Section: README.md

Research Integrity Project

The **Research Integrity Project** is a web application designed to help researchers and institutions verify the integrity of scientific journals and publishers. It provides tools to detect predatory practices, analyze conflicts of interest, and ensure compliance with ethical standards.

Features

- **Predatory Journal Detection**: Analyze journals against a comprehensive database of known predatory pub
- **Conflict of Interest Analysis**: AI-powered analysis of potential conflicts of interest in research papers.
- **ISSN Verification**: Automated verification of Online and Print ISSNs.
- **User Dashboard**: Manage analysis history and reports.
- **Authentication**: Secure login and registration system.

Tech Stack

- **Backend**: Python 3.13+, FastAPI, SQLAlchemy, SQLite.
- **Frontend**: HTML5, CSS3, JavaScript (served statically by FastAPI).
- **AI/ML**: OpenAI API for text analysis and summarization.
- **Database**: SQLite (`ria.db`).

Prerequisites

- Python 3.13 or higher.
- `pip` (Python package manager).
- OpenAI API Key (for AI features).

Installation

1. **Clone the repository:**
git clone <repository-url>
cd RIA

2. **Create a virtual environment (recommended):**
python3 -m venv venv
source venv/bin/activate # On Windows: venv\Scripts\activate

3. **Install dependencies:**
pip install -r requirements.txt

Configuration

1. Create a `.env` file in the root directory (copy from `.env.example` if available, or use the template below):

Core

SECRET_KEY=your_secret_key_here

AI Services

OPENAI_API_KEY=sk-...

Social Login (Optional)

GOOGLE_CLIENT_ID=...
GOOGLE_CLIENT_SECRET=...

Running the Application

1. **Start the server:**
uvicorn backend.main:app --reload

2. **Access the application:**
Open your browser and navigate to `http://127.0.0.1:8000`.

Project Structure

RIA/
??? backend/ # Python Backend
? ??? api/ # API Endpoints (Routes)
? ??? core/ # Configuration & Security
? ??? database/ # Database Models & Session
? ??? engine/ # Core Logic (Scorer, LLM, etc.)
? ??? schemas/ # Pydantic Models (Data Validation)
? ??? scripts/ # Utility Scripts (Scraping, Seeding)
? ??? main.py # Application Entry Point
??? frontend/ # Static Frontend Assets
? ??? css/ # Stylesheets
? ??? js/ # JavaScript Logic
? ??? *.html # HTML Pages
??? RESOURCES/ # Data Resources (Excel lists, etc.)
??? ria.db # SQLite Database
??? requirements.txt # Python Dependencies

Database & Scripts

- **Database Initialization**: The database tables are automatically created when the application starts.
- **Data Ingestion**:
- The system uses `backend/scripts/scrape_issn.py` to generate the initial database of journals and publishers
- To run the scraper:

python3 backend/scripts/scrape_issn.py

Section: DEVELOPER_GUIDE.md

Developer Guide

This document provides a deep dive into the technical architecture, core processes, and data models of the Research Integrity Project.

1. System Architecture

The application is built as a monolithic web application with a clear separation between the frontend (static assets) and backend (API).

[Mermaid Diagram Omitted in PDF]
graph TD
Client[Browser Client]
LB[FastAPI Server]
DB[(SQLite Database)]
LLM[OpenAI API]

Client -- HTTP/JSON --> LB
LB -- SQL --> DB
LB -- API Calls --> LLM

subgraph Backend
LB --> Auth[Auth Module]
LB --> Engine[Analysis Engine]
Engine --> Scorer[COI Scorer]
Engine --> Extractor[Text Extractor]
Engine --> Detector[Predatory Detector]
end

Tech Stack Details

- **Framework**: FastAPI (Python 3.13+) for high-performance async API.
- **Database**: SQLite with SQLAlchemy ORM.
- **AI Integration**: OpenAI GPT-4o via `openai` python client.
- **Frontend**: Vanilla HTML/JS/CSS served statically by FastAPI.

2. Core Processes

2.1. Analysis Pipeline

The core function of the app is analyzing PDF research papers. The pipeline flow is as follows:

1. **Upload**: User uploads a PDF via `/api/analyze/pdf`.
2. **Extraction**: `backend.engine.pdf_processor.extract_text_from_pdf` converts PDF to raw text.
3. **Scoring**: `backend.engine.scorer.COIScorer` computes a risk score (0-100) based on 5 dimensions.

4. **Summarization**: `backend.engine.llm_wrapper.LLMWrapper` sends the score and evidence to OpenAI to generate an executive summary.
5. **Persistence**: The result (score, risk level, summary, full JSON) is saved to the `analyses` table.

2.2. Scoring Logic (The 5 Dimensions)

The `COIScorer` evaluates the paper across 5 specific dimensions. Each dimension contributes to the final score.

Dimension	Weight	Description	Rules
D1: Transparency	20%	Checks for presence of Funding and COI sections.	+100 if COI missing, +50 if Funding missing.
D2: Funding Alignment	20%	Checks for commercial keywords in funding sources.	+80 if "pharma", "inc", etc. found in funding.
D3: Network	20%	Checks author affiliations for commercial entities.	+60 if commercial affiliation detected.
D4: Journal Integrity	20%	Checks against the Predatory Database.	**CRITICAL**: If predatory, score is forced to 100 (High Risk).
D5: Bias	20%	Checks for promotional/sensationalist language.	+40 if words like "miracle", "perfect" are found.

Risk Levels:
- **Low**: 0 33
- **Medium**: 34 66
- **High**: 67 100

2.3. Predatory Journal Detection

Located in `backend.engine.predatory_detector`.
- **Input**: Metadata extracted from the paper (Journal Name, Publisher).
- **Process**: Queries the `predatory_journals` table.
- **Logic**: Exact match or fuzzy match (if implemented) against known predatory entities.

3. Data Model

The database schema is defined in `backend.database.models`.

[Mermaid Diagram Omitted in PDF]
erDiagram
User ||--o{ Analysis : "has many"
User {
int id PK
string email
string hashed_password
}
Analysis {
int id PK
int user_id FK
string filename

string overall_risk
int score
text summary
json full_result
}
PredatoryJournal {
int id PK
string name
string issn
string publisher
string url
}

4. Directory Structure & Key Files

- `backend/engine/scorer.py`: **Core Logic**. Contains the `COIScorer` class and the 5 dimension scoring fu
- `backend/engine/llm_wrapper.py`: **AI Interface**. Handles the prompt engineering and API calls to OpenA
- `backend/api/api_v1/endpoints/analysis.py`: **Controller**. Orchestrates the upload -> process -> save flow.
- `backend/scripts/scrape_issn.py`: **Data Ingestion**. Script to scrape and populate the `predatory_journals`

5. Setup & Development

Environment Variables

Ensure `.env` is configured:
OPENAI_API_KEY=sk-...
SECRET_KEY=...

Database Management

- **Initialization**: Tables are created automatically on startup in `main.py`.
- **Seeding**: Use `python backend/scripts/scrape_issn.py` to populate the predatory journal list.

Running Tests

Run `pytest` in the root directory. Key tests:
- `test_llm.py`: Verifies OpenAI connectivity.
- `test_register.py`: Verifies user registration flow.