**Requirements:**

Take a look at <https://mobile-tha-server.firebaseapp.com/>. The endpoint returns a list of products. Your task is to build out a REST endpoint that provides Search and Filter capability over the products info. Here are the query parameters/capabilities we expect to see:

search  
minPrice  
maxPrice  
minReviewRating  
maxReviewRating  
minReviewCount  
maxReviewCount  
inStock

The clients of this API should be able to provide any combination of these parameters and get the products matching all the criteria.

**Tools Used:**

* NodeJS
* Elastic Search
* Docker for Mac

**Architecture**

REST

API

ELASTIC SEARCH

DATABASE

SEARCH PRODUCT SERVICE

EXTERNAL API SERVICE

GET/searchProduct/?search=abc

I have opted for a micro service architecture as it is Easier to scale development and can also have performance advantages.

Lets simulate that we are going we want to search for a Product.  The following diagram shows us how is going to be the inner communication with microservices through REST. As soon as the server is up the API will fetch the data from the External API and store it in the database. Elastic Search is powerful search engine that takes care of removing the duplicate data.Hence we don’t need to worry about it. Now as the user hits the url with the paramters the Search product API will get the data from the Elastic Search database. The search only happens on the productName attribute.

If no search parameters is provided, the API will fetch and display all the data.

If the search parameters doesn’t match any valid parameter or the corresponding value is not provided to that search parameter, error will be thrown.

Every hour the API checks for the external API if it has new sets of data or if there is an update to the existing data. The Elastic search engine takes care of the duplicate data and replaces the data with the updated one in case there is an update.

The structure for the API projects will look like this:

- api/ # our apis  
- config/ # config for the app  
- mock/ # not necessary just for data examples  
- repository/ # abstraction over our db  
- server/ # server setup code  
- package.json # dependencies  
- index.js # main entrypoint of the app

-utilities.js #some required functions

**Different Approaches available:**

There are lots of ways to do search:

* Via a back-end search engine such as Elasticsearch or Solr.
* Commercial search services such as Algolia and AWS Cloudsearch.
* Using a database with built-in search such as MySQL or MongoDB.
* Application platforms and database services with add-on search functionality, such as Firebase and Cloudant.
* Client-side JavaScript search libraries such as FlexSearch

Whatever the target platforms, there are several key considerations when choosing a solution:

* **How often do you update your data?**
* **Do you need to search content, data, metadata — or all three?**
* **What are you searching, in what context?**
* **How much data do you need to search?**
* **Would you like search to work offline?**

**Why I chose Elastic Search:**

* A search engine can also serve as a data store
* Search engines can handle petabytes of data on hundreds of servers.
* Queries are extremely fast, and updates can be searchable with very low latency.
* Potentially relatively cheap and straightforward to set up and run.

**Further Enhancements**

Currently I am using a simple text search to query the database since the application doesn’t require that level of complexity. But it can easily be scaled up depending how complex we want our search to be for example currently I am using search feature in the productName but multiple columns can be easily combined and used for search.

We can have cache on our server if the data is too huge.