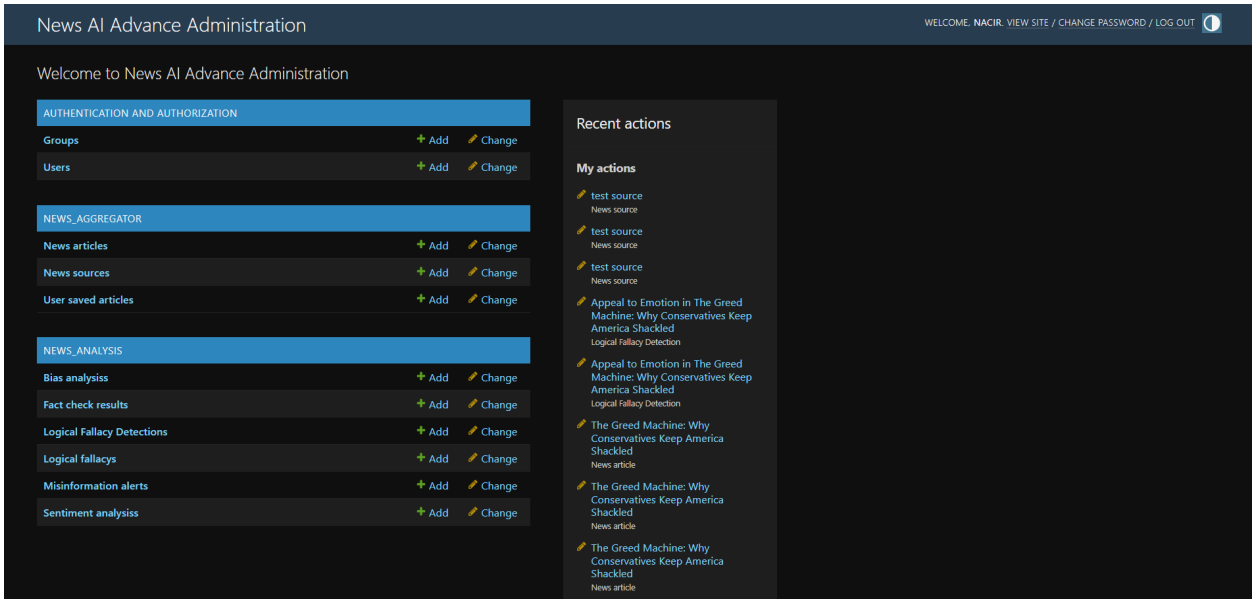


Figure 9: Customized Dashboard 2

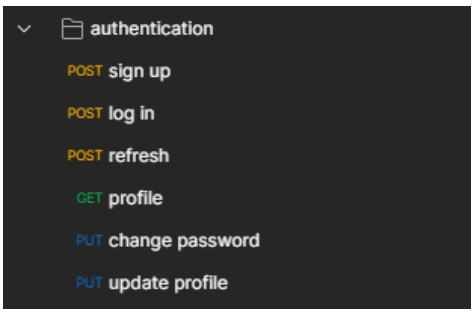
Another example of a customized dashboard, with a different color scheme and typography, making it more legible and easier to use.

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. 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 10: 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:

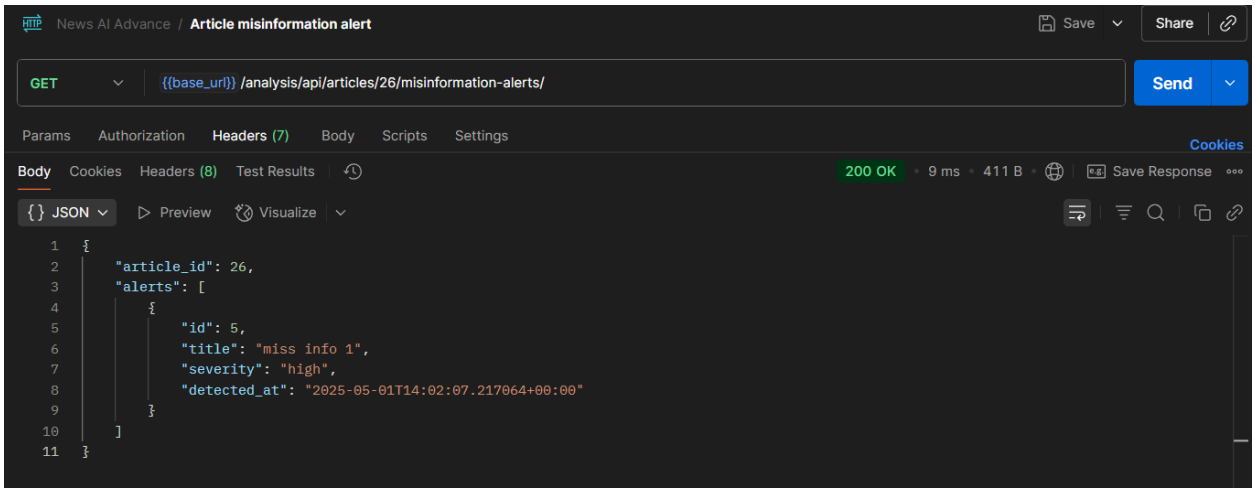


Figure 11: Misinformation Alerts API

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.

5. 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.

6. 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
```

7. Fact-checking and logical fallacies

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

8. 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

9. UI/UX and user preferences

- Article page sections: AI summary, key insights, sentiment, bias, fallacies, fact checks, alerts
- User toggles: show/hide summary, insights, and fact checks; persist preference
- Accessibility: semantic headings, keyboard navigation, and minimal color dependency

10. 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

11. 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:
    - Loading spinners per panel; partial results stream in as steps complete
    - 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

12. 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

13. 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

14. 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

Conclusion

News Advance is an AI-assisted news analysis platform that reliably aggregates sources, runs sentiment and bias detection, summarizes with an ML model (and Ollama fallback), extracts key insights, detects logical fallacies, performs basic claim verification, and produces a transparency-oriented reliability score per source. The architecture is modular (apps for aggregation and analysis), the admin workflows enable practical curation, and a small JSON API opens integration paths. This system improves news literacy by pairing readable AI outputs with confidence and provenance.

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/

Table of figures

Figure 1: System Layers

Figure 2: Use Case

Figure 3: Class Diagram

Figure 4: Model

Figure 5: View

Figure 6: Template

Figure 7: Main Dashboard

Figure 8: Customized Dashboard 1

Figure 9: Customized Dashboard 2

Figure 10: APIs requests

Figure 11: Misinformation Alerts API