Step 1: Building the Frontend for Image Search and Retrieval

Overview

This section provides a detailed account of the development process behind the frontend user interface (UI) designed for the image search and retrieval system. The frontend empowers users to seamlessly upload images (restricted to a maximum of 8 images at a time due to the rate limiting factor of OpenAI) and input search queries, facilitating the retrieval of pertinent images from the database.

Technologies Used

- Framework: Next.js
- UI Library: Shadcn UI
- CSS Framework: Tailwind CSS

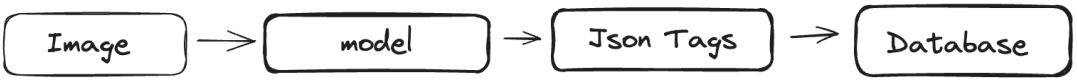
Challenges Encountered

Surprisingly, no significant challenges were encountered during the frontend development phase. Despite the intricacies of the task and the constraints of time, the process unfolded smoothly, allowing for efficient progress.

Future Improvements

- Moving forward, implementing tools such as eslint, husky, and prettier stands as a priority to enhance code quality and ensure consistency across the project. These improvements will further elevate the user experience and streamline the development process.
- Another improvement would be to minimize the size of the images during the upload process from the frontend.

Core Logic (Backend)



Image

- The initial step involves receiving images from the frontend. Upon receiving the images, they are stored in AWS S3, and public image URLs are generated for further use.
- One of the main challenges encountered was determining the process for storing images in AWS S3 and obtaining public URLs that could be accessed from anywhere.

Model

The most significant challenge encountered in this project was determining the appropriate models to utilize. The objective was to pass the image to the model, which would then detect and extract various entities present within the image. The extracted data would then be structured into a specific format, facilitating database indexing and enabling efficient search and retrieval mechanisms.

For example:

```json
{
 "subjects": ["dog"],
 "attributes": ["black", "running"],
 "themes": ["adventure"],
 "contexts": ["outdoors", "park"],
 "description" :"In detail description of the image"
}
```

Initially, attempts were made using pre-trained models from TensorFlow and OpenAI's CLIP. However, the setup process for these models proved to be time-consuming and challenging. As a result, an alternative method was explored.

An API utilizing a pre-trained model was employed to achieve the desired outcome. Eventually, the GPT-4 Vision API was selected for its superior performance compared to other available APIs. The image URL was passed into the API along with a specific prompt, resulting in the extraction of data in the desired format.

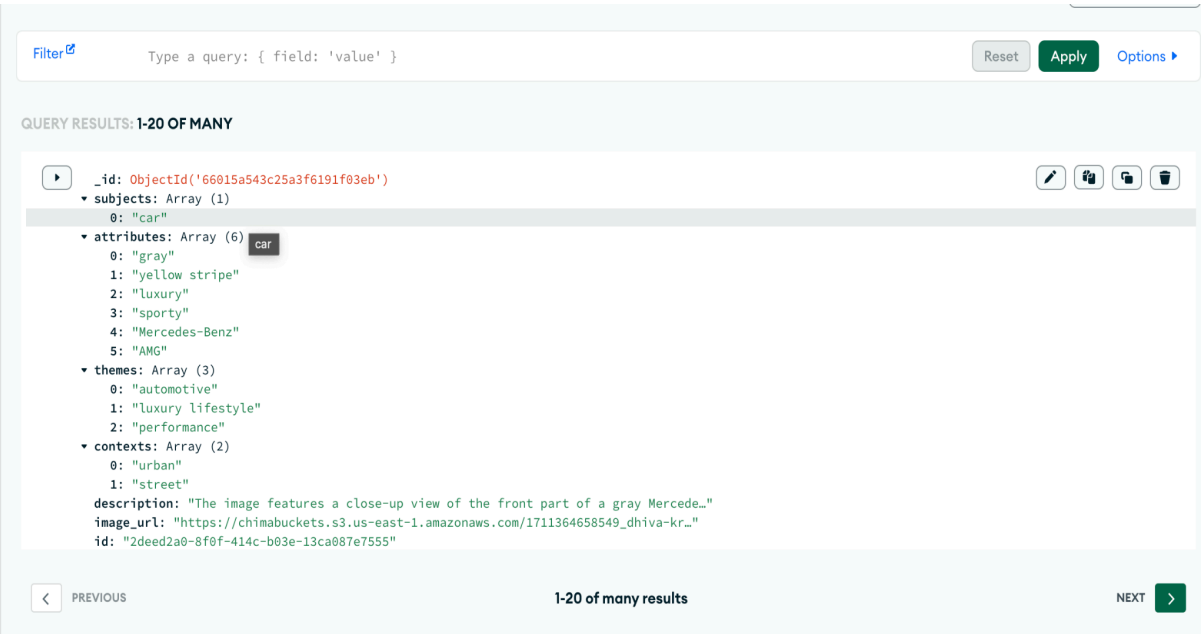
PROMPT = `
Analyzing image data to identify subjects, themes, contexts, and provide a detailed description of the image. Please structure the output in the following format:
{
 "subjects": ["dog"],
 "attributes": ["black", "running"],
 "themes": ["adventure"],
 "contexts": ["outdoors", "park"],
 "description" :"In detail description of the image"
}

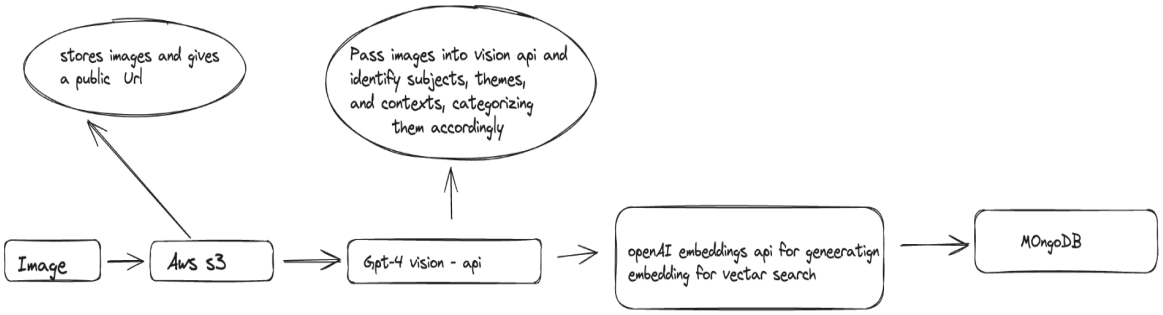
Upon receiving the data from the model, the image description was utilized to create embeddings. These embeddings were then stored in a vector database, enabling efficient retrieval search in the future.

This approach streamlined the process of extracting meaningful information from images, laying the groundwork for effective database indexing and retrieval.

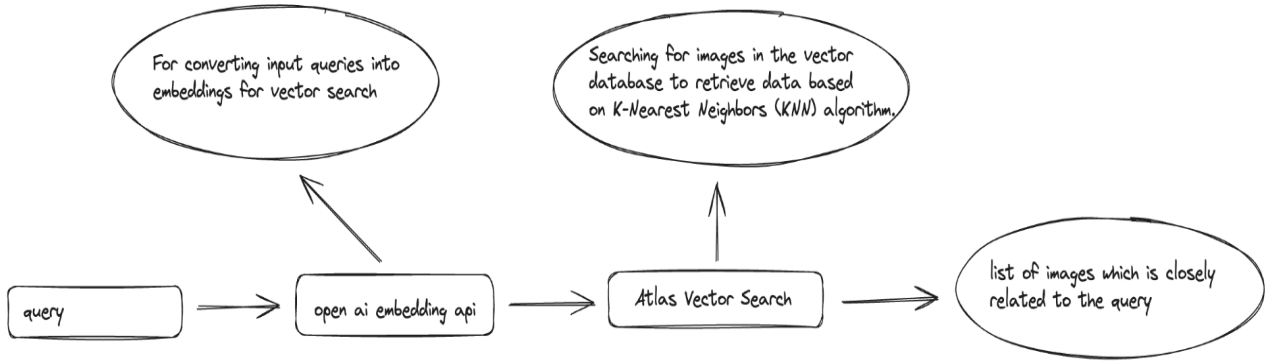
JSON Tags and Database

The image below illustrates the structure of data stored in the MongoDB database. MongoDB was chosen for its indexing capabilities and scalability, making it an ideal choice for handling large volumes of data. Additionally, MongoDB supports vector search, further enhancing its suitability for this project.





Search and Retrieval Mechanisms



Improvements:

1. Image Minification: After uploading images, implement a process to reduce and minify image sizes to conserve storage space.

2. Background Model Processing: Perform the model processing as a background task instead of within the API itself. Notify users once the upload is successful or unsuccessful.

3. Scalability: Consider setting up Docker and Kubernetes for scaling the solution in production to handle increased workload efficiently.

4. Alternative Model Integration: Explore alternative models beyond OpenAI Vision API to overcome limitations such as the number of tokens per minute.

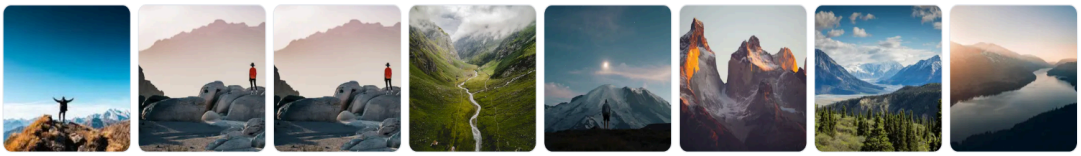
5. Model Improvement with Data: Continuously enhance the model's performance by incorporating more images, allowing it to improve over time.

Testing:

A diverse range of images has been successfully uploaded into the MongoDB database. These assets can be accessed and viewed in the "assets" folder located within the backend directory. This folder will be shared via a drive link for easy access and review.

Testcases below :

hiker climbing the mountain



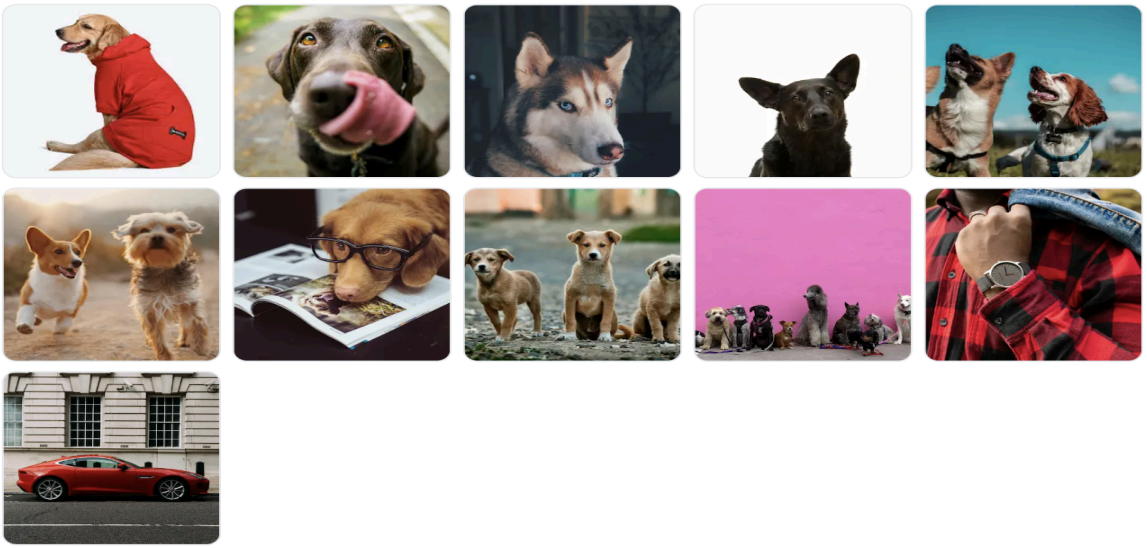
people working at office

beach

festival of light

beach

dog with red hoodie



bullets