CODEMONK ASSIGNMENT

Introduction:

The Paragraphs API is a RESTful API designed to facilitate the storage of multiple paragraphs of text and provide functionality for searching specific words within these paragraphs. It offers a seamless way to manage and retrieve textual data efficiently.

Custom User Model:

The API employs a custom user model to handle user authentication and authorization. The model includes the following attributes:

- **id:** A unique identifier for each user, serving as the primary key.
- **name:** The user's name.
- **email:** The user's email address, which is also used as a unique identifier.
- **dob:** The user's date of birth.
- **createdAt:**The timestamp indicating when the user account was created.
- **modifiedAt:** The timestamp indicates when the user account was last modified.

Functionality:

Storing Paragraphs:

- ☐ **Tokenization:** The API tokenizes words by splitting the text at whitespace, resulting in a list of individual words.
- ☐ **Lowercasing:** All words are converted to lowercase to ensure case-insensitive search functionality.
- ☐ **Database Storage:** Each paragraph of text is stored in the PostgreSQL database along with mappings of words to paragraphs.
- ☐ **Unique ID Generation:** A unique identifier is generated for every paragraph that is indexed in the database.
- ☐ **Paragraph Definition:** A paragraph is defined by the presence of two newline characters ("\n\n").

Searching for a Word:

- ➢ **Input Processing:** Given a word to search for, the API tokenizes the search word in a similar manner as during paragraph storage.

- ➢**Word Indexing:**The search word is indexed against the paragraphs in which it is found, facilitating efficient retrieval.
- ➢**Top 10 Results:** The API returns the top 10 paragraphs containing the search word along with their respective word counts.
- ➢**Search Results:**The paragraphs are returned in descending order of word count, providing the most relevant results first.

Tech Stack:

- ➔**Framework:**The API is built using the Django framework, providing a robust foundation for web development.
- ➔**Database:** PostgreSQL is utilised as the backend database, offering reliability and scalability.
- ➔**Authentication:** Token-based authentication is implemented for user authentication and authorization.

API Endpoints:

Create Paragraphs:
URL: /api/paragraphs/
Method: POST

Request Body: Accepts a JSON object containing an array of paragraphs.

Search for Word:
URL: /api/wordsearch/
Method: GET
Query Parameter: `word` (Word to search for)

Implementation Details:

- **Models:** The API includes the following models:
- **CustomUser:** Represents the custom user model.
- **Paragraph:** Represents a paragraph of text.
- **WordIndex:** Represents the mapping of words to paragraphs.
- **Views:** Views are implemented to handle API endpoints, including paragraph creation and word search.
- **URLs:** URL patterns are defined to map requests to the appropriate views.

Usage:

- Storing Paragraphs:
  Send a POST request to `/api/paragraphs/` with an array of paragraphs in the request body.

- Search for Word:
  Send a GET request to `/api/wordsearch/` with the `word` query parameter containing the word to search for.

Conclusion:

The Paragraphs API provides a comprehensive solution for managing textual data and performing word searches efficiently. It offers a user-friendly interface for storing paragraphs and retrieving relevant information based on specific search criteria. By following the detailed documentation provided, users can seamlessly integrate and utilise the API in their applications.