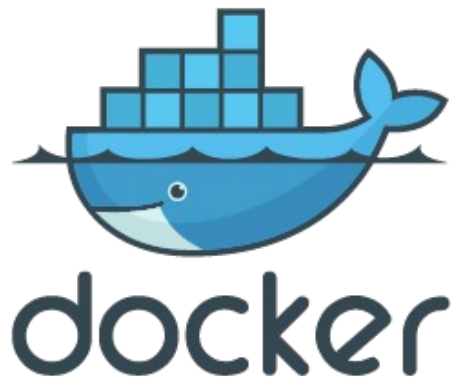


DOCKER & ELASTICSEARCH



+



ÍNDICE

Crear directorio local e instalar docker-compose.....	4
Crear estructura de Docker.....	4
Encender servicio de elasticsearch.....	5
Solucionar problemas de limitación de memoria. (OPCIONAL).....	6
Operaciones CRUD replicando una tienda online.....	6
- Crear un índice:.....	6
- Crear un producto:.....	7
- Obtener todos los productos:.....	7
- Obtener un producto por query (ID):.....	8
- Actualizar y Editar un producto:.....	8
- Eliminar un producto:.....	9
Resumen Operaciones CRUD.....	10
Pruebas con datos no reales.....	10
- Comprobar que los datos y el script se han ejecutado correctamente.....	12
- Búsqueda por familia.....	13
- Búsqueda por familia y subfamilia.....	15
- Búsqueda por más de dos filtros.....	16
_count API.....	17
Maneras de eliminar documentos.....	18
- Eliminar utilizando ID.....	19
- Eliminar todos.....	19
- Eliminar por criterio.....	19
Relacion entre SQL y Elasticsearch.....	21
Diferencias entre Elasticsearch y Bases de datos tradicionales.....	22
- Fortalezas de elasticsearch:.....	22
- Fortalezas de bbdd tradicionales:.....	23
ElasticStack.....	24
- Instalación o actualización de Java JDK.....	24
- Instalación de Elasticsearch y Kibana.....	24
- Despliegue.....	26
- Gestión de Kibana y Datos.....	30
Search-as-you-type.....	32
Proyecto Buscador.....	34
- Primer paso.....	34
- Segundo paso.....	36
- Tercer paso.....	37
Proyecto Buscador con AngularJS.....	40
- Iniciar repositorio, instalar Angular12 y generar un componente.....	40
- Modificación del contenido.....	41
Proyecto Filtros con AngularJS.....	44
- Componente de filtros.....	45
- Últimas modificaciones.....	48
Conteo de ítems.....	49
Buscador de Marcas.....	49
Listado de marcas.....	50
Bibliografía.....	52

Crear directorio local e instalar docker-compose

1. `mkdir Sites/elasticsearch`
2. `sudo apt-get install docker-compose`

Crear estructura de Docker

Posteriormente será necesario crear el archivo Dockerfile y docker-compose.yml en el directorio previamente creado y agregar el siguiente contenido:

Archivo "Dockerfile":

```
FROM elasticsearch:7.8.1
```

Archivo "docker-compose.yml":

```
version: '3.7'
services:
  elasticsearch:
    build:
      context: .
      dockerfile: Dockerfile
    container_name: elasticsearch
    restart: always
    environment:
      node.name: "es01"
      cluster.name: es-docker-cluster
      cluster.initial_master_nodes: "es01"
      bootstrap.memory_lock: "true"
      ES_JAVA_OPTS: "-Xms512m -Xmx512m"
    ulimits:
      memlock:
        soft: -1
        hard: -1
    volumes:
      - ./data01:/usr/share/elasticsearch/data
    ports:
      - 9200:9200
    networks:
      elasticsearch_network:
        aliases:
          - elasticsearch_host
```

```
volumes:
  data01: {}
networks:
  elasticsearch_network:
    name: elasticsearch_net
    driver: bridge
```

Encender servicio de elasticsearch

Para activar el servicio, se habrá que ejecutar el siguiente comando dentro del directorio previamente creado.

```
docker-compose up -d
```

Para comprobar si funciona correctamente, al entrar en <http://localhost:9200> se deberá de mostrar contenido en JSON

```
{
  "name" : "fran-ThinkCentre-M910s",
  "cluster_name" : "elasticsearch",
  "cluster_uuid" : "c0I8HtVwTHCC6i9-NlPAmg",
  "version" : {
    "number" : "7.9.1",
    "build_flavor" : "default",
    "build_type" : "tar",
    "build_hash" : "083627f112ba94dffc1232e8b42b73492789ef91",
    "build_date" : "2020-09-01T21:22:21.964974Z",
    "build_snapshot" : false,
    "lucene_version" : "8.6.2",
    "minimum_wire_compatibility_version" : "6.8.0",
    "minimum_index_compatibility_version" : "6.0.0-beta1"
  },
  "tagline" : "You Know, for Search"
}
```

Solucionar problemas de limitación de memoria. (OPCIONAL)

Este paso es opcional dependiendo la memoria virtual disponible de tu equipo. 2 Opciones:

- Ejecutar este comando para que los cambios se guarden en caché (Al reiniciar, los cambios se desharán):

```
sudo sysctl -w vm.max_map_count=262144
```

- Modificar el kernel del sistema operativo (/etc/sysctl.conf) y agregar al final del documento (Los cambios permanecerán aplicados siempre):

```
#elasticsearch  
vm.max_map_count=262144
```

Ahora, es posible que tengamos que otorgar permisos de acceso a la carpeta data01 la cual se creó a la hora de iniciar los servicios de elasticsearch con docker-compose. Para ello:

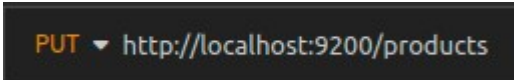
```
sudo chown tu_usuario:tu_usuario data01/ -R
```

Operaciones CRUD replicando una tienda online

Empezando a crear el API REST para agregar/indexar, editar, eliminar y buscar datos. Los siguientes ejemplos corresponden a una tienda online de ropa. Para probar estas operaciones, tendremos que instalar cualquier cliente de API REST, en este caso se esta utilizando “Insomnia”

- Crear un índice:

```
curl -X PUT http://localhost:9200/products
```

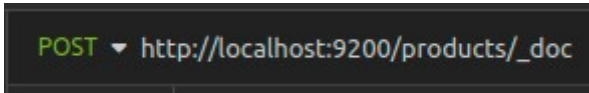
A screenshot of the Insomnia REST client interface. It shows a PUT request to the URL http://localhost:9200/products. The interface is dark-themed with a light-colored text box for the URL.

Esta secuencia cURL creará un índice para los productos. Si da el siguiente resultado significará que el índice se creó satisfactoriamente:

```
{"acknowledged":true,"shards_acknowledged":true,"index":"products"}
```

- Crear un producto:

```
curl -X POST -H 'Content-Type: application/json' -d '{ "name": "Awesome T-Shirt", "description": "This is an awesome t-shirt for casual wear.", "price": 19.99, "category": "Clothing", "brand": "Example Brand" }' http://localhost:9200/products/_doc
```



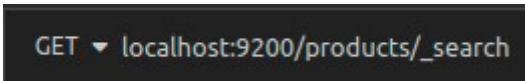
POST ▾ http://localhost:9200/products/_doc

Esta secuencia cURL creará un nuevo producto con los valores y campos proporcionados en la sentencia. Si da el siguiente resultado significará que el producto se creó satisfactoriamente:

```
{"_index":"products","_id":"ZwvjmogB0lKdOCVeL2DZ","_version":1,"result":"created","_shards":{"total":2,"successful":1,"failed":0},"_seq_no":0,"_primary_term":1}
```

- Obtener todos los productos:

```
curl --request GET \
  --url 'http://localhost:9200/products/_search?pretty=' \
  --header 'Content-Type: application/json'
```



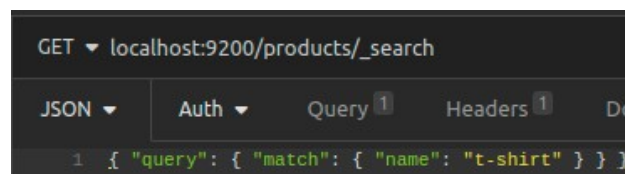
GET ▾ localhost:9200/products/_search

Esta secuencia cURL mostrará todos los productos creados. La sentencia mostrará todos los productos con sus respectivos campos y valores si se ha ejecutado con éxito.

- Obtener un producto por query (ID):

En este caso, queremos encontrar un producto que contenga un parámetro (query), es decir, queremos obtener el o los productos que contengan el nombre “t-shirt”:

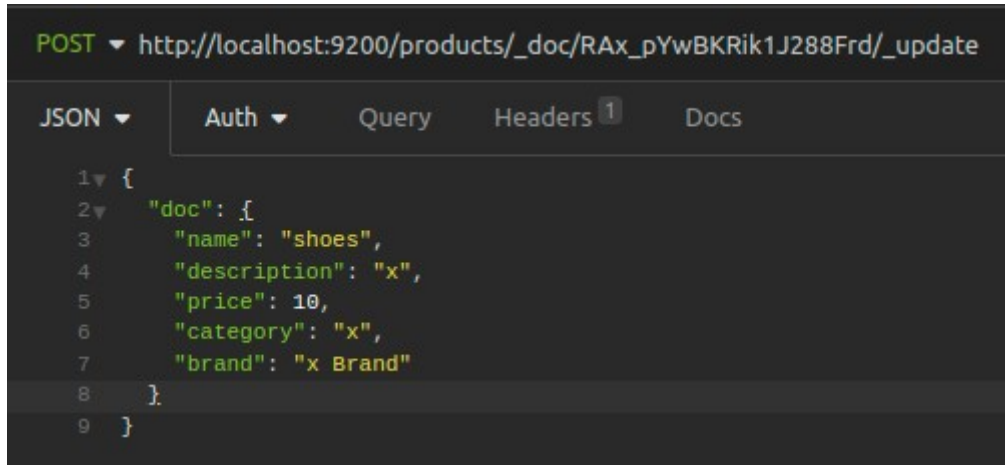
```
curl --request GET \
  --url 'http://localhost:9200/products/_search?pretty=true' \
  --header 'Content-Type: application/json' \
  --data '{ "query": { "match": { "name": "t-shirt" } } }'
```



Esta secuencia cURL mostrará todos los productos donde en el campo nombre contenga el valor “t-shirt”. La sentencia mostrará todos los productos con sus respectivos campos y valores si se ha ejecutado con éxito

- Actualizar y Editar un producto:

```
curl --request POST \
  --url
http://localhost:9200/products/_doc/RAx_pYwBKRik1J2d9
Frd/_update \
  --header 'Content-Type: application/json' \
  --data '{
    "doc": {
      "name": "shoes",
      "description": "x",
      "price": 10,
      "category": "x",
      "brand": "x Brand"
    }
  }'
```



Esta secuencia cURL actualizará el producto cuya ID fue pasada por parámetro, en este caso, “RAx_pYwBKRIk1J2d9”. La sentencia editará el producto y reemplazará los valores de los campos “name, description, price, category y brand” por los nuevos valores.

También, si se introducen nuevos campos a la hora de crear el producto, estos se crearan para ese mismo producto

El campo ID es generado automáticamente por lo cual no se podrá crear ni modificar.

- Eliminar un producto:

```
curl --request DELETE \  
  --url  
http://localhost:9200/products/_doc/Pgx8pYwBKRIk1J2d0  
FpG
```

DELETE ▼ http://localhost:9200/products/_doc/OgxOpYwBKRIk1J2dUVrp Eliminar por ID

Esta secuencia cURL eliminará el producto cuya ID fue pasada por parámetro, en este caso, “Pgx8pYwBKRIk1J2d0FpG”.


```
POST http://localhost:9200/products/_delete_by_query?conflicts=proceed

JSON
1 {
2   "query": {
3     "match_all": {}
4   }
5 }
```

Eliminar todo
(Delete ALL)

Resumen Operaciones CRUD

CREAR ÍNDICE	PUT localhost:9200/coches
INDEXAR	POST coches/_doc/
LEER UN DOCUMENTO	GET coches/{ID}
ACTUALIZAR	POST coches/_doc/{ID}/_update
ELIMINAR	DELETE coches/_doc/{ID}
_SEARCH	GET coches/_search
_COUNT	GET coches/_count

Pruebas con datos no reales

Para empezar las operaciones con datos de prueba habrá que generar estos. Para ello, ingresaremos a esta página [“Mockaroo”](#) y según nuestras necesidades, elegiremos los campos que queramos siempre y cuando estén disponibles.

En este caso, implementaremos campos que se asemejen a una tienda de piezas de coches.

Field Name	Type	Options
family	Airport Continent	blank: 0 % Σ X
subfamily	Airport Code	blank: 0 % Σ X
brand	Car Make	blank: 0 % Σ X
model	Car Model	blank: 0 % Σ X
year	Car Model Year	blank: 0 % Σ X
vin	Car VIN	blank: 0 % Σ X

+ ADD ANOTHER FIELD GENERATE FIELDS USING AI...

Rows: 1000 Format: JSON ☒ array ☒ include null values

Hint: Use "." in column names to generate nested json objects, brackets to generate arrays. [More information...](#)

Queremos que genere un fichero .JSON con 1000 líneas de información que pueda incluir valores nulos dentro de un array.

Paso opcional: Antes de ejecutar el script, podemos crear un nuevo índice para tener una organización en nuestro proyecto. Para crear un nuevo índice de coches habrá que ejecutar esta secuencia:

```
curl -X PUT http://localhost:9200/coches
```

Seguidamente haremos un script para crear por enlaces cURL, estos productos generados en el paso anterior.

```
mock_data_1000.sh
~/Sites/elasticsearch

1 #!/bin/bash
2
3 curl -X POST -H 'Content-Type: application/json' -d '{"family":"NA","subfamily":"JAV","brand":"Daewoo","model":"Lanos","year":-
1999,"vin":"WBAKES57BE142086"}' http://localhost:9200/products/_doc
4 curl -X POST -H 'Content-Type: application/json' -d '{"family":"EU","subfamily":"CSA","brand":"Dodge","model":"Stratus","year":-
1997,"vin":"SALAF2D46AA715428"}' http://localhost:9200/products/_doc
5 curl -X POST -H 'Content-Type: application/json' -d '{"family":"OC","subfamily":"TKK","brand":"Oldsmobile","model":"Bravada","year":-
2001,"vin":"1G4PP5SK2C4153560"}' http://localhost:9200/products/_doc
6 curl -X POST -H 'Content-Type: application/json' -d '{"family":"OC","subfamily":"LUU","brand":"Mitsubishi","model":"Lancer
Evolution","year":2004,"vin":"SCBFU7ZA6DC028203"}' http://localhost:9200/products/_doc
7 curl -X POST -H 'Content-Type: application/json' -d '{"family":"AF","subfamily":"HIL","brand":"Chevrolet","model":"S10 Blazer","year":-
1994,"vin":"WBAKABC57AC477971"}' http://localhost:9200/products/_doc
8 curl -X POST -H 'Content-Type: application/json' -d '{"family":"EU","subfamily":"CSA","brand":"Buick","model":"Park Avenue","year":-
1995,"vin":"1N6AF0KY4EN629872"}' http://localhost:9200/products/_doc
```

Una vez generado el script con todos los productos comenzaremos a realizar operaciones para jugar con estos datos y elasticsearch.

- Comprobar que los datos y el script se han ejecutado correctamente

El primer paso es comprobar que al ejecutar el script, los datos se han insertado correctamente. Para ello realizaremos la operación GET previamente creada. Sabremos que ha funcionado si esta nos devuelve que existen un total de 1000 coches.

```
{
  "took": 4,
  "timed_out": false,
  "_shards": {
    "total": 1,
    "successful": 1,
    "skipped": 0,
    "failed": 0
  },
  "hits": {
    "total": {
      "value": 1001,
      "relation": "eq"
    },
    "max_score": 1.0,
    "hits": [
      {
        "_index": "coches",
        "_type": "_doc",
        "_id": "wyVtz4wBnuX30NB-Et3K",
        "_score": 1.0,
        "_source": {
          "family": "EU",
          "subfamily": "LJU",
          "brand": "Infiniti",
          "model": "FX",
          "year": 2010,
          "vin": "2G4GU5GC2B9835635"
        }
      }
    ]
  }
}
```

GET ▾ localhost:9200/coches/_search

- Búsqueda por familia

Vamos a empezar a buscar por filtros o categorías.

Para ello, haremos una búsqueda normal con el método GET pero le pasaremos por el body unos requisitos en JSON.

Francisco González

localhost:9200/coches/_search

```
{ "query": { "match": { "family": "eu" } } }
```

Esta sentencia significa que busque los coches donde su categoría “family” tienen valor “eu”. De esta manera encontrará únicamente los coches que son de Europa.

GET localhost:9200/coches/_search Send 200 OK 23.3 ms 3.8 KB

JSON Auth Query 1 Headers 1 Preview Headers 2 Cookies Time

```
1 { "query": { "match": { "family": "eu" } } }
2
3
4
5
6
7
8
9
10 {
11   "took": 18,
12   "timed_out": false,
13   "_shards": {
14     "total": 1,
15     "successful": 1,
16     "skipped": 0,
17     "failed": 0
18   },
19   "hits": {
20     "total": {
21       "value": 132,
22       "relation": "eq"
23     },
24     "max_score": 2.0231707,
25     "hits": [
26       {
27         "_index": "coches",
28         "_type": "_doc",
29         "_id": "wyVtz4wBnuX30NB-Et3K",
30         "_score": 2.0231707,
31         "_source": {
32           "family": "EU",
33           "subfamily": "LJU",
34           "brand": "Infiniti",
35           "model": "FX",
36           "year": 2010,
37           "vin": "2G4GU5GC2B9835635"
38         }
39       }
40     ]
41   }
42 }
```

- Búsqueda por familia y subfamilia

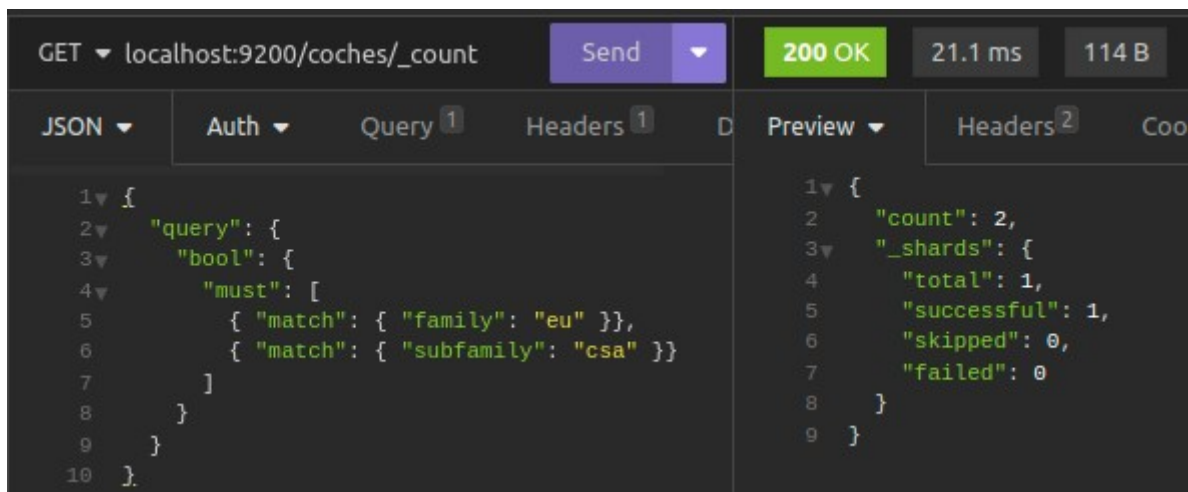
Cuando nos encontramos con 2 o más filtros, se deberá de aplicar las clausulas "MUST" o "SHOULD" según nuestras necesidades.

Resumidamente, la clausula MUST funciona como la operación lógica AND y la clausula SHOULD como la operación lógica OR. Gracias a esto podemos encontrar resultados que contengan 2 o mas filtros o encontrar resultados que contengan obligatoriamente esos 2 o mas filtros.

Para añadir estos filtros tendremos que hacer un GET al API donde se encuentren los resultados y hacer el siguiente esquema JSON para aplicarlo en el body. (Las operaciones MUST y SHOULD deberán de ser hijas de la clausula BOOL)

```
{
  "query": {
    "bool": {
      "must": [
        { "match": { "family": "eu" } },
        { "match": { "subfamily": "csa" } }
      ]
    }
  }
}
```

Si todo ha ido bien, la variable "value" nos indicará que se han encontrado tantos resultados y a parte los mostrará.



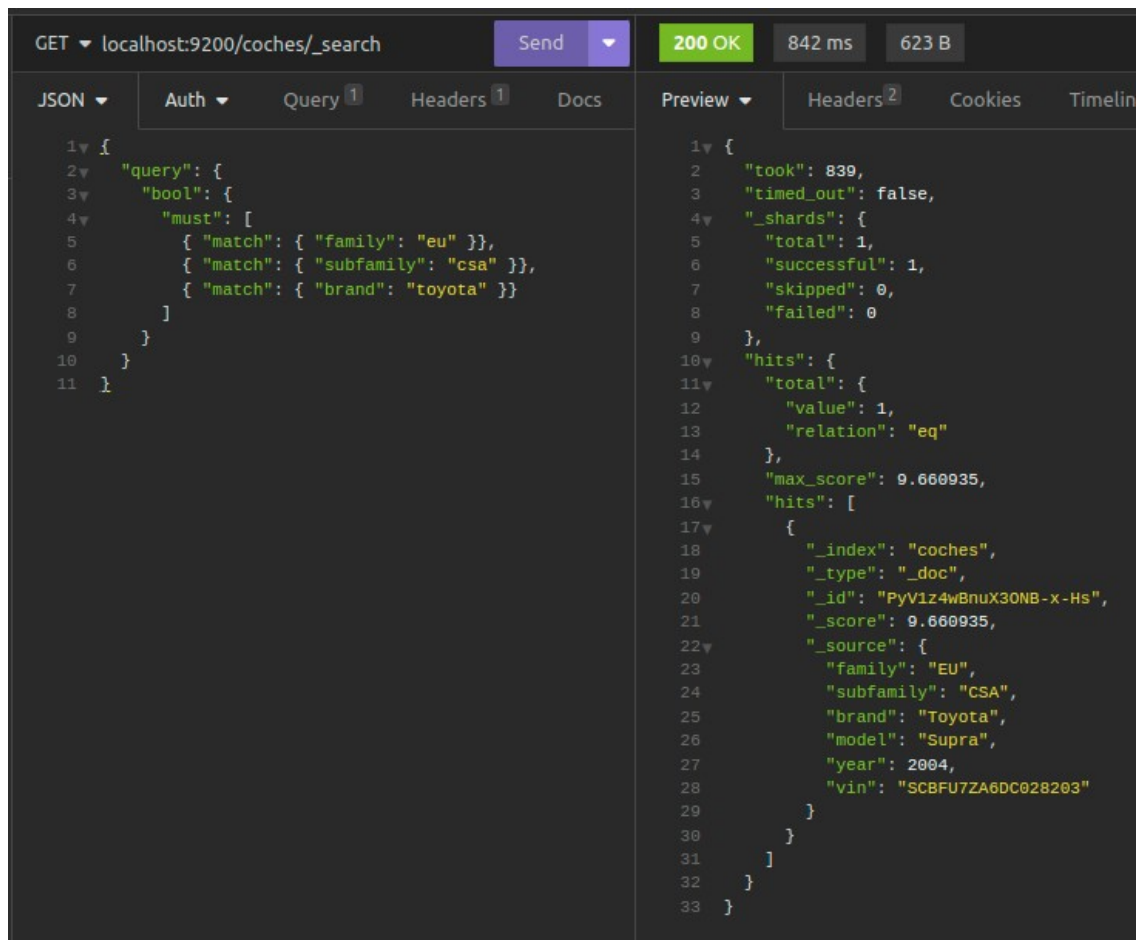
- Búsqueda por más de dos filtros

Este caso es idéntico al anterior. Para filtrar por 3 filtros usaremos MUST o SHOULD dependiendo si queremos que se cumplan los 3 o que puedan contener alguno de los 3.

En el ejemplo, se filtra por familia, subfamilia y marca:

```
{
  "query": {
    "bool": {
      "must": [
        { "match": { "family": "eu" } },
        { "match": { "subfamily": "csa" } },
        { "match": { "brand": "toyota" } }
      ]
    }
  }
}
```

Si todo ha ido bien, la variable “value” nos indicará que se han encontrado tantos resultados y a parte los mostrará.



```
GET localhost:9200/coches/_search

{
  "query": {
    "bool": {
      "must": [
        { "match": { "family": "eu" } },
        { "match": { "subfamily": "csa" } },
        { "match": { "brand": "toyota" } }
      ]
    }
  }
}
```

```
{
  "took": 839,
  "timed_out": false,
  "_shards": {
    "total": 1,
    "successful": 1,
    "skipped": 0,
    "failed": 0
  },
  "hits": {
    "total": {
      "value": 1,
      "relation": "eq"
    },
    "max_score": 9.660935,
    "hits": [
      {
        "_index": "coches",
        "_type": "_doc",
        "_id": "PyV1z4wBnuX30NB-x-Hs",
        "_score": 9.660935,
        "_source": {
          "family": "EU",
          "subfamily": "CSA",
          "brand": "Toyota",
          "model": "Supra",
          "year": 2004,
          "vin": "SCBFU7ZA6DC028203"
        }
      }
    ]
  }
}
```

_count API

Gracias a este término o API, podemos realizar un conteo de ítems o productos muy rápidamente.

Normalmente, es muy utilizado en tiendas online para contar los productos existentes dependiendo de las categorías o filtros aplicados.

AIXAM	136
ALFA ROMEO	3161
APRILIA	48
AUDI	14124
BMW	14100
CADILLAC	45
CHEVROLET	3346
CHRYSLER	3042
CITROEN	39807
CUPRA	35

Esto es un ejemplo de lo que se puede conseguir con elasticsearch. Casi al instante, se cuentan los productos existentes por categoría y se muestra el número.

Básicamente, esto se utiliza por query. Para ello, haremos un ejemplo con la petición GET de todos los productos. En el script de datos aleatorios habían 1000 líneas de información, por lo cual la sentencia debería de contar 1000 ítems.

```
localhost:9200/coches/_count
```

Para contar ítems filtrados será igual que los ejemplos anteriores. Habrá que indicar por el body, el tipo de filtro y por método GET indicar la URL del API.

```
{
  "count": 2,
  "_shards": {
    "total": 1,
    "successful": 1,
    "skipped": 0,
    "failed": 0
  }
}
```

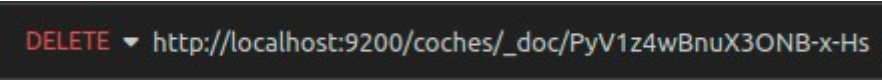
Maneras de eliminar documentos

Existen 3 formas de eliminar documentos en elasticsearch. Eliminar por ID, eliminar todos y eliminar por query o criterio.

- Eliminar utilizando ID

Para eliminar documentos por ID, deberemos especificar en la URL la ID, en este caso, del documento de coches. También se habrá de aplicar el método DELETE.

http://localhost:9200/coches/_doc/OgxOpYwBKRIk1J2dUVrp



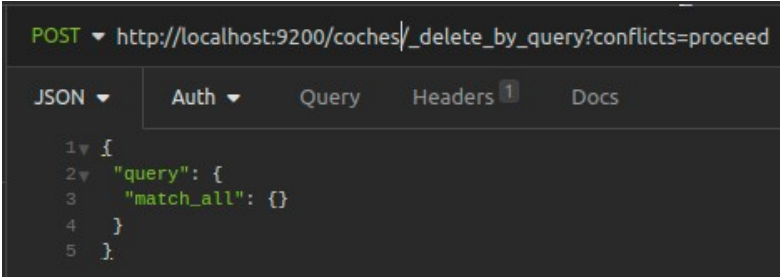
```
DELETE ▾ http://localhost:9200/coches/_doc/PyV1z4wBnuX3ONB-x-Hs
```

- Eliminar todos

Para eliminar todos los documentos, tendremos que introducir la siguiente sentencia. Además, se habrá de aplicar el método POST.

http://localhost:9200/coches/_delete_by_query?conflicts=proceed

```
{  
  "query": {  
    "match_all": {}  
  }  
}
```



```
POST ▾ http://localhost:9200/coches/_delete_by_query?conflicts=proceed  
JSON ▾ Auth ▾ Query Headers 1 Docs  
1 {  
2   "query": {  
3     "match_all": {}  
4   }  
5 }
```

El funcionamiento de esta secuencia, consiste en omitir todos los conflictos o errores y en el body habrá que indicar que se eliminen todos los datos, pasándole a la query unos filtros vacíos.

- Eliminar por criterio

Para eliminar bajo un criterio establecido, deberemos ejecutar el siguiente comando. El método tendrá que ser POST.

http://localhost:9200/coches/_delete_by_query

Francisco González

```
{
  "query": {
    "match": {
      "brand": "Volvo"
    }
  }
}
```

Esto eliminará todos los documentos donde la marca sea igual a Volvo.

The screenshot shows a REST client interface with a POST request to `http://localhost:9200/coches/_delete_by_query` and a successful 200 OK response. The request body is a JSON object with a query matching the brand "Volvo". The response body contains statistics about the deletion operation.

Request	Response
<pre>1 { 2 "query": { 3 "match": { 4 "brand": "volvo" 5 } 6 } 7 }</pre>	<pre>1 { 2 "took": 43, 3 "timed_out": false, 4 "total": 16, 5 "deleted": 16, 6 "batches": 1, 7 "version_conflicts": 0, 8 "noops": 0, 9 "retries": { 10 "bulk": 0, 11 "search": 0 12 }, 13 "throttled_millis": 0, 14 "requests_per_second": -1.0, 15 "throttled_until_millis": 0, 16 "failures": [] 17 }</pre>

Relacion entre SQL y Elasticsearch.

Por defecto, Elasticsearch no tiene uniones como en una base de datos SQL (Modelo relacional) por lo cual cada herramienta se ajusta a cada necesidad. Por ejemplo:

Cuando se necesita unir múltiples conjuntos de datos, Elasticsearch puede cargar los datos e indexarlos para permitir una consulta de alto rendimiento.

Una de las principales diferencias entre ambos, es que Elasticsearch destaca en la búsqueda y el análisis de los datos mientras que SQL se emplea para una amplia variedad de aplicaciones ya que las bases de datos son más versátiles y adecuadas.

De hecho, Elasticsearch y SQL se comunican entre si para acelerar, escalar y flexibilizar los datos y de esta manera poder hacer consultas con queries.

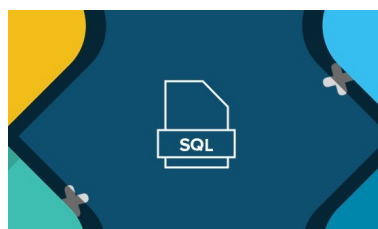
Algunas de las operaciones que podemos hacer son:

```
GET _xpack/sql?format=json
{
  "query": "DESCRIBE logstash*"
}
```

Esto devolverá el tipo de dato de cada campo de logstash mostrado en JSON.

```
GET _xpack/sql?format=txt
{
  "query": "DESCRIBE logstash*"
}
```

Igualmente si especificamos el tipo de dato como .txt



Diferencias entre Elasticsearch y Bases de datos tradicionales

Para diferenciar elasticsearch y bdd tradicionales deberemos de saber las principales características de cada uno.

- Fortalezas de elasticsearch:

- Indexación invertida → Elasticsearch utiliza índices por lo cual cada palabra es única y cada estructura de datos está optimizada para la velocidad, lo que permite búsquedas rápidas entre datos masivos.
- Procesamiento de texto avanzado → Elasticsearch tiene un sistema de tokenización, derivación y manejo de sinónimos y cuando se realiza una búsqueda, el algoritmo de relevancia, garantiza que los resultados más importantes salgan primero.
- Flexibilidad con datos → La estructura de datos nativa es en JSON por lo cual permite estructurar los datos jerárquicamente, lo que permite consultas más complejas. También se pueden detectar automáticamente los tipos de datos (Mapping).
- Ingestión de datos → Gracias a la API bulk, elastic permite múltiples operaciones en una sola solicitud. De hecho, elastic, permite manejar operaciones de indexación simultáneas en sus nodos distribuidos.
- Disponibilidad de datos instantáneos → los datos en elastic, están disponibles para las operaciones de búsqueda casi de inmediato. Esta capacidad de indexación se debe a los intervalos de actualización optimizados.

- Fortalezas de bbdd tradicionales:

- Transacciones → SQL da prioridad a transacciones ACID, ya que requieren estricta integridad y coherencia de datos.
- Relaciones complejas → SQL esta diseñado para la normalización y las relaciones de datos (Uniones complejas y modelado de datos relacionales).
- Uso de propósito general → SQL se utiliza para bases de datos más versátiles y adecuadas para una amplia variedad de aplicaciones.

ElasticStack

Para terminar, veremos como realizar una infraestructura full-stack con elasticsearch y su framework Kibana para la analítica y gestión de datos. Gracias a este kit, podremos sacar más partido de estas herramientas. Comenzaremos instalando Java y su kit de desarrollo (JDK) y posteriormente iremos desplegando los servicios y configurándolos.



- Instalación o actualización de Java JDK

El primer paso será instalar JDK. Para ello, actualizaremos nuestro sistema en busca de nuevos paquetes para posteriormente descargar Java:

1. `sudo apt update`
2. `sudo apt install default-jre`
3. `sudo apt install default-jdk`



- Instalación de Elasticsearch y Kibana

El siguiente paso será descargar [Elasticsearch](#) y [Kibana](#). Para ello, descargaremos los archivos comprimidos de estas dos herramientas y los descomprimiremos para que se nos quede una carpeta con todos los archivos dentro. Una vez hecho esto, ya podremos desplegar los

servicios pero primero cambiaremos unas configuraciones en los archivos .yml.

En la carpeta “config” dentro de elasticsearch, encontraremos los archivos elasticsearch.yml y jvm.options. En el primer archivo, encontraremos configuraciones como por ejemplo, el nombre del clúster, la ruta de datos, configuraciones de red, memoria entre otros, pero es importante añadir una línea la cual active las opciones de seguridad y nos proporcione 1 mes gratis para poder utilizar otras herramientas como LogStash o Beats:

```
xpack.security.enabled: true

# ----- Cluster
#
# Use a descriptive name for your cluster:
#
#cluster.name: my-application
xpack.security.enabled: true
```

Una vez hecho esto, podemos modificar el archivo jvm.options para ajustar un límite de la memoria virtual.

Seguidamente, configuraremos el archivo kibana.yml que se encuentra dentro de la carpeta “config”. Dentro de este archivo añadiremos estas dos líneas que están relacionadas con las credenciales de inicio de sesión:

```
elasticsearch.username: "elastic"
elasticsearch.password: "elastic"

elasticsearch.username: "elastic"
elasticsearch.password: "elastic"
```

- Despliegue

El tercer paso será desplegar los servicios. Primero, abriremos una terminal, y dentro de la carpeta de elasticsearch, ejecutaremos el siguiente comando `bin/elasticsearch-setup-passwords`. Esto, nos pedirá crear una contraseña para cada servicio.

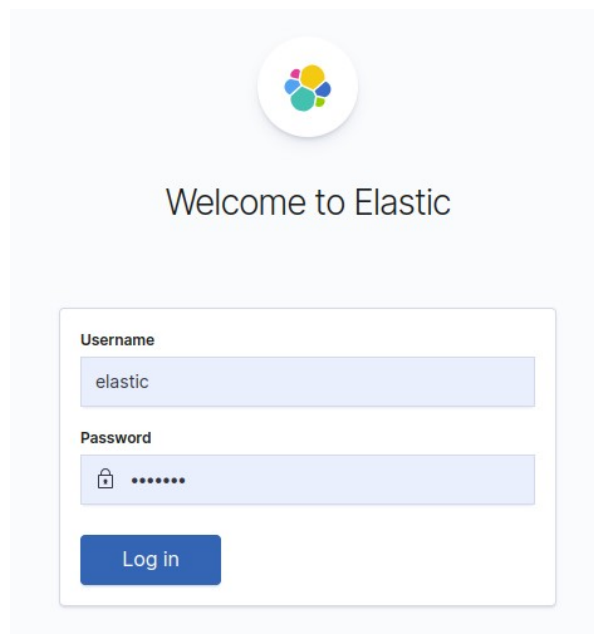
Seguidamente ejecutaremos el siguiente comando para inicializar el servicio de elasticsearch `bin/elasticsearch`

Para comprobar si se ha iniciado correctamente, ingresaremos a <http://localhost:9200/> y si se muestra contenido en JSON, significa que el servicio se ha desplegado sin ningún problema.

```
{
  "name" : "fran-ThinkCentre-M910s",
  "cluster_name" : "elasticsearch",
  "cluster_uuid" : "c0I8HtVwTHCC6i9-NlPAmg",
  "version" : {
    "number" : "7.9.1",
    "build_flavor" : "default",
    "build_type" : "tar",
    "build_hash" : "083627f112ba94dffc1232e8b42b73492789ef91",
    "build_date" : "2020-09-01T21:22:21.964974Z",
    "build_snapshot" : false,
    "lucene_version" : "8.6.2",
    "minimum_wire_compatibility_version" : "6.8.0",
    "minimum_index_compatibility_version" : "6.0.0-beta1"
  },
  "tagline" : "You Know, for Search"
}
```

Una vez abierto el servicio de elasticsearch, abriremos kibana. Para ello, entraremos a la carpeta de kibana y ejecutaremos el comando `bin/kibana`

Si ha ido todo bien, ingresaremos a <http://localhost:5601/> y se nos abrirá un panel de inicio sesión, el cual introduciremos las credenciales previamente configuradas (elastic,elastic).



- Gestión de Kibana y Datos

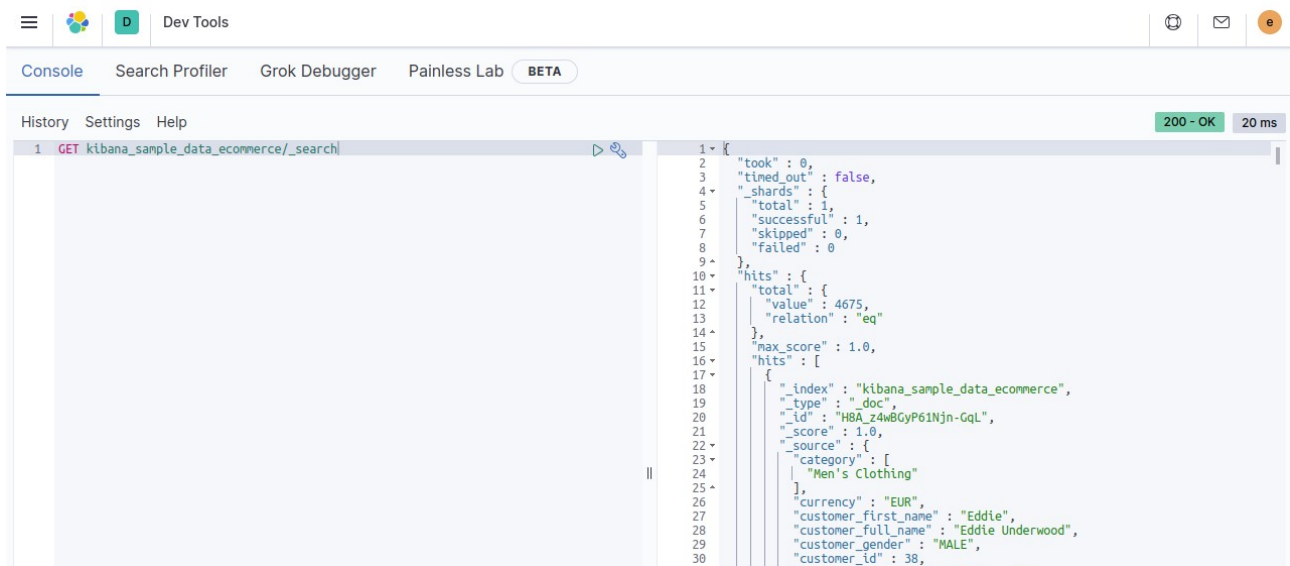
Una vez dentro de Kibana, podremos añadir datos de prueba que nos proporciona Kibana, importar datos en CSV, JSON o utilizar documentos ya creados de elasticsearch. Al ser datos de prueba podemos insertar los de ejemplo que nos proporciona Kibana ya que solo los usaremos para hacer pruebas.

Add sample data
Load a data set and a Kibana dashboard

Upload data from log file
Import a CSV, NDJSON, or log file

Use Elasticsearch data
Connect to your Elasticsearch index

Cuando ya hayamos insertado los datos, automáticamente se nos habrá creado un índice con el cual podremos hacer operaciones en la consola. Para acceder a la consola, tendremos que dirigirnos al apartado "Dev Tools" que se encuentra en el menú desplegable en la sección "Management". Aquí podremos ir jugando con los comandos que ya hemos visto durante esta memoria.



The screenshot shows the DevTools interface with the Console tab selected. A REST client request is visible in the History pane, and its response is shown in the main console area.

Request:

```
1 GET kibana_sample_data_ecommerce/_search
```

Response:

```
1 {
2   "took": 0,
3   "timed_out": false,
4   "shards": {
5     "total": 1,
6     "successful": 1,
7     "skipped": 0,
8     "failed": 0
9   },
10  "hits": {
11    "total": {
12      "value": 4675,
13      "relation": "eq"
14    },
15    "max_score": 1.0,
16    "hits": [
17      {
18        "_index": "kibana_sample_data_ecommerce",
19        "_type": "_doc",
20        "_id": "H8A_z4w8CyP61Njn-GqL",
21        "_score": 1.0,
22        "_source": {
23          "category": [
24            "Men's Clothing"
25          ],
26          "currency": "EUR",
27          "customer_first_name": "Eddie",
28          "customer_full_name": "Eddie Underwood",
29          "customer_gender": "MALE",
30          "customer_id": 38,
```

The response status is 200 - OK and the execution time is 20 ms.

Search-as-you-type

Search_as_you_type es un campo de tipo texto que está optimizado para proporcionar soporte inmediato a las consultas. Crea una serie de subcampos que se analizan para indexar términos que puedan coincidir de forma eficaz con una consulta que coincida parcialmente con todo el valor de texto indexado. Admite tanto la complexión prefija (es decir, la coincidencia de términos a partir del principio de la entrada) como la complexión infija (es decir, la coincidencia de términos en cualquier posición dentro de la entrada).

Para realizar peticiones con esta característica habrá que crear un mapping personalizado. En Insomnia, crearemos una petición PUT para crear un índice nuevo:

<http://localhost:9200/pruebacoche>

Posteriormente, agregaremos al body el siguiente código:

```
{
  "mappings": {
    "properties": {
      "my_field": {
        "type": "search_as_you_type"
      }
    }
  }
}
```

La forma más eficaz de realizar una consulta search-as-you-type suele ser una consulta `multi_match` de tipo `bool_prefix` que se dirige al campo raíz `search_as_you_type` y a sus subcampos `shingle`. Esto puede coincidir con los términos de la consulta en cualquier orden, pero puntuará los documentos más alto si contienen los términos en orden en un subcampo `shingle`.

```
{
  "query": {
    "multi_match": {
      "query": "brown f",
      "type": "bool_prefix",
      "fields": [
        "my_field",
        "my_field._2gram",
        "my_field._3gram"
      ]
    }
  }
}
```

Esto devolverá todos aquellos valores que contengan la query aplicada, independientemente de si tiene minúsculas o mayúsculas.

Normalmente es muy utilizado en páginas web con buscador de artículos ya que esta herramienta hace de texto predictivo, es decir, cada vez que se escribe una letra la página va mostrando artículos relacionados con la letra o palabras que se están escribiendo.

Proyecto Buscador

Vamos a realizar un buscador de piezas de coche con elasticsearch y tecnología search_as_you_type.

- Primer paso

Crear índice mapeado que se ajuste a nuestras necesidades a la hora de buscar artículos.

En mi caso, definí varios sinónimos que podrían ser de utilidad a la hora de buscar piezas de coches.

Por otro lado, apliqué una norma que empiece a mostrar resultados a partir del segundo carácter o dígito.

Por último, configuré el mapping para que buscase por los 4 principales campos: marca, modelo, categoria y subcategoria.

```
curl --request PUT \  
  --url http://localhost:9200/buscador \  
  --header 'Content-Type: application/json' \  
  --data '{  
    "settings": {  
      "index.max_ngram_diff": 10,  
      "analysis": {  
        "filter": {  
          "custom_synonyms": {  
            "type": "synonym",  
            "synonyms": [  
              "auto, coche, vehículo",  
              "rueda, neumático",  
              "pieza, componente",  
              "motor, propulsor",  
              "filtro, purificador",  
              "repuesto, recambio"  
            ]  
          }  
        }  
      }  
    }  
  }'
```

```
    }
  },
  "analyzer": {
    "custom_ngram_analyzer": {
      "type": "custom",
      "tokenizer": "custom_ngram_tokenizer",
      "filter": ["lowercase", "asciifolding",
"custom_synonyms"]
    }
  },
  "tokenizer": {
    "custom_ngram_tokenizer": {
      "type": "ngram",
      "min_gram": 2,
      "max_gram": 10,
      "token_chars": ["letter", "digit"]
    }
  }
},
"mappings": {
  "properties": {
    "marca": {
      "type": "text",
      "analyzer": "custom_ngram_analyzer"
    },
    "modelo": {
      "type": "text",
      "analyzer": "custom_ngram_analyzer",
      "boost": 1.5
    },
    "categoria": {
```

```
        "type": "text",
        "analyzer": "custom_ngram_analyzer"
    },
    "subcategoria": {
        "type": "text",
        "analyzer": "custom_ngram_analyzer"
    }
}
}
```

- Segundo paso

Para poder seguir necesitamos tener unos datos, en este caso usaremos 10 scripts exportados de una base de datos con 120.000 líneas cada uno. La estructura para crear un script sería esta:

`#!/bin/bash`

```
curl -X POST -H 'Content-Type: application/json' -d '{
    "id" : 1,
    "marca" : "FORD",
    "modelo" : "MONDEO BERLINA",
    "categoria" : "PRODUCTOS",
    "subcategoria" : "PLASTICO PASE RUEDA TRAS. IZQ."
}' http://localhost:9200/buscador/_doc
curl -X POST -H 'Content-Type: application/json' -d '{
    "id" : 2,
    "marca" : "FORD",
    "modelo" : "MONDEO BERLINA",
    "categoria" : "CARROCERIA TRASERA",
    "subcategoria" : "PORTON TRASERO"
}' http://localhost:9200/buscador/_doc
...
```

- Tercer paso

Programar una llamada asíncrona a la URL de elasticsearch, en este caso, http://localhost:9200/buscador/_search:

Seguidamente, implementaremos los filtros necesarios, en mi caso:

```
function search(query) {
    $.ajax({
        url:
'http://localhost:9200/delfincar_mapeo2/_search',
        type: 'POST',
        contentType: 'application/json',
        data: JSON.stringify({
            query: {
                bool: {
                    should: [
                        { match: { "marca": { query: query } } },
                        { match: { "modelo": { query:
query } } },
                        { query: query } } ],
                    { match: { "categoria":
{ query: query } } },
                    { match: { "subcategoria":
{ query: query } } }
                ]
            }
        })),
        success: function (data) {
            mostrarResultados(data.hits.hits, query)
        },
        error: function (error) {
            console.error('Error en la búsqueda:', error)
        }
    });
}
```

Después de la función que realiza la búsqueda, habrá que manejar la palabra que mete el usuario por la barra de búsqueda. Para ello:

```
$('#searchInput').on('input', function () {
    var query = $(this).val()
    if (query.length >= 2) {
        search(query)
    } else {
        mostrarResultados([])
    }
})
```



```
    }  
  })
```

Si queremos que los resultados se muestren habrá que realizar una función para mostrarlos:

```
function mostrarResultados(results, query) {  
    var resultsList = $('#results');  
    resultsList.empty();  
  
    results.slice(0, 6).forEach(function (result) {  
        var car = result._source;  
  
        var listItem = $('<li>');  
  
        // Crear un elemento de párrafo para mostrar la  
información del coche  
        var paragraph = $('<p>').text(car.marca + ' ' +  
car.modelo + ' | ' + car.categoria + ' --- ' + car.subcategoria);  
  
        // Resaltar la cadena de búsqueda en el párrafo  
        if (query) {  
            var highlightedText =  
resaltarQuery(paragraph.text(), query);  
            paragraph.html(highlightedText);  
        }  
  
        // Agregar el párrafo al elemento de lista  
        listItem.append(paragraph);  
  
        // Agregar el elemento de lista a la lista de  
resultados  
        resultsList.append(listItem);  
    });  
}
```

Por último, implementaremos dos funcionalidades que permitirán darle vida a los resultados. Una de ellas será resaltar la palabra que el usuario escribe y otra será tratar expresiones o caracteres regulares que puedan ser introducidas:

```
function resaltarQuery(text, query) {  
    // Usar una expresión regular para envolver la cadena  
    de búsqueda en una etiqueta span con un estilo específico  
    var regex = new RegExp('(' +  
    tratarExpresionesRegulares(query) + ')', 'gi');  
    return text.replace(regex, '<span style="background-  
color: yellow;">$1</span>');  
}  
  
// Función para escapar caracteres especiales en la cadena  
de búsqueda para usar en una expresión regular  
function tratarExpresionesRegulares(string) {  
    return string.replace(/[\.*+?^${}()|[\]\[\]]/g, "\\$&");  
}
```

Proyecto Buscador con AngularJS

Una vez hecho el programa en HTML y JS, intentaremos migrar a Angular para que a la hora de implementar el buscador a la página, se pueda realizar con menor dificultad.

- Iniciar repositorio, instalar Angular12 y generar un componente

Para empezar, instalaremos Angular localmente y utilizaremos npx para poder ejecutar/compilar nuestra aplicación de manera más rápida.

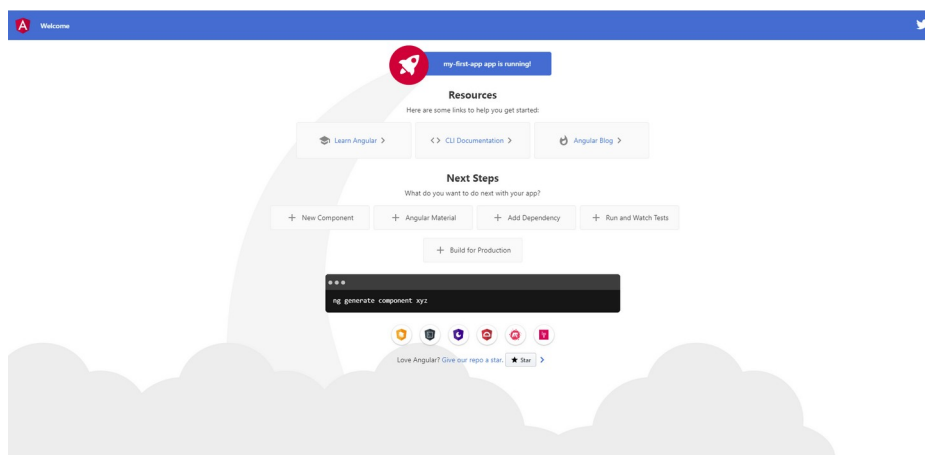
Creamos una carpeta y abrimos un terminal dentro de esta. Una vez dentro de la carpeta en el terminal, iremos ejecutando los comandos necesarios para crear un proyecto en Angular, en este caso, la versión 12.

- `npm init -y` (dentro del directorio)
- `npm install @angular/cli@12` (Para instalar angular localmente y no general (-g))

- Si todo ha ido bien, al ejecutar `npx ng version`, la terminal nos indicará que la versión de angular instalada es la 12.

- Ahora entraremos al directorio donde se encuentran todos los archivos de configuración .json y ejecutaremos la orden `npx ng generate component buscador` para crear el componente buscador, donde realizaremos todas las operaciones.

- Para finalizar, ejecutaremos `npx ng serve` en el terminal y nos debería de salir una página como esta



- Modificación del contenido

Trabajaremos dentro del componente buscador para realizar las funciones y poder buscar en el índice de elasticsearch.

Para ello, modificaremos el archivo buscador.component.ts:

```
import { Component } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { catchError, debounceTime, distinctUntilChanged, switchMap } from
'rxjs/operators';
import { Observable, of } from 'rxjs';

interface Car {
  marca: string;
  modelo: string;
  categoria: string;
  subcategoria: string;
}

interface SearchResult {
  hits: {
    total: { value: number };
    hits: { _source: Car }[];
  };
}

@Component({
  selector: 'app-buscador',
  templateUrl: './buscador.component.html',
  styleUrls: ['./buscador.component.css']
})
export class BuscadorComponent {
  searchResults: Car[] = [];
  totalResults = 0;
  shownResults = 0;
  searchQuery = '';

  constructor(private http: HttpClient) {}

  onSearchInput(query: string): void {
    this.searchQuery = query;

    if (query.length >= 2) {
      this.search(query).subscribe(
        (data: SearchResult) => {
          if (data && data.hits && data.hits.total && data.hits.hits) {
            this.totalResults = data.hits.total.value;
            this.shownResults = data.hits.hits.length;
            this.searchResults = data.hits.hits.map(result => result._source);
          } else {
            this.handleSearchError('Respuesta de Elasticsearch incompleta o
malformada');
          }
        },
        error => {
          this.handleSearchError('Error en la búsqueda', error);
        }
      );
    } else {
      this.clearResults();
    }
  }
}
```

Francisco González

```
    }
  }

  handleSearchError(message: string, error?: any): void {
    console.error(message, error);
    // Puedes agregar lógica adicional para manejar el error aquí
    // Por ejemplo, mostrar un mensaje de error al usuario
  }

  onChange(event: Event): void {
    const inputValue = (event.target as HTMLInputElement).value;
    this.onSearchInput(inputValue);
  }

  search(query: string): Observable<SearchResult> {
    const elasticSearchUrl = 'http://localhost:9200/delfincar_mapeo2/_search';
    const requestBody = {
      query: {
        bool: {
          should: [
            { match: { "marca": { query: query } } },
            { match: { "modelo": { query: query } } },
            { match: { "categoria": { query: query } } },
            { match: { "subcategoria": { query: query } } }
          ]
        }
      }
    };

    const headers = { 'Content-Type': 'application/json' };

    return this.http.post<SearchResult>(elasticSearchUrl, requestBody, { headers
  }).pipe(
    debounceTime(300),
    distinctUntilChanged(),
    catchError(error => of(error))
  );
}

highlightSearchQuery(text: string, query: string): string {
  const regex = new RegExp('(' + this.escapeRegExp(query) + ')', 'gi');
  return text.replace(regex, '<span style="background-color:
yellow;">$1</span>');
}

escapeRegExp(string: string): string {
  return string.replace(/[\.\*\+\?\^\$\{\}\|\[\]\\\]/g, "\\$&");
}

clearResults(): void {
  this.searchResults = [];
  this.totalResults = 0;
  this.shownResults = 0;
}
}
```

y posteriormente buscador.component.html:

```
<div id="app">
  <input type="text" id="searchInput" placeholder="marca, modelo, categoria o
subcategoria" (input)="onChange($event)">
  <div id="resultsInfo">{{ totalResults }} resultados - mostrando
{{ shownResults }}</div>
  <ul id="results">
    <li *ngFor="let car of searchResults">
      <p>
```

```
        <strong>{{ car.marca }} - {{ car.modelo }}</strong><br/>
        {{ car.categoria }} - {{ car.subcategoria }}
    </p>
</li>
</ul>
</div>
```

finalmente, habrá que asegurarse de que en `app.component.html`, hayamos introducido la etiqueta para llamar al componente de buscador y que cargue las búsquedas. También, habrá que verificar si en `app.module.ts`, tenemos importado el módulo de `HttpClientModule` e insertado en la sección `imports`, ya que sin este no podremos realizar las peticiones HTTP a `elasticsearch`.

Proyecto Filtros con AngularJS

Finalmente, vamos a recrear un buscador de filtros, en este caso, 3 campos para filtrar, los cuales serán marca, modelo y categoría respectivamente. Este apartado o componente será muy parecido al del buscador anteriormente realizado por lo cual podremos reciclar algo de código.

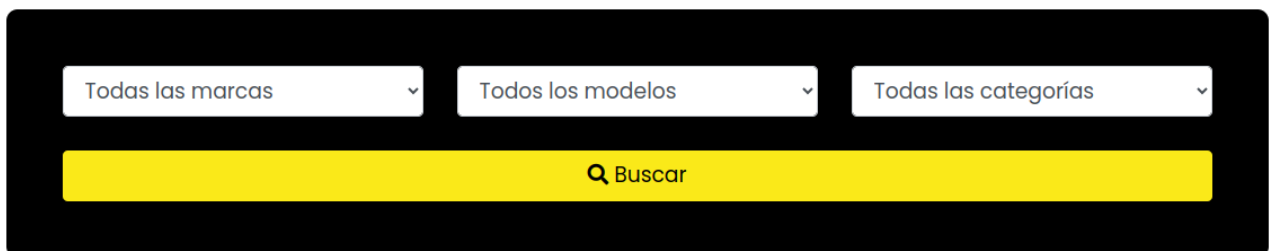


Diagrama de un buscador de filtros. El componente tiene un fondo negro y contiene tres selectores de texto blancos con flechas de selección hacia abajo. Los selectores están etiquetados como "Todas las marcas", "Todos los modelos" y "Todas las categorías". Debajo de los selectores hay un botón rectangular de color amarillo con el texto "Buscar" y un ícono de lupa a la izquierda.

Este es el buscador que queremos conseguir.

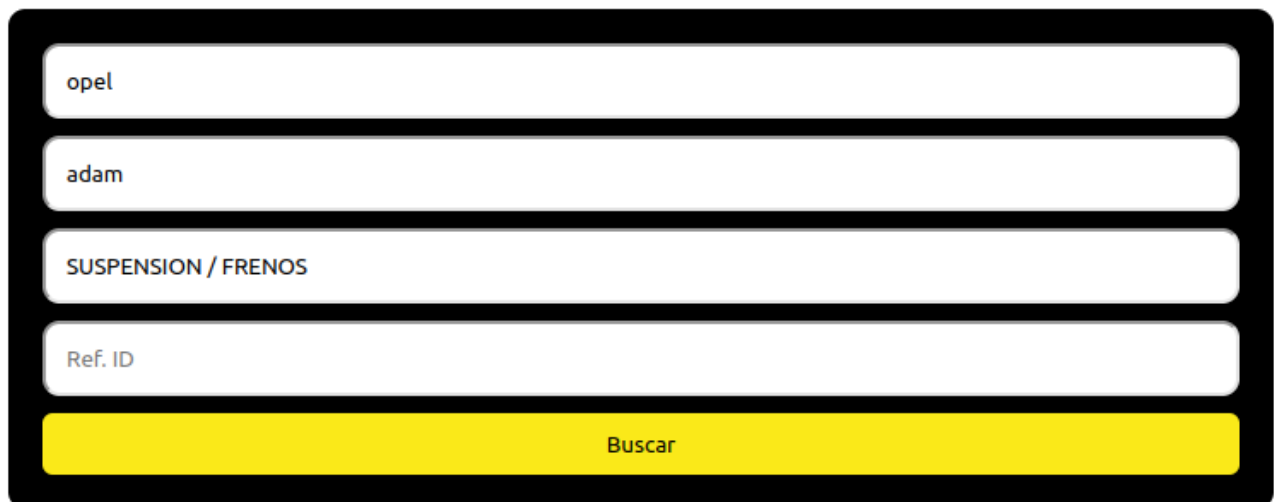


Diagrama de un buscador de filtros. El componente tiene un fondo negro y contiene cuatro campos de texto blancos. Los campos están etiquetados como "opel", "adam", "SUSPENSION / FRENOS" y "Ref. ID". Debajo de los campos hay un botón rectangular de color amarillo con el texto "Buscar".

Este sería el resultado.

Para conseguir este formulario, posteriormente modificaremos `filtros.component.html`

- Componente de filtros

Para empezar, se creará el componente de filtros para poder realizar el apartado de filtros correspondiente.

```
npx ng generate component filtros
```

Se crearán varios archivos, uno de ellos será filtros.component.html que tendrá el siguiente contenido:

```
<div id="filtros" class="container mt-5 text-dark">
  <div class="form-container">
    <input type="text" id="marca" [value]="selectedMarca" placeholder="Marca"
      (input)="onMarcaChange($event)" class="form-input">

    <input type="text" id="modelo" [value]="selectedModelo" placeholder="Modelo"
      (input)="onModeloChange($event)" class="form-input">

    <input type="text" id="categoria" [value]="selectedCategoria"
      placeholder="Categorías" (input)="onCategoriaChange($event)" class="form-input">

    <button (click)="buscar()" class="btn-buscar mt-3">Buscar</button>
  </div>
  <br>
  <div *ngIf="showResultados" class="mt-5">
    <h2 class="text-yellow">Resultados de la búsqueda:</h2>
    <p>{{ totalResultados }} resultados - mostrando {{ resultados.length }}</p>
    <div class="resultados-grid">
      <div *ngFor="let resultado of resultados" class="resultado-item">
        <p>ID: {{ resultado._source.id }}</p>
        <p>Marca: {{ resultado._source.marca }}</p>
        <p>Modelo: {{ resultado._source.modelo }}</p>
        <p>Categoría: {{ resultado._source.categoria }}</p>
      </div>
    </div>
  </div>
</div>
```

Y seguidamente modificaremos filtros.component.ts:

```
import { Component } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { Observable } from 'rxjs';

interface FiltroOptions {
  marcas: string[];
  modelos: string[];
  categorias: string[];
}

export interface MatchCondition {
  match: { [key: string]: string };
}

@Component({
  selector: 'app-filtros',
  templateUrl: './filtros.component.html',
  styleUrls: ['./filtros.component.css']
})
export class FiltrosComponent {
  filtroOptions: FiltroOptions = { marcas: [], modelos: [], categorias: [] };
  selectedMarca: string = '';
  selectedModelo: string = '';
  selectedCategoria: string = '';
```


Francisco González

```
modelosPorMarca: { [key: string]: string[] } = {
  'Renault': ["12", "14", "18", "19", "19 chamade", "19 hatchback", "21 berlina", "21 nevada",
    "25", "4 /berlina/familiar/furgoneta", "6", "7", "9", "B 120-35/50/55/60/65 Messenger", "Captur",
    "Clio", "Clio grandtour", "Clio I fase I+II", "Clio I fase III", "Clio II fase I", "Clio II fase
    II", "Clio II Symbol", "Clio III", "Clio IV", "Clio IV Grandtour", "Espace", "Espace/Grand espace",
    "espace IV", "fluence", "fuego", "Grand modus", "Kadjar", "Kangoo", "Kangoo 4x4",
    "Koleos", "Laguna", "Laguna Grandtour", "Laguna Grandtour III", "Laguna II", "Laguna II
    Grandtour", "Laguna III", "Mago", "Master", "Master Bus", "Master II PH. 2 Doka/PR.FGST", "Master II Phase
    2 cada cerrada", "Master II Phase 2 combi", "Master Kasten", "Master Kombi", "Megane", "Megane I Berlina
    Hatchback", "Megane I Cabriolet", "Megane I Classic", "Megane I Coach/Coupe", "Megane I Coupe Fase
    2", "Megane I Fase 2 Berlina", "Megane I Fase 2 Classic", "Megane I Fase 2 Grandtour", "Megane I
    scenic", "Megane II Berlina 3P", "Megane II Berlina 5P", "Megane II Classic Berlina", "Megane II
    Coupe/Cabrio", "Megane II Familiar", "Megane III Berlina 5 P", "Megane III Coupe", "Megane III Sport
    Tourer", "Megane IV", "MEgane IV Berlina 5p", "Megane IV Grandtour", "Megane scenic II", "Midliner s 120
    08A", "Modus", "R 11", "Rapid/Express", "Safrane", "Scenic", "Scenic II", "Scenic III", "Scenic
    Rx4", "Talisman", "Talisman Grandtour", "Trafic", "Trafic caja cerrada", "Trafic combi", "Trafic
    furgón", "Twingo", "Twingo III"]
  // ... Agrega más marcas y modelos según sea necesario
};

totalResultados: number = 0;
resultados: any[] = [];
showResultados: boolean = false;

constructor(private http: HttpClient) {}

onMarcaChange(event: any): void {
  this.selectedMarca = event.target.value;
}

onModeloChange(event: any): void {
  this.selectedModelo = event.target.value;
}

onCategoriaChange(event: any): void {
  this.selectedCategoria = event.target.value;
}

buscar(): void {
  // Lógica para realizar la búsqueda en Elasticsearch basada en las selecciones de filtros
  const requestBody: any = {
    query: {
      bool: {
        must: [] as MatchCondition[]
      }
    }
  };

  this.selectedMarca && requestBody.query.bool.must.push({ match: { marca:
this.selectedMarca } });
  this.selectedModelo && requestBody.query.bool.must.push({ match: { modelo: this.selectedModelo }
});
  this.selectedCategoria && requestBody.query.bool.must.push({ match: { categoria:
this.selectedCategoria } });
  console.log('Resultados de la búsqueda:');
  console.log('Marca:', this.selectedMarca);
  console.log('Modelo:', this.selectedModelo);
  console.log('Categoría:', this.selectedCategoria);

  // Puedes ajustar la URL según tu configuración de Elasticsearch
  const elasticSearchUrl = 'http://localhost:9200/delfincar_mapeo2/_search';

  this.http.post(elasticSearchUrl, requestBody).subscribe(
    (data: any) => {
      //console.log('Resultados de la búsqueda:', data.hits.hits);
      // Almacena los resultados
      this.totalResultados = data.hits.total.value;
      this.resultados = data.hits.hits;

      // Muestra los resultados en la interfaz
      this.showResultados = true;
    },
    error => {
      console.error('Error en la búsqueda:', error);
    }
  );
}
```

Una vez hecho estos cambios, dispondremos de un formulario de filtros para buscar piezas de coches, en este caso, estas piezas estarán implementadas en el índice de elasticsearch.

Veremos, que cuando buscamos algún coche ya sea por la marca o el modelo, nos saldrán los 10 resultados más próximos a los filtros aplicados.

Resultados de la búsqueda:

117 resultados - mostrando 10

ID: 4631354 Marca: OPEL Modelo: ADAM Categoría: ELECTRICIDAD	ID: 4357529 Marca: OPEL Modelo: ADAM Categoría: SUSPENSION / FRENOS	ID: 4357530 Marca: OPEL Modelo: ADAM Categoría: INTERIOR	ID: 4357531 Marca: OPEL Modelo: ADAM Categoría: INTERIOR
ID: 4357537 Marca: OPEL Modelo: ADAM Categoría: SUSPENSION / FRENOS	ID: 4357540 Marca: OPEL Modelo: ADAM Categoría: INTERIOR	ID: 4357544 Marca: OPEL Modelo: ADAM Categoría: SUSPENSION / FRENOS	ID: 4357553 Marca: OPEL Modelo: ADAM Categoría: MOTOR / ADMISION / ESCAPE
	ID: 4357554 Marca: OPEL Modelo: ADAM Categoría: DIRECCION / TRANSMISION	ID: 4357557 Marca: OPEL Modelo: ADAM Categoría: DIRECCION / TRANSMISION	

- Últimas modificaciones

Vamos a añadir un nuevo campo a los inputs que asemeje el código de barras pero que realmente su valor sea la ID de la pieza. Para ello:

Creamos una nueva variable que contendrá la ID escrita por el usuario:

```
selectedID: string = '';
```

Implementamos una nueva función para manejar el evento del input. Cuando el usuario introduzca toda la información en cada cuadro de texto se actualizará la variable selected por esta información.

```
onIDChange(event: any): void {  
    this.selectedID = event.target.value;  
}
```

Seguidamente, habrá que verificar si la variable ID tiene valor y en caso afirmativo, se confirmará si existe una coincidencia de datos.

```
this.selectedID && requestBody.query.bool.must.push({ match:  
{ id: this.selectedID } });
```

Por último, haremos una condición para mostrar un mensaje de inexistencia en caso de que la ID introducida no exista.

```
if (this.totalResultados === 0) {  
    // Muestra la alerta emergente  
    alert(`Pieza de coche con ID: ${this.selectedID} no  
encontrada`);  
}
```

Conteo de ítems

A partir de ahora, haremos uso de `_count` de `elasticsearch`, el cual se encarga de hacer conteos de ítems existentes en el índice. Haremos dos tipos de pruebas, un buscador y un listado de marcas.

Buscador de Marcas

La idea del buscador es muy simple, al introducir una marca, `elasticsearch` la buscara en su índice, y realizará un conteo de todas las piezas donde la marca de estas sea la que introduce el usuario por query.

Para empezar a crear este buscador:

```
npx ng generate component BuscarPorMarca
```

En el componente de `typescript`, realizaremos el funcionamiento del programa, específicamente la llamada al índice de `elasticsearch` donde se encuentran los recambios.

```
import { Component } from '@angular/core';
import { HttpClient } from '@angular/common/http';

interface Marca {
  nombre: string;
  conteo: number;
}

@Component({
  selector: 'app-buscar-por-marca',
  templateUrl: './buscar-por-marca.component.html',
  styleUrls: ['./buscar-por-marca.component.css']
})
export class BuscarPorMarcaComponent {
  marcaBuscada: string = '';
  marca: Marca | null = null;

  constructor(private http: HttpClient) {}

  buscarPorMarca(): void {
    // Puedes ajustar la URL según tu configuración de Elasticsearch
    const elasticSearchUrl = 'http://localhost:9200/delfincar_mapeo2/_count';

    const requestBody = {
      query: {
        match: {
          marca: this.marcaBuscada
        }
      }
    };

    this.http.post(elasticSearchUrl, requestBody).subscribe(
      (data: any) => {
        const conteo = data.count;

        // Si la marca existe, asigna los valores a la variable 'marca'
        this.marca = {
          nombre: this.marcaBuscada,
          conteo: conteo
        };
      },
      error => {
        console.error('Error al buscar la marca:', error);

        // Si hay un error, la marca no existe, muestra una alerta
        alert('No existe la marca: ${this.marcaBuscada}');
      }
    );
  }
}
```

En el componente de html, mostraremos el nombre de la marca y cuantas piezas tiene.

```
<div class="container mt-5 text-dark">
  <h1>Búsqueda por Marca</h1>
  <input type="text" [(ngModel)]="marcaBuscada"
placeholder="Introduce la marca" class="form-input">
  <button (click)="buscarPorMarca()" class="btn-buscar mt-
3">Buscar</button>

  <div *ngIf="marca !== null" class="mt-5">
    <h2 class="text-yellow">Detalles de la marca:</h2>
    <p>Nombre: {{ marca.nombre }}</p>
    <p>Total de Piezas: {{ marca.conteo }}</p>
  </div>
</div>
```

Con todo esto implementado, tendremos un simple buscador que contará los ítems dependiendo de la marca introducida.

Listado de marcas

El listado mostrará todas las marcas disponibles en el índice de elastic, y mostrará el nombre, cuantos recambios existen de esta marca y el logo.

En el componente de typescript, realizaremos el funcionamiento del programa, específicamente la llamada al índice de elasticsearch donde se encuentran los recambios.

```
// desplegable-marcas.component.ts
import { Component, OnInit } from '@angular/core';
import { HttpClient } from '@angular/common/http';

interface Marca {
  nombre: string;
  logo: string; // Puedes ajustar el tipo según sea necesario
  conteo: number;
}

interface Logos {
  [marca: string]: string;
}

@Component({
  selector: 'app-desplegable-marcas',
  templateUrl: './desplegable-marcas.component.html',
  styleUrls: ['./desplegable-marcas.component.css']
})
export class DesplegableMarcasComponent implements OnInit {
  marcas: Marca[] = [];

  constructor(private http: HttpClient) { }

  ngOnInit(): void {
    // Aquí haces la solicitud a Elasticsearch para obtener el conteo de cada marca
    this.obtenerConteoMarcas();
  }

  obtenerConteoMarcas(): void {
    // Puedes ajustar la URL según tu configuración de Elasticsearch
    const elasticSearchUrl = 'http://localhost:9200/delfincar_mapeo2/_count';

    // Supongamos que tus marcas están en un array llamado 'marcasDisponibles'
    const marcasDisponibles = ['Audi', 'BMW', 'Citroën', 'Ford', 'Hyundai', 'Kia', 'Mercedes Benz', 'Nissan', 'Opel',
    'Peugeot', 'Renault', 'Seat', 'Toyota', 'Volkswagen', 'DS', 'Alfa Romeo', 'Aro', 'Asia Motors', 'Aston Martin', 'Austin', 'Authi', 'Auto
```

Francisco González

```
Bianchi', 'Auverland', 'Bentley', 'Bertone', 'Buick', 'Cadillac', 'Caterham', 'Chevrolet', 'Chrysler', 'Daewoo', 'Daihatsu', 'De
Tomaso', 'Ferrari', 'Fiat', 'FSM', 'FSO', 'Galloper', 'Honda', 'Iato', 'Innocenti', 'Isuzu', 'Jaguar', 'Jeep', 'Lada Autovaz', 'Lancia', 'Land
Rover', 'Lexus', 'Lotus', 'Mahindra', 'Mazda', 'Maserati', 'MG', 'Mitsubishi', 'Morgan', 'Oldmobile', 'Pontiac', 'Porsche', 'Suzuki', 'Rolls
Royce', 'Rover', 'Saab', 'Santana', 'Skoda', 'Smart', 'Ssangyong', 'Subaru', 'Suzuki', 'Talbot', 'Tata', 'Triumph', 'Uaz
Martorelli', 'UMM', 'Volvo', 'Wartburg', 'Zastava', 'Daf', 'Daimler', 'Dodge', 'Gme', 'Hummer', 'Iveco-Pegaso', 'Lamborghini', 'Maybach', 'Renault
V.I.', 'Dacia', 'Suzuki', 'Piaggio', 'Infinity', 'Mini', 'Abarth', 'Fornasari', 'McLaren', 'Tesla', 'Scania', 'Man', 'Ausa', 'Pegaso', 'Comarth', 'Z
eus', 'Still', 'Aixam', 'MG Rover', 'Kymco', 'Chatenet', 'SYM', 'Aprilia', 'Maxus', 'Miles'];
const logos: Logos = {
  Renault: 'https://upload.wikimedia.org/wikipedia/commons/thumb/4/49/Renault_2009_logo.svg/2048px-Renault_2009_logo.svg.png',
  Opel: 'https://upload.wikimedia.org/wikipedia/commons/thumb/7/7b/Opel-Logo_2017.png/1200px-Opel-Logo_2017.png',
  Ford: 'https://upload.wikimedia.org/wikipedia/commons/thumb/a/a0/Ford_Motor_Company_Logo.svg/2560px-
Ford_Motor_Company_Logo.svg.png'
};

// Realiza la solicitud para cada marca
for (const marca of marcasDisponibles) {
  const requestBody = {
    query: {
      match: {
        marca: marca
      }
    }
  };

  this.http.post(elasticSearchUrl, requestBody).subscribe(
    (data: any) => {
      const conteo = data.count;
      const logo = logos[marca]; // Obtiene la URL del logo de la marca

      // Agrega la información de la marca al array
      this.marcas.push({ nombre: marca, logo: logo, conteo: conteo });
    },
    error => {
      console.error('Error al obtener el conteo de la marca', marca, error);
    }
  );
}
}
```

En el componente de html, mostraremos una tabla con los detalles.

```
<!-- desplegable-marcas.component.html -->
<h1>Listado de items</h1>
<table class="table table-bordered">
  <thead>
    <tr>
      <th scope="col">Marca</th>
      <th scope="col">Piezas</th>
      <th scope="col">Logo</th>
    </tr>
  </thead>
  <tbody>
    <tr *ngFor="let marca of marcas">
      <td>{{ marca.nombre }}</td>
      <td>{{ marca.conteo }}</td>
      <td><img [src]="marca.logo" alt="{{ marca.nombre }} logo"
class="marca-logo"></td>
    </tr>
  </tbody>
</table>
```

Bibliografía

[Primeros pasos y operaciones con elasticsearch](#)

[Guía básica Docker y Elasticsearch](#)

[Configuración Docker y Elasticsearch](#)

[Mock Data](#)

[Elasticsearch Bool Query \(MUST & SHOULD\)](#)

[Count API](#)

[SQL-JOINS en Elasticsearch](#)

[Elasticsearch y SQL](#)

[Diferencias entre Elasticsearch y Bases de datos tradicionales](#)

[Eliminar documentos con elasticsearch](#)

[Instalación de Java para ElasticStack](#)

[Pagina de descarga elasticsearch y kibana](#)

[Search-as-you-type](#)