Goneutral Recruitment assignment

Objective

Build a robust full-stack web application that allows users to query the YouTube API, store video details, and display the results in a dynamic and interactive React frontend.

Core Functionalities

1. **YouTube Querying:**
    - Fetch video data from YouTube based on user-defined keywords.
2. **Asynchronous Data Storage:**
    - Store video details (URL, Title, Likes, Published Date, etc.) in a database asynchronously.
3. **API for Data Retrieval & Filtering:**
    - Provide a RESTful API to fetch, filter, and paginate stored video records.
4. **Interactive React Frontend:**
    - Display video data in a user-friendly table with pagination and dynamic filtering options.
5. **Deployment (Bonus):**
    - Containerize and deploy the application to a cloud platform.

Tech Stack

- **Backend:**
    - Django
    - Django REST Framework (DRF)
    - Celery (for asynchronous tasks)
- **Frontend:**
    - React (CRA, Next.js, or Vite)
    - UI Library: Material UI, Tailwind CSS, or similar
- **Database:**
    - PostgreSQL (Recommended) or SQLite
- **Task Queue:**
    - Celery with Redis
- **YouTube API:**
    - YouTube Data API v3
- **Containerization (Bonus):**
    - Docker, Docker Compose
- **Hosting (Bonus):**
    - AWS, Google Cloud, Azure, or platforms like Render, Heroku, or Digital Ocean

Detailed Task Breakdown

1. Backend (Django & DRF)

A. YouTube Search & Data Storage API

- **Functionality:**
  - Accepts a query string as input.
  - Fetches the top N videos (e.g., 10-20) from YouTube using the YouTube Data API v3.
  - Stores the following video details in the database:
    - Video URL
    - Title
    - Description
    - Likes/Dislikes Count
    - View Count
    - Published Date
    - Thumbnail URL
  - Utilizes Celery to handle the YouTube API request and database storage asynchronously.
- **Implementation Details:**
  - Create a Django REST Framework API endpoint to receive the query.
  - Implement a Celery task to:
    - Call the YouTube Data API.
    - Parse the API response.
    - Create or update video records in the database.
  - Handle API rate limiting and errors gracefully.

B. Filtered Video Retrieval API

- **Functionality:**
  - Provides an API endpoint to fetch stored videos based on various filters.
  - Supports the following query parameters:
    - `keyword`: Search within the video title and description.
    - `min_likes`: Filter videos with a minimum number of likes.
    - `date_range`: Filter videos published within a specific date range (start and end dates).
    - `ordering`: Sort results by relevance, likes, or published date.
- **Implementation Details:**
  - Use Django REST Framework to create a read-only API endpoint.
  - Implement filtering logic within the Django queryset using `Q` objects for complex lookups[3].
  - Use Django's built-in pagination classes to paginate the results[3].
  - Include metadata in the API response, such as `next_page`, `previous_page`, and `total_count`.

C. Pagination

- **Functionality:**
  - API should return paginated results.

    - Include metadata for pagination (e.g., `next_page`, `prev_page`, `total_count`).
- **Implementation:**
    - Utilize Django REST Framework's built-in pagination classes for consistent pagination.
    - Customize pagination settings as needed (e.g., page size).

2. Frontend (React)

A. Video Table View

- **Functionality:**
    - Fetches data from the backend API and displays it in a tabular format.
    - Displays the following columns:
        - Title (linked to the YouTube video)
        - Likes
        - Published Date
        - Description (truncated with a "Read More" option)
        - Thumbnail
    - Implements pagination controls (Previous, Next buttons).
- **Implementation Details:**
    - Use a React table library (e.g., `react-table`, `MUI Data Grid`, or similar).
    - Fetch data from the API using `axios`, `fetch`, or a library like `React Query`[5].
    - Implement pagination logic to update the table based on user navigation.

B. Dynamic Querying Interface

- **Functionality:**
    - Provides a user interface for filtering and searching video data.
    - Includes the following filter components:
        - Search bar for querying video titles and descriptions.
        - Slider or input fields for specifying a range of likes.
        - Date picker for selecting a date range.
        - "Apply Filters" button to trigger the API request with the selected filters.
- **Implementation Details:**
    - Use React state management (e.g., `useState`, `useReducer`, or a library like `Redux` or `Context API`) to manage filter values.
    - Implement debouncing to limit the number of API requests triggered by user input[3].
    - Construct the API request URL dynamically based on the selected filter values.

C. UI Enhancements (Optional but Encouraged)

- Dark mode toggle
- Loading states and error handling
- Responsive design for different screen sizes

- Animations and transitions for a smoother user experience

Bonus Points

- □ **Dockerization:**
  - Provide a `docker-compose.yml` file to run the application with all its dependencies (PostgreSQL, Redis, Celery, etc.)[4].
- □ **Hosting & Deployment:**
  - Deploy the API and frontend to a cloud provider.
  - Provide a live demo link and repository link.
- □ **Code Quality:**
  - Use TypeScript for the frontend.
  - Implement ESLint and Prettier for code formatting and linting.
  - Write well-structured Django models and serializers.
- □ **Testing:**
  - Implement unit and integration tests for both the frontend and backend.
- □ **API Documentation:**
  - Generate API documentation using a tool like Swagger or OpenAPI.
- □ **Use of GenAI Tools:**
  - Document any AI tools used, along with the prompts.

Submission Requirements

- **GitHub Repository:**
  - Include a clear README file with setup instructions, dependencies, and API endpoints.
- **Demo:**
  - Provide a short demo video (2-5 minutes) showcasing the application's functionality. Alternatively, provide a deployed link to a live demo.
- **AI Prompts (if applicable):**
  - Include the exact prompts used for any AI-generated code or documentation.

This improved outline provides a more comprehensive and detailed plan for the recruitment assignment, incorporating best practices for full-stack development with React and Django. Remember to emphasize clean code, clear documentation, and a focus on delivering a functional and user-friendly application.