DeepKlarity Technologies - AI Wiki Quiz Generator

Objective

Build a **Frontend UI** and an **API in Python (FastAPI/Django)** that accepts a Wikipedia article URL as input and automatically generates a quiz based on the article content using a Large Language Model (LLM).

The system should have two main tabs:

TAB 1 - GENERATE QUIZ

1. **User Input:** User provides a Wikipedia article URL (e.g., `https://en.wikipedia.org/wiki/Alan_Turing`).
2. **Backend Processing:**
   - Scrape the page content using a library such as **BeautifulSoup**.
   - Send the extracted text to an LLM (**Gemini free tier API** or any other free tier API via **LangChain**) to generate a quiz (5-10 questions).
   - The generated quiz output must contain:
     - Question text
     - Four options (A-D)
     - Correct answer
     - Short explanation
     - Difficulty level (easy, medium, hard)
     - Suggested related Wikipedia topics for further reading.
3. **Data Storage:** Store all scraped and generated data in a **MYSQL or POSTGRESQL** database.

4. **API Response:** The API should return **JSON** containing the extracted and generated information.
5. **Frontend Display:** The frontend should display this information neatly in a structured, card-based layout.

TAB 2 - PAST QUIZZES (HISTORY)

1. Display a table listing all previously processed Wikipedia URLs stored in the database.
2. Clicking "**Details**" should open a modal displaying the full quiz in the same structured layout as Tab 1.

Frontend Requirements

- Clean, minimal UI (React, Vue, or simple HTML acceptable).
- **Tab 1:** URL input field, "Generate Quiz" button, Structured display of quiz and related topics.
- **Tab 2:** Table of historical quizzes, "Details" modal (reused from Tab 1).
- **Optional:** Implement a "**Take Quiz**" mode (answers hidden until submitted).

Technical Requirements

- **Backend:** FastAPI / Django
- **Database: MYSQL or POSTGRESQL**
- **Frontend:** React or minimal HTML
- **LLM:** Gemini or any free tier API via LangChain
- **Scraping:** BeautifulSoup
- **Data Source:** Wikipedia article URLs (HTML scraping only, **no Wikipedia API**).

NOTE: The use of Node.js for the backend or any core API functionality is strictly prohibited and will result in rejection. The backend must be implemented using Python (FastAPI/Django) as specified in the Technical Requirements.

Sample API Output Structure

```
{
  "id": 1,
  "url": "[https://en.wikipedia.org/wiki/Alan_Turing](https://en.wikipedia.org/wi
ki/Alan_Turing)",
  "title": "Alan Turing",
  "summary": "Alan Turing was a British mathematician and computer scientis
t...",
  "key_entities": {
    "people": ["Alan Turing", "Alonzo Church"],
    "organizations": ["University of Cambridge", "Bletchley Park"],
    "locations": ["United Kingdom"]
  },
  "sections": ["Early life", "World War II", "Legacy"],
  "quiz": [
    {
      "question": "Where did Alan Turing study?",
      "options": [
        "Harvard University",
        "Cambridge University",
        "Oxford University",
        "Princeton University"
      ],
      "answer": "Cambridge University",
      "difficulty": "easy",
      "explanation": "Mentioned in the 'Early life' section."
    },
    {
      "question": "What was Alan Turing's main contribution during World War I
I?",
      "options": [
        "Atomic research",
        "Breaking the Enigma code",
        "Inventing radar",
      "Developing jet engines"
     ],
     "answer": "Breaking the Enigma code",
     "difficulty": "medium",
     "explanation": "Detailed in the 'World War II' section."
    }
  ],
  "related_topics": ["Cryptography", "Enigma machine", "Computer science his
tory"]
}
```

Submission Requirements

1. Complete working code for backend and frontend.
2. Screenshots of: Quiz generation page (Tab 1), History view (Tab 2), and Details modal.
3. `sample_data/` folder containing: Example Wikipedia URLs tested and Corresponding JSON API outputs.
4. `README` file explaining setup, endpoints, and testing steps.
5. Include the **LangChain prompt templates** used for quiz and related-topic generation.

Evaluation Criteria

Category	Description
Prompt Design & Optimization	Effectiveness and clarity of prompts used for quiz generation, grounding of outputs in article content, and minimization of hallucination.
Quiz Quality	Relevance, diversity, factual correctness, and appropriate difficulty levels of generated questions.
Extraction Quality	Clean scraping and accurate extraction of key sections, summary, and entities.

Category	Description
Functionality	End-to-end flow: accepts URL, scrapes, generates quiz, and stores in database.
Code Quality	Modular, readable, and logically structured code with meaningful comments.
Error Handling	Handles invalid URLs, network errors, or missing sections gracefully.
UI Design	Clear, minimal, and visually organized layout; both tabs functional.
Database Accuracy	Data is correctly stored and retrievable in history view.
Testing Evidence	Sample data and screenshots demonstrate system robustness and variety.

Bonus Points

- "Take Quiz" mode with user scoring.
- URL validation and preview (auto-fetch article title before processing).
- Store scraped raw HTML in database for reference.
- Caching to prevent duplicate scraping of the same URL.
- Section-wise question grouping in UI.