**Image-based Search Engine System Design**

**Reinald Chenartha**

**Hardware**

Horizontal scaling would be preferred here because the number of users accessing our server APIs will be an unknown variable. Although it is safe to assume that horizontal scaling would be better here, vertical scaling could also work since most of the workload can be put into cache. It would just be a lot slower if we have to create a queue cache in the disk when receiving tons of requests. Although, having a horizontally scalable server means we would need to have a load balancer to balance the workloads of accepting requests for the multiple servers that we have. A physical load balancer would be preferred for performance’s sake, but considering the cost, using a virtualized load balancer or a software balancer would be sufficient to cut cost. For this system, a software load balancer would be enough. The type of load balancer that is chosen here is the network load balancer, which hashes data at the network level, such as IP addresses because it is more performant and also it can generate more random hashes due to having its hash components from the network layer. For security and microservice request management, we can use API gateway for an access point with security for directing clients to specific services only.

For computing visual-based machine learning results, it would be better to use NVIDIA graphic cards, since it is more compatible with most of the pretrained machine learning models in TensorFlow. It is also easier to setup and supported by TensorFlow. For a visual-based machine learning model, considering that this is going to be a search engine that will be used by more than one person at a time, it would be better if we use GPUs with above standard performance.

**Database**

Using a MongoDB-based server, we can do queries quite easily with also creating pipelines for the document projections. MongoDB can also save blobs, but I would make the whole system run unbearably slow. Instead, we can just keep the database specially for item metadata, save the image URLs that refer to the images that are uploaded on another cloud server. Another benefit from using a cloud database is it is very simple to use and also reliable. Even though we are given limitations on how flexible we can use it, it is simply a fair trade-off for the reliability. A NoSQL database was chosen here for more flexibility. Knowing how common languages such as Python and JavaScript are very suitable for JSON, handling JSON type data would be easier and having nested data (parent-child relation) would be less complicated.

Possible sample document projection from database:

[

handbag: [

{

name: “item\_name”,

image\_url: “image\_url”,

brand: “brand\_name”,

color: [“color\_name1”, “color\_name2”, …],

…

},

…

],

sneakers: { … }, …

]

We can just write every single document in a single collection in MongoDB and just retrieve data with the filter logic of the queries. But it would be a little bit quicker if we use multiple collections due to the filtering process doesn’t have to go through all the data. The problem with that is MongoDB is stricter with the created number of collections than the number of documents itself. Even though the maximum number of collections are mostly around tens of thousands, it is still more simple to query out a single collection at a time. So, for the sake of safety, simplicity, and also because all the data will have the same document schema, we are just going to use a single collection system in MongoDB.

**System capacity**

There would be three types of servers for this system. The machine learning server, the image server, and the database server. The image server can possibly be integrated with the machine learning server.

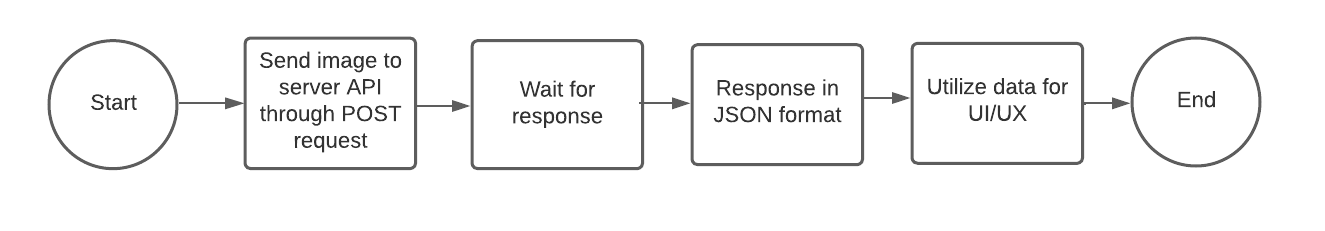
The first one would be the server or computer for the machine learning model itself, which handles communication directly to the client. The hard drive on this server won’t need to be as huge as the other ones, since it will only consist of the machine learning program. In the case of integrating this server with the raw file server (image server), we would need this server to have more hard disk space. Also, this server would need to be more performant than the other servers/cloud servers.

The MongoDB server is already very quick the way it is, since this is just for retrieving a number of countable and considerably small-sized text-based data/documents.

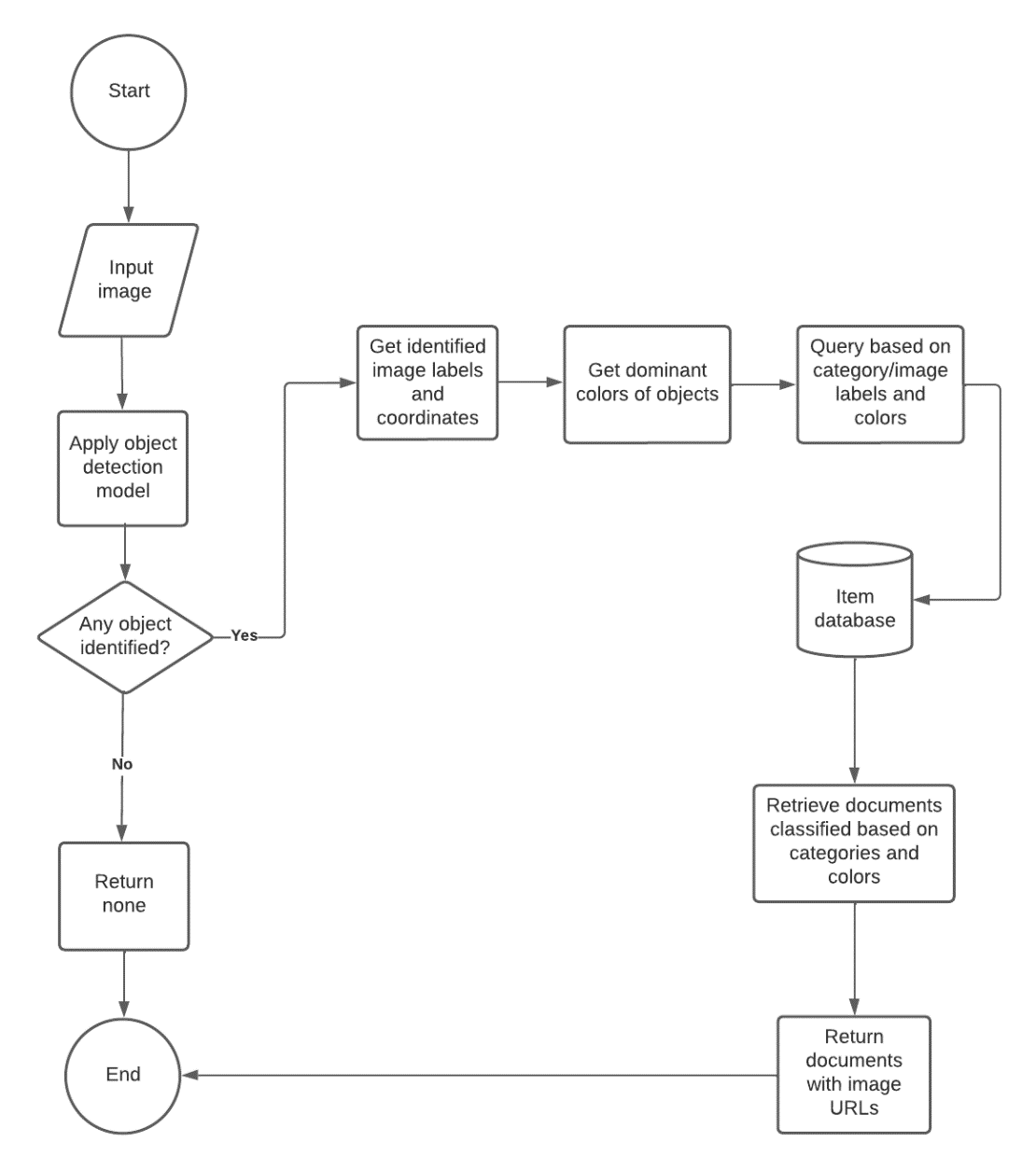
The image server, if not integrated with the machine learning server, would be best implemented also as a cloud server so that it can be practically accessed through the URL. We can implement some hosting services in which we can store our images in, such as GitHub, 000WebHost (custom PHP script), Flickr, etc.

**Algorithms and Flows**

**Front-end:**



**General back-end system:**



**For creating MongoDB projection pipeline:**

PyMongo function for getting the documents format we want in Python and JavaScript:

***DB***.aggregate(*pipeline*)

The pipeline would need to be a nested dictionary in a certain format, i.e.:

{

“$facet”: {

“handbag”: [

{

“$match”: {

“label”: “handbag”,

“color”: { $in: [“color\_name1”, “color\_name2”] }

}

},

{

“$project”: {

“name”: 1,

“color”: 1,

“label”: 1, …

}

}

],

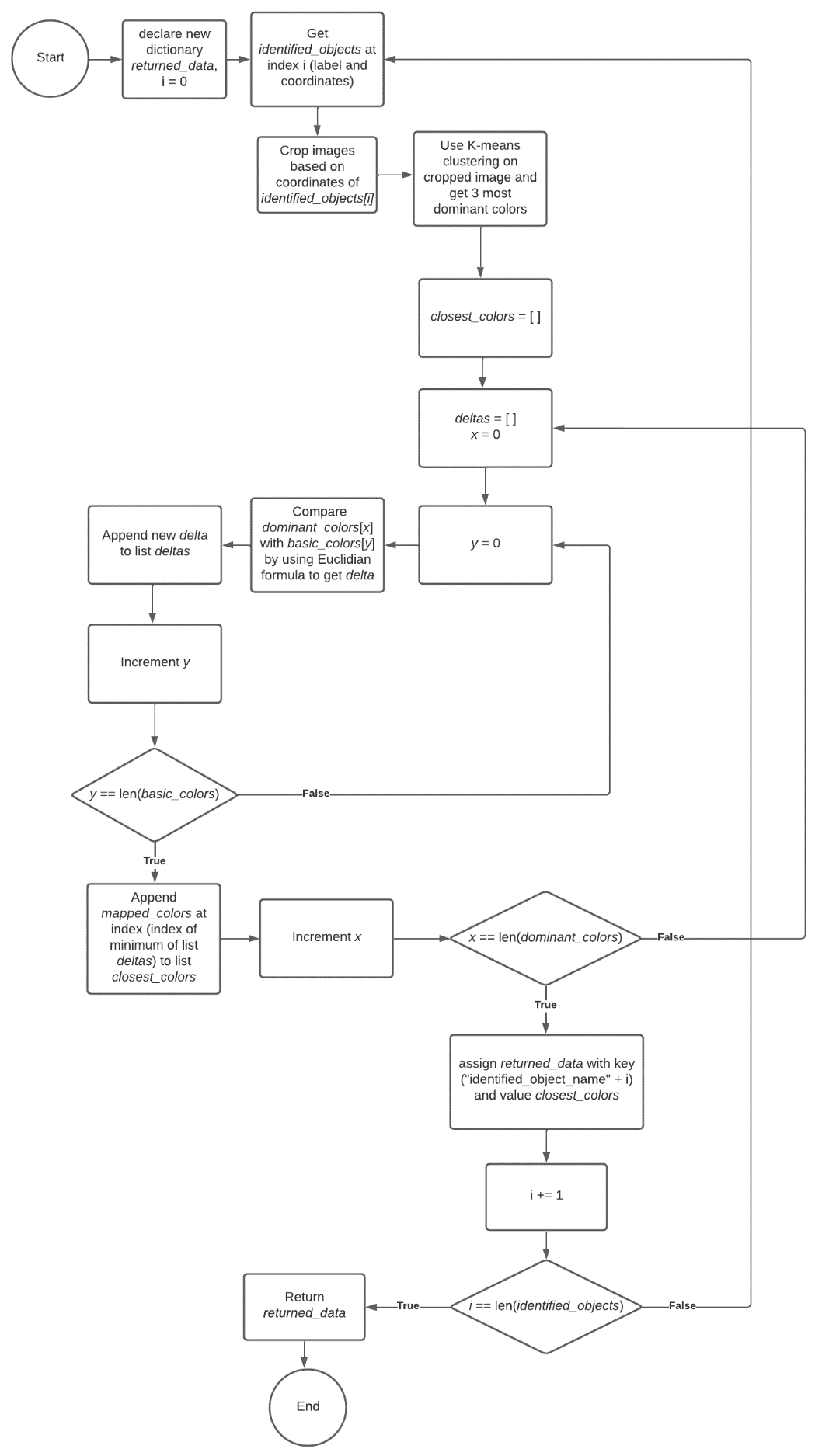
“sneakers”: [ … ], …

}

}

This pipeline can be created using a for loop and appending dictionaries of the $match and $project sets.

**Function for getting dominant colors of objects:**



Note:

1. *returned\_data* is the data that is going to be returned (in dictionary type).
2. *identified\_objects* is a list of identified objects in the image.
3. *dominant\_colors* is a list of colors that defined as dominant colors by the K-means clustering function in OpenCV.
4. *closest\_colors* is a list of basic colors closest to the dominant colors in string format, such as “red”, “green”, “blue”, etc.
5. *basic\_colors* is a list of basic colors, such as red, green, blue, yellow, etc. (in RGB format)
6. *mapped\_colors* is a list of string corresponding to the RGB values in the ­*basic­\_colors* list.
7. Euclidian formula in this case is Delta = sqrt((R2 – R1)2 + (G2 – G1)2  + (B2 – B1)2). This is to find the distance between one point to another in a 3-D space.

**Conclusion**

Scalability-wise, this system is going to use a horizontally scalable design, with a software network load balancer and API gateways. The GPU chosen here should be anything performant, as long as it is NVIDIA (for TensorFlow support).

The database server chosen is a cloud database, using NoSQL technology, MongoDB (a Serverless subscription of MongoDB Atlas should be sufficient). Aside from the MongoDB server which saves data of fashion items, such as the name, image URL, price, etc., we would also need another cloud server in which we can store our images in. The platform for the images logically should be larger than the microservice server (machine learning server) and the database server.

