TECNOLOGÍAS BASES DE DATOS

PRÁCTICA OBLIGATORIA

GRUPO 6 – 22 MAYO 2020

CONTENIDO

CONSIDERACIONES GENERALES

GRUPO 6

Daniel Anguita Viñas

Alfonso Casanova Muñoz

Ana María Jurado Crespo

David Rial Vega

Juan Manuel Verano Garri

RESÚMEN

En esta práctica hemos construído una aplicación web donde poder hacer consultas a una base de datos en MongoDB con datos de personajes y cómics principalmente del universo Marvel.

La base de datos ha sido poblada con datos provenientes de 7 ficheros distintos en formato CSV aportados por el profesor. Nuestro trabajo ha consistido en entender la naturaleza y estructura de esos datos y tratarlos y procesarlos para poblar una base de datos NoSQL orientada a objetos en MongoDB. Una vez poblada esa base de datos, que en nuestro caso ha quedado dividida en dos colecciones (characters y comics) hemos implementado 14 consultas distintas (11 sencillas y 3 complejas) que pueden ser visualizadas en la aplicación web que hemos implementado para tal fin.

TECNOLOGÍAS

Las tecnologías usadas para esta práctica son las siguientes:

- **Angular [v9]:** El framework utilizado para desarrollar el frontend. Escogimos esta opción dada nuestra afinidad hacia él por la asignatura de DAW y por su conocida fiabilidad como producto desarrollado por Google.

- **NodeJS [v12.16.3]:** Dado que el uso de MongoDB estaba impuesto por la práctica, nos pareció lógico el uso de node dado que ambos utilizan de forma nativa JSON y también ha sido desarrollado por Google, además del plus de tener el frontend y el backend en el mismo lenguaje.
- **Swagger[OpenAPI-3.0]:** Se trata de una herramienta para la especificación de APIs. Elegimos esta opción dado que nos proporcionaba una forma de especificar claramente los métodos del API que utilizaremos como backend. Además nos proporciona algunas facilidades como la generación de una estructura de código predeterminada así como una fácil documentación de todos los métodos. Esto último nos ha ayudado a mantener una muy buena coordinación en el equipo.
- **Mongoose:** Se trata de una librería en NodeJS para interactuar con bases de datos mongo. Proporciona una forma muy cómoda de manejar los datos basada en "esquemas".
- **MongoDB en CLOUD [4.2]:** Decidimos alojar nuestra base de datos en un servidor cloud. Esto nos ha ayudado a agilizar el proceso de desarrollo, previniendonos de la necesidad de levantar y alimentar una base de datos en local cada uno de nosotros, además ha ayudado a mantener la integridad de los datos que manejamos por separado.
- **Amazon Web Services [EC2]:** Decidimos alojar la aplicación en una máquina EC2 de AWS, gracias a esto podemos tener una "versión de prueba" de la aplicación.

ORGANIZACIÓN Y REPARTO DE TAREAS

Decidimos hacer todos juntos la parte de exploración de los datos y la toma de decisiones sobre el "esquema" que iban a tener nuestras colecciones en BD.

A continuación separamos las tareas en un subequipo de dos personas para el back, otro subequipo de dos personas para el front y otro subequipo de una persona de apoyo a ambos subequipos y principal encargada de la memoria.

FASES

FASE 0: EXAMINANDO LOS DATOS

Para realizar esta práctica se nos ha proporcionado un conjunto de siete ficheros en formato CSV. Lo primero que hicimos fue abrir estos ficheros para ver los datos y formato de los mismos. A continuación vamos a mostrar brevemente la información de los datos de cada uno de los ficheros y la relación entre ellos.

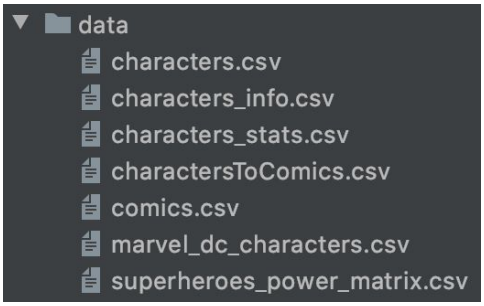

Imagen 1. Ficheros csv aportados por el profesor

Algunos comentarios:

characters.csv está relacionado con characters_info.csv, characters_stats.csv, mervel_dc_characters.csv, superheroes_power_matrix.csv a través del campo nombre.

characters.csv y **comics.csv** se relacionan entre sí a través del fichero charactersToComic.csv

marvel_dc_characters.csv - Tuvimos algunos problemas de formateado con los valores de la columna “FirstAppearance”.

superheroes_power_matrix.csv - Este fichero tiene 168 columnas. Contiene información sobre los poderes que tienen los personajes. Los personajes están presentes por el nombre.

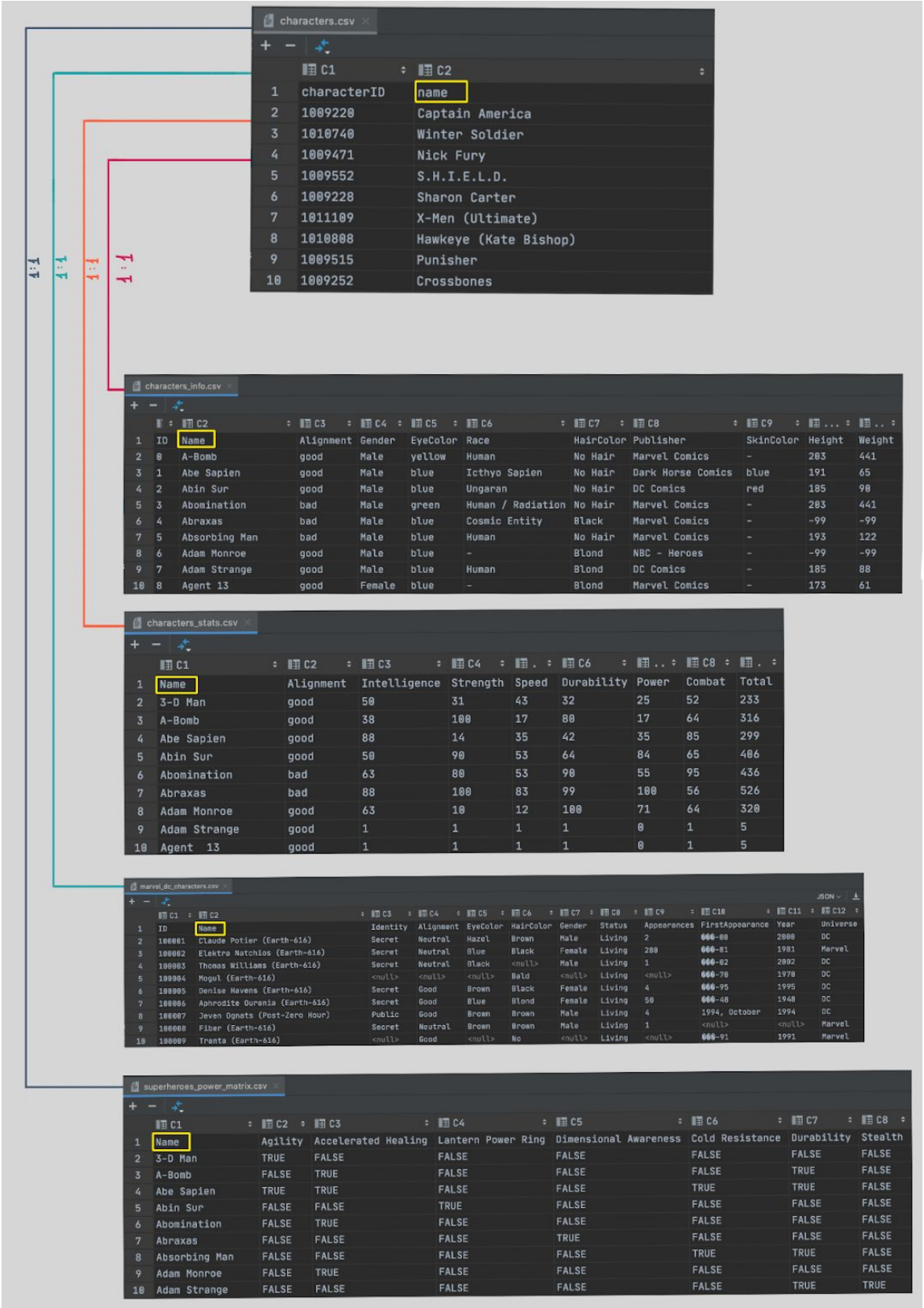

Imagen 2. Relación entre CHARACTERS y cuatro de los ficheros

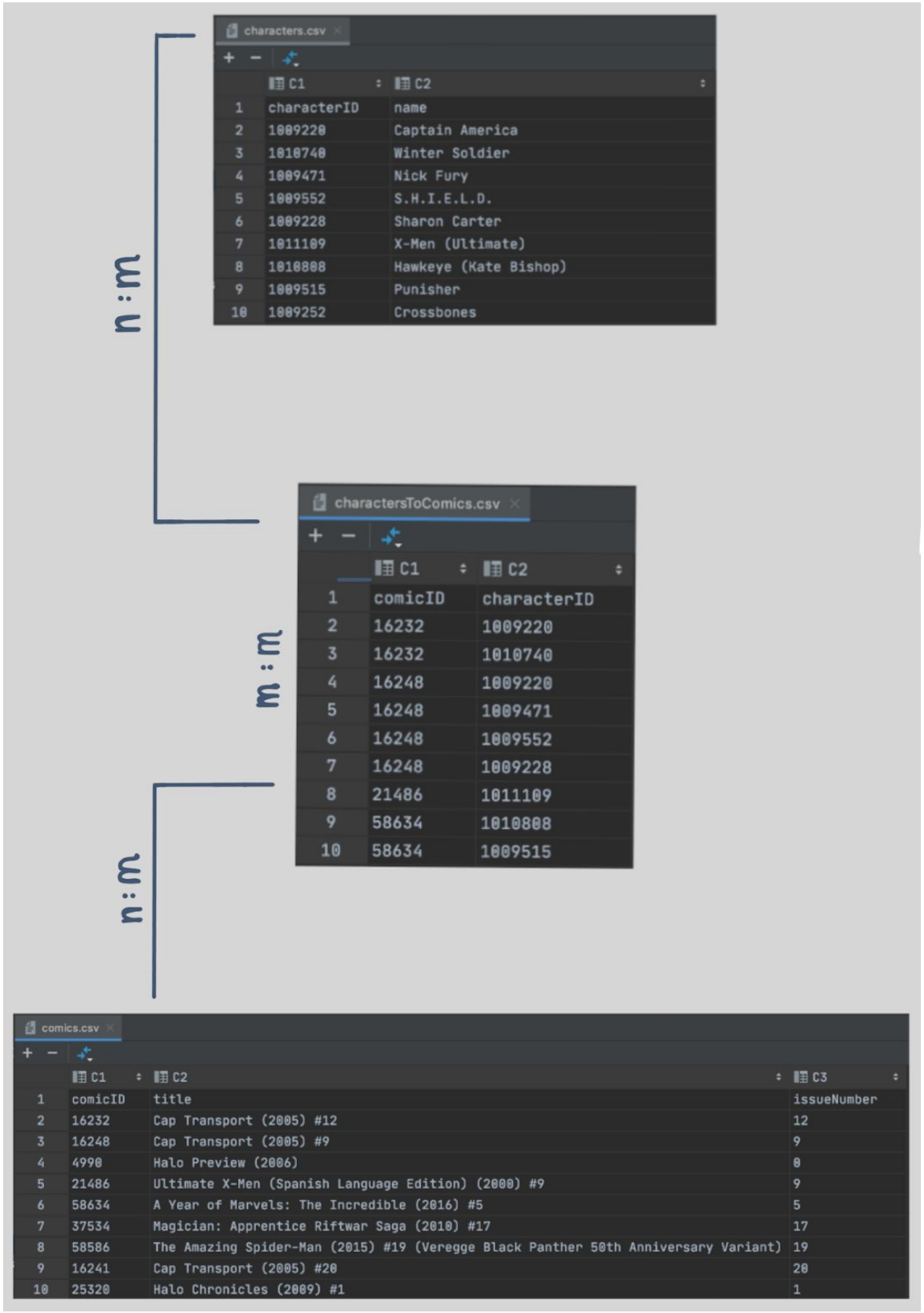

Imagen 3. Relación entre characters y comics

FASE 1: LECTURA, LIMPIEZA, PROCESAMIENTO Y VOLCADO

Realizado el primer análisis de los datos, pensamos en qué información nos interesaba incluir en nuestra base de datos y qué tareas de procesamiento eran necesarias.

Para esto diseñamos y desarrollamos un script que hace lo siguiente:

1. Leer los ficheros linea a linea y limpiar posibles inconsistencias
2. Guardar, a medida que va leyendo los ficheros, los datos en memoria.
3. Convertir los datos de la memoria a objetos esquema de mongoose para su inserción.

LECTURA Y LIMPIEZA

Para la lectura, nos centramos en eliminar algunas de las inconsistencias que tenían ciertos campos. Por ejemplo, eliminar espacios iniciales y finales o borrar comas y comillas en campos internos. Este proceso se lleva a cabo en este método:

```
readFileLineByLine(fileName, lineConversor) {
    const liner = new lineByLine(`data/${fileName}`);
    let line;
    const result = [];
    let firstLine = true;
    let fieldNames = [];

    while (line = liner.next()) {
        let lineStr = line.toString('utf-8').replace( searchValue: '\r',  replaceValue: '');
        if (firstLine) {
            fieldNames = lineStr.split( separator: ',');
            fieldNames = fieldNames.map(fn => fn.replace( searchValue: / /g,  replaceValue: '').trim());
            firstLine = false;
        } else {
            if (lineStr.includes('"')) {

                lineStr.match( matcher: /"(.*?)"/g).forEach(function (val : string ) {
                    const replaced = val.replace( searchValue: /,/g,  replaceValue: '');
                    lineStr = lineStr.replace(val, replaced);
                    lineStr = lineStr.replace( searchValue: /"/g,  replaceValue: '');
                });
            }
            result.push(lineConversor(lineStr, fieldNames));
        }
    }
    return result;
}
```

Image 4. Función `readFileLineByLine` en dumper/src/conversor/DataConversor.js

ALMACENADO EN MEMORIA

A medida que leemos los ficheros y eliminamos las inconsistencias de algunos campos, almacenamos su contenido en memoria, para hacerlo accesible posteriormente en la fase de procesamiento y volcado.

En este método almacenamos los datos de los ficheros en la memoria, la estructura continúa siendo la misma.

```
convertData() {
    const characters = this.convertCharactersData();
    const charactersInfo = this.convertCharactersInfoData();
    const charactersStats = this.convertCharacterStatsData();
    const characterToComic = this.convertCharacterToComicData();
    const crossOvers = this.convertCrossoversInfoData();
    const comics = this.convertComicData();
    const powers = this.convertSuperPowersData();

    return {
        characters: characters,
        charactersInfo: charactersInfo,
        charactersStats: charactersStats,
        characterToComic: characterToComic,
        crossOvers: crossOvers,
        comics: comics,
        powers: powers
    };
}
```

Image 5. Función `convertData` en dumper/src/manager/DataManager.js

PROCESAMIENTO Y VOLCADO

Procesamiento y volcado se realizan ambos en un mismo paso.

Para ayudar al procesamiento, incluimos algunas funciones de ayuda, que nos permiten realizar consultas previas sobre los datos en memoria.

```
getAllComics() {
    return this.getData()
        .comics;
}

getCharacterById(id) {
    return this.getData().characters
        .find(c => c.characterID === id);
}

getCharacterByName(name) {
    return this.getData().characters
        .find(c => c.name === name);
}

getCharacterInfo(characterName) {
    return this.getData().charactersInfo
        .find(ci => ci.Name === characterName);
}

getCharacterStats(characterName) {
    return this.getData().charactersStats
        .find(cs => cs.Name === characterName);
}

getComicById(id) {
    return this.getData().comics
        .find(c => c.comicID === id);
}

getComicsWhereCharacterAppears(characterName) {
    const character = this.getCharacterByName(characterName);
    if (character) {
        return this.getData().characterToComic
            .filter(a => a.characterID === character.characterID)
            .map(a => this.getComicById(a.comicID));
    }
}
```

Image 6. Funciones de apoyo en dumper/src/service/DataService.js

Durante el procesamiento decidimos excluir algunos de los datos, los cuales no nos sirven, además de añadir ciertas modificaciones a los datos existentes:

1. **Filtrar** los datos correspondientes al fichero Marvel-DC-Characters.csv, para esto, excluimos los datos que **NO** estaban relacionados con los datos almacenados en Characters.csv. La relación que tienen es por nombre de character (por ejemplo “Captain America” aparece en las dos)

2. **Convertir** los datos a objetos de mongoose, ya con la estructura final que tendrán en la base de datos (detallada en la fase 2) para el caso de los datos del fichero Marvel-DC-Characters.csv, decidimos nombrarlos como “CrossOvers” dado que aparentemente almacenan los datos de apariciones de characters en comics junto con otros publishers (Marvel vs DC). Además separamos algunos datos adicionales que aparecían en el nombre, como por ejemplo (EARTH-616) haciendo referencia al universo, o algunas aclaraciones que aparecen entre paréntesis también, guardandolas como “información adicional”.

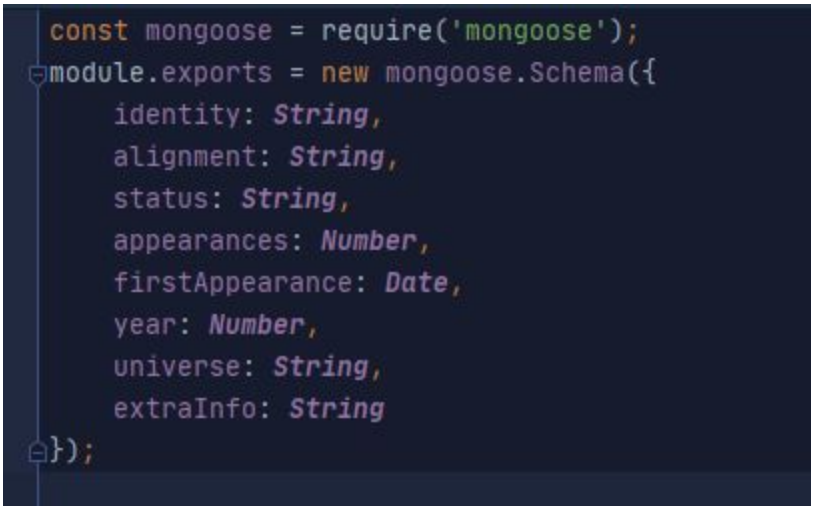

Image 7. Esquema mongoose para los datos del fichero Marvel-DC-Characters.csv

3. **Volcar** los datos en la base de datos, para esto convertimos los objetos de la memoria en modelos de mongoose y los insertamos por batches (ahorrando tiempo) en la base de datos.

```
recursiveDumpComics(index, batches, callback) {

    const batch = batches[index];
    const comics = [];
    console.log(`Dumping batch ${index} of comics!`);

    for (const comic of batch) {
        const comicModel = new Comic({
            _id: comic.comicID,
            name: comic.title.trim(),
            issue: comic.issueNumber
        });
        comics.push(comicModel);
    }

    Comic.collection.insert(comics, (err, resp) => {
        if (err) {
            callback(err);
        }
        if (resp) {
            if (index >= batches.length - 1) {
                callback();
            } else {
                this.recursiveDumpComics( index: index + 1, batches, callback);
            }
        }
    });
}
```

Image 8. Función recursiva para insertar los comics en la base de datos.

FASE 2: ESTRUCTURA DE LOS DATOS

Hemos decidido tener dos colecciones: characters y comics

Vamos a desgranar a continuación el "esquema" de un documento de cada colección para explicar su estructura y la procedencia de los datos de esa estructura.

CHARACTERS

En el primer nivel nos encontramos con estos campos a los que hemos llamado nodos:

```
{  "_id":1009610,
   "comics":[...],
   "crossovers":[{},{},...],
   "info":{...},
   "name":"Spider-Man",
   "powers":[ "Agility", "AcceleratedHealing", ... ],
   "stats":{ "_id":{...}, ... } }
```

Vamos ahora a explicar en detalle cada uno de estos campos de primer nivel.

"comics"

Es una lista de comicsIDS que están sacados de relacionar el characterID con comicID en el fichero **charactersToComics.csv**

Imagen 9. Ejemplo nodo comic

"crossovers"

Es una lista de crossovers, es decir, apariciones de ese personaje en otras editoriales ("publishers") distintas de la suya. La información se ha obtenido del fichero **marvel_dc_characters.csv.** Puede ser una lista vacía.

Tuvimos en cuenta las indicaciones del profesor y declaramos las siguientes constantes:

```
const DC = 'Earth-616';
const marvel = 'New Earth';
```

CrossOver.js

```
const DynamicModel = require('../basic/DynamicModel');
const DC = 'Earth-616';
const marvel = 'New Earth';

class CrossOver extends DynamicModel {

    constructor(fieldNames, fieldValues) {
        super(fieldNames, fieldValues);
        let nameDivisions = this.Name
            .match(/\((.*?)\)/g)

        if (nameDivisions) {
            nameDivisions = nameDivisions.map(n => n.replace(/\(/g, '').replace( searchValue: /\)/g,  replaceValue: ''));
            nameDivisions.forEach(nd => this.Name = this.Name.replace(`(${nd})`, '').trim());
            this.extraInfo = this.getExtraInfo(nameDivisions);
        }
    }

    getExtraInfo(nameDivisions) {
        if (nameDivisions.length > 0) {
            for (const division of nameDivisions) {
                if (!division.includes(DC) && !division.includes(marvel)) {
                    return division;
                }
            }
        }
        return null;
    }
}
```

Imagen 10. Código de la clase para obtener los nodos crossover.

dumper/src/model/character/CrossOver.js


```
"crossovers":[
    {
        "_id":{
            "$oid":"5ebad7fc00a88a00c4c9ae22"
        },
        "identity":"Secret",
        "alignment":"good",
        "status":"living",
        "appearances":4043,
        "firstAppearance":{
            "$date":"1961-12-31T23:00:00.000Z"
        },
        "year":1962,
        "universe":"Marvel",
        "extraInfo":"Peter Parker"
    },
    ...
],
```


Imagen 11. Ejemplo nodo crossovers

“info”

Es una lista de pares clave valor con los datos contenidos en el fichero **characters_info.csv.** Puede ser un campo con sólo _id, sin valores para “alignment”, “gender”, etc.

Imagen 12. Ejemplo nodo info

“name”

Es un campo clave valor con el nombre del personaje

“powers”

Este nodo es una lista de strings. Cada string es el nombre de un poder. Cada personaje que tiene poderes tiene los que tienen su valor a TRUE en la tabla del fichero **superheroes_power_matrix.csv.** Puede ser una lista vacía.

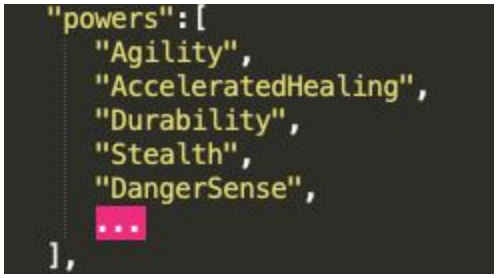

Imagen 13. Ejemplo nodo powers

“stats”

Es una lista de pares clave valor con los datos contenidos en el fichero **characters_stats.csv.** Puede ser un campo con sólo _id, sin valores para “intelligence”, “strength”, etc.

Imagen 14. Ejemplo nodo stats

character

Imagen 15. Ejemplo documento character completo

COMICS

El documento comic es una traducción literal del fichero comics.csv. No hemos procesado ni limpiado nada en el proceso de creación de esta colección.

Imagen 16. Ejemplo documento comic completo

METADATA

Una vez volcados todos los datos a nuestra base de datos en MongoDB hemos realizado algunas consultas básicas para ver qué pinta tienen los datos y cómo de completos están al juntar tanta la información dispersa de los ficheros csv. Estos son los metadatos que quisimos averiguar y las queries usadas para ello.

Número total de personajes: 1170

```
db.characters.count();
```

Número total de comics: 38875

```
db.comics.count();
```

Número de personajes con "crossover" info: 368

```
db.characters.count({crossovers: {$ne: []}});
```

Número de personajes con "powers" info: 200

```
db.characters.count({powers: {$ne: []}});
```

Número de personajes con "info" info: 215

```
db.characters.count({$or: [
   {"info.alignment": { "$exists": true }},
   {"info.gender":{"$exists": true}},
   {"info.hair_color":{"$exists": true}},
   {"info.height":{"$exists": true}},
   {"info.publisher":{"$exists": true}},
   {"info.skin_color":{"$exists": true}},
   {"info.weight":{"$exists": true}},
   {"info.eye_color":{"$exists": true}}]
});
```

Número de personajes con "stats" info: 197

```
db.characters.count({$or: [
       {"stats.combat": { "$exists": true }},
       {"stats.durability":{"$exists": true}},
       {"stats.intelligence":{"$exists": true}},
       {"stats.power":{"$exists": true}},
       {"stats.speed":{"$exists": true}},
       {"stats.strength":{"$exists": true}}]
});
```

Promedio de poderes por personaje: 9,75

```
db.characters.aggregate([
   // First stage: Filter the chars that have powers
   { $match: {powers: {$ne: []}}},
   // Second stage: count the number of chars with powers and sum the number of powers
   { $group:
           {
               _id: {},
               totalPower: {$sum: {$cond: {if: {$isArray: "$powers"},
                                           then: {$size: "$powers"}, else: 0}}},
               countCharsWithPowers: {$sum: 1},
           }
   }
]);
```

```
    "countCharsWithPowers": 200

    "totalPower": 1950
```

Promedio de apariciones por personaje: 64,45

```
db.characters.aggregate([
   // First stage: count the number of chars with appearances and sum the number of
powers
   { $group:
           {
               _id: {},
               totalAppearances: {$sum: {$cond: {if: {$isArray: "$comics"},
                                                 then: {$size: "$comics"},
                                                 else: 0}}},
               countAppearances: {$sum: 1},
           }
   }
]);
```

"countAppearances": 1170

"totalAppearances": 75402

Como se puede comprobar, a pesar de tener más de mil personajes tenemos información extra sobre poderes, estadísticas, información y crossover para aproximadamente el 20% de los personajes. Por tanto las queries que vamos a hacer a continuación y los resultados que se muestran en la aplicación web que hemos construído se verán afectadas por la falta de datos para muchos de los personajes.

FASE 3: CONSULTAS

En esta sección vamos a hablar de las queries que hemos implementado para esta práctica. En el enunciado de la práctica se nos pedía que implementáramos al menos 10 consultas de las cuales mínimo 3 debían ser “complejas”. Entendiendo por “compleja” toda aquella consulta que utilice más de una colección.

En nuestra base de datos tenemos dos colecciones como ya hemos comentado anteriormente: **characters** y **comics**. Por tanto hemos hecho la división de las consultas entre sencillas y complejas teniendo en cuenta si la consulta usa uno o ambas colecciones.

Para más información sobre las consultas, consultar el APÉNDICE A

CONSULTAS SENCILLAS

1. **¿Quién es el más gordo?**: Dame el personaje más gordo de todos.
2. **¿Quién es el más alto?**: Dame el personaje más alto de todos.
3. **¿Quién es el héroe más poderoso?**: El héroe con más poderes.
4. **¿Quién nos salvará del coronavirus?**: Todos los personajes que tienen DOCTOR en el nombre.
    - Esta consulta surgió de saber qué personajes contenían “DOCTOR” en su nombre. En la aplicación se muestra como una consulta donde el usuario puede introducir la cadena de texto que quiere buscar dentro del nombre del personaje.
5. **Las chicas más populares:** Top 10 de personajes femeninos con más apariciones
6. **Las chicas son guerreras**: Todos los personajes femeninos ordenadas por la cantidad de poderes que tienen.
7. **Malos héroes:** Todos los héroes que han sido villanos en un crossover.
8. **El villano más listo:** Busca al personaje villano con más puntos de inteligencia.
9. **El héroe más tonto:** Busca al personaje héroe con menos puntos de inteligencia.
10. **Los más poderosos:** Top 10 de personajes más poderosos
11. **Los que más aparecen:** Top 5 de personajes que más apariciones tienen separados por su alignment

CONSULTAS COMPLEJAS

12. **¿Quién es el héroe con mejores estadísticas del cómic?**: Dado un cómic, busca el héroe con mejores estadísticas.
13. **El comic de los vigoréxicos:** De una muestra aleatoria de 1000 comics, obtiene el cómic que tiene los héroes más fuertes.
14. **Buscador de villanos:** Busca todos los villanos que comparten comic con un personaje.

FASE 4: VISUALIZACIÓN DE CONSULTAS Y RESULTADOS

Para el Front hemos usado Angular 9, junto a Angular Material, para la parte de diseño, con los componentes mat-table donde se muestran los resultados de las consultas, mat-input para las consultas que requieren de algún filtro por parte del usuario como es la búsqueda de personaje por nombre, mat-select para seleccionar la consulta a mostrar y por último el componente ngx-spinner, para mostrar un spinner durante la carga de los datos e informar al usuario que se está cargando la consulta. A continuación se muestran unos pantallazos de los componentes mencionados.

La aplicación web se compone de una única pantalla principal. Como se puede observar en la Figura 1. Si hacemos scroll hacia abajo, podemos encontrar tres “tabs” con consultas de personajes, Top consultas de apariciones y Consultas sobre cómics tal como se aprecia en la Figura 2 y 3.

Página principal

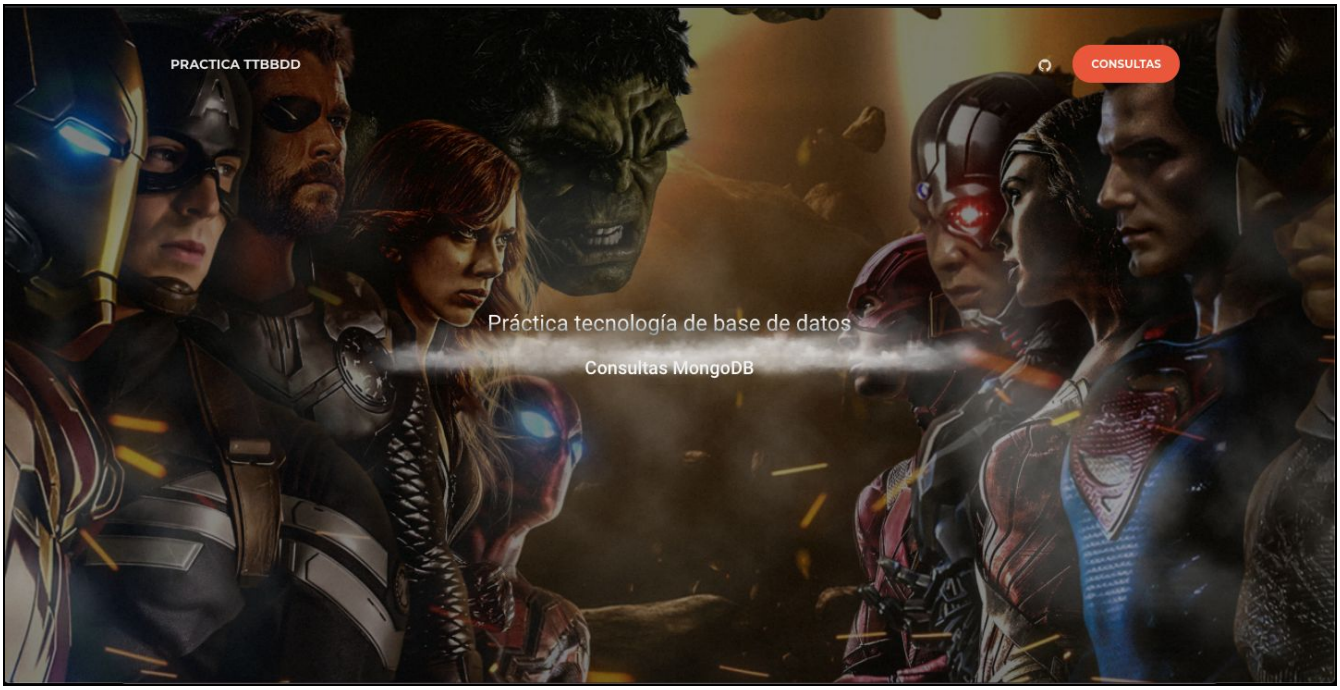

Imagen 17. Aplicación web - Home

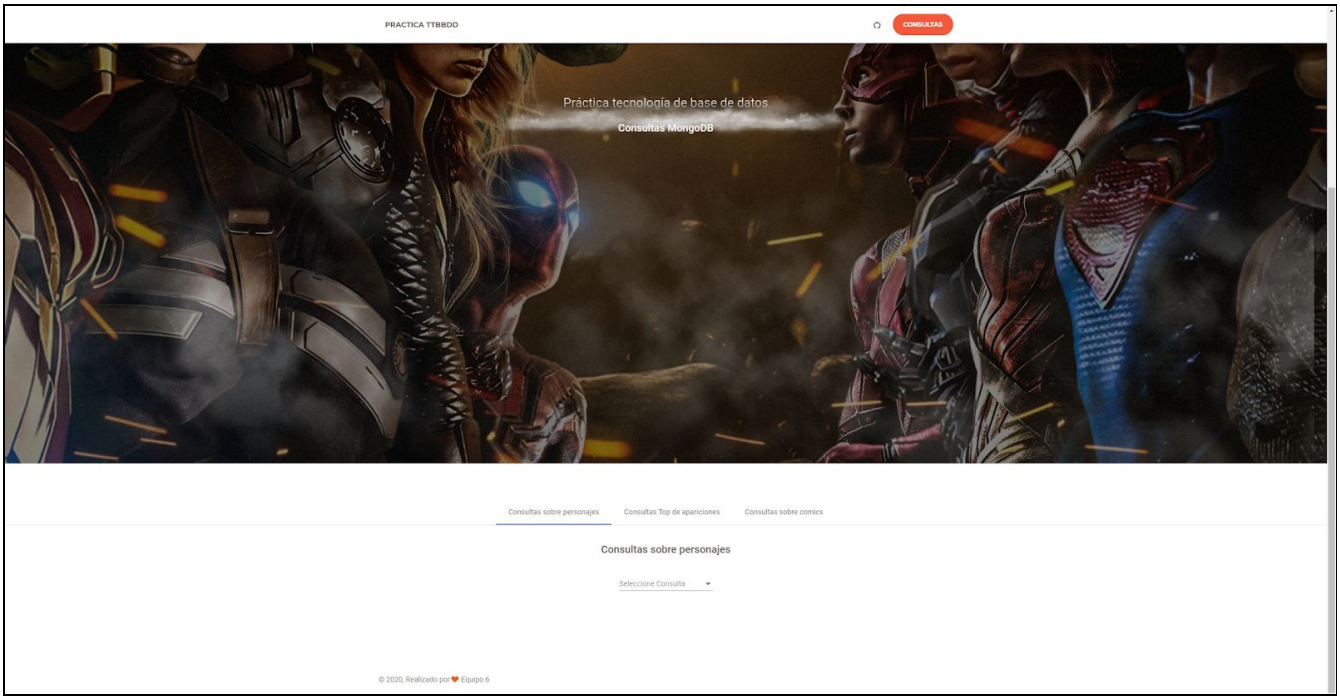

Imagen 18. Aplicación web – Consultas sobre personajes

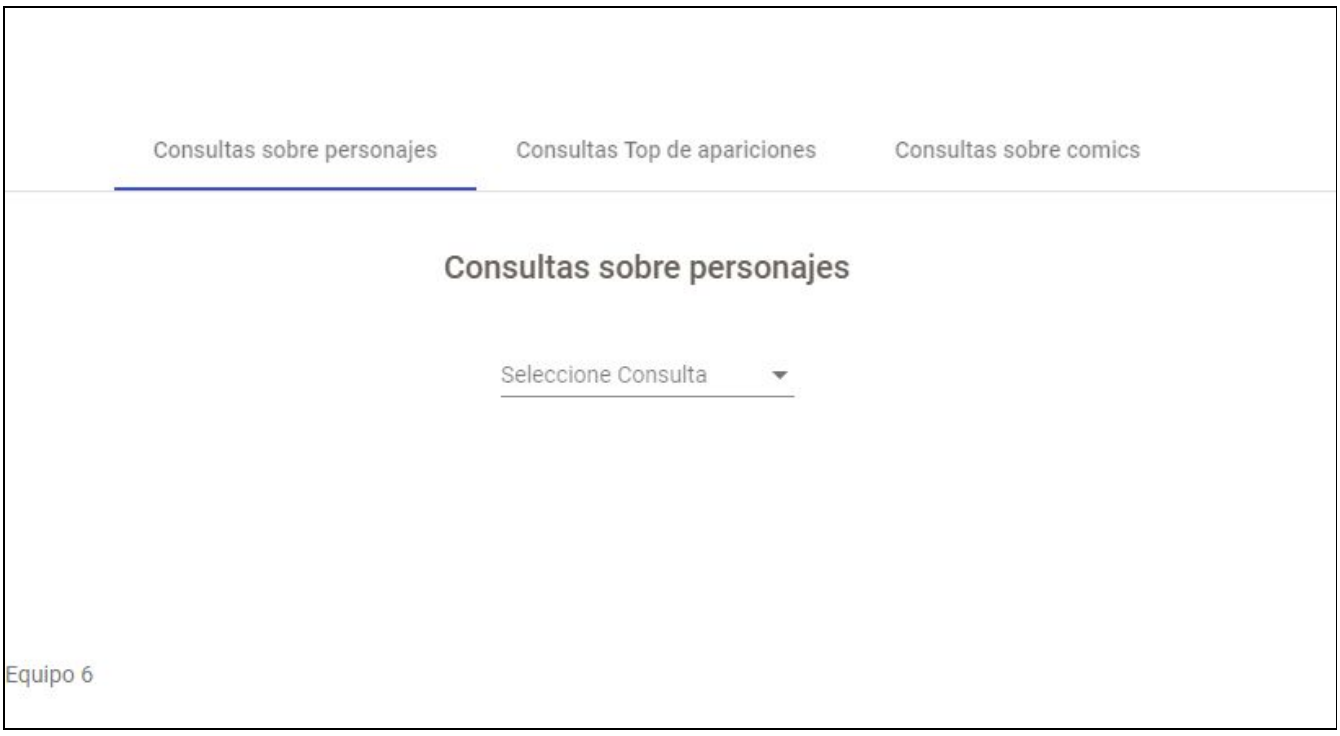

Imagen 18. Aplicación web – Consultas sobre personajes Detalle

“Consultas sobre personajes”

En esta opción se nos presentan a su vez diversas consultas recogidas en un mat-option como podemos observar en la **Imagen 19.** Aquí podemos encontrar las consultas de la 1-4, 6-9, 12, 14 y 15. Algunas consultas como la de “Todos los héroes” nos devuelve como su nombre indica todos los héroes ordenador en una tabla que es paginable y en la que se puede hacer scroll tal como muestra la **Imagen 20**

Otras consultas devuelven el resultado con una simple frase dado que solo se les pide un nombre tal como se observa en la **Imagen 21.**

Otras sin embargo le dan la posibilidad al usuario para interactuar con la aplicación ofreciendo un input de datos para buscar, por ejemplo, el número de identificación de un héroe y encontrar una lista de todos los villanos con los que se ha cruzado en algún cómic. **Imagen 20.**

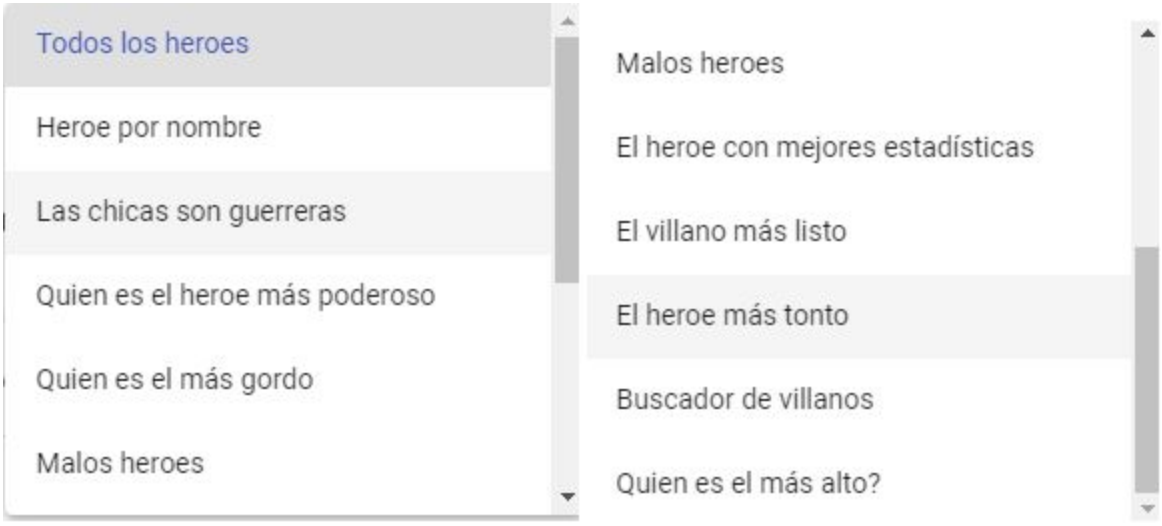

Imagen 19. Aplicación web - Consultas sobre personajes Detalle

Imagen 20. Aplicación web - Buscar el número de identificador de un héroe

Imagen 21. Aplicación web - Ejemplo de consulta que devuelve una frase

“Consultas Top de apariciones”

Al seleccionar esta opción, se nos presentan otras tres opciones como ejemplifica la **Imagen 22.** De este modo estaríamos viendo los 5 personajes con más apariciones dependiendo de si son femeninos (consulta 5) si son los más poderosos (consulta 10) y si son buenos/malos/neutrales (consulta 11). Estos resultados están agrupados mediante una tabla de angular material.

Imagen 22. Aplicación web - Ejemplo de resultado agrupado en una tabla

"Consultas sobre cómics"

Estas consultas se distinguen por su temática de comics, y te ofrecen buscar el cómic con los héroes más fuertes (consulta 13) o hacer una búsqueda por nombre de cómic en un mat-input donde escribir el nombre del mismo como muestra la **Imagen 23**

Imagen 23. Aplicación web - Ejemplo de consulta tipo búsqueda

Como parte adicional en todas las consultas cuyos resultados requieren un tiempo de espera (como por ejemplo cargar los 4038 heroes) se ha añadido un mat-spinner para dar un feedback positivo de que la consulta se está ejecutando como muestra la **Imagen 24.**

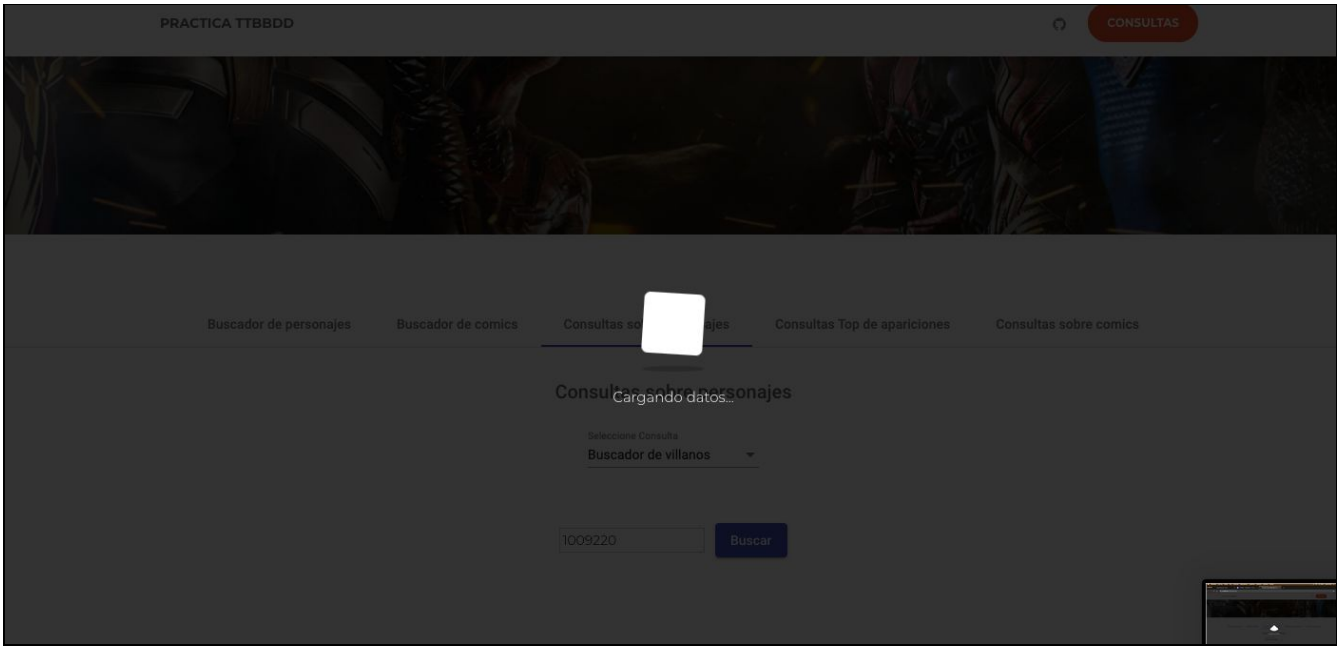

Imagen 24. Aplicación web - Ejemplo de feedback usuario

Funcionamiento y estructura del código:

El front al estar hecho en Angular 9, tiene una arquitectura hecha por componentes en la que cada componente tiene su vista HTML, sus estilos CSS y el código que describe su funcionamiento que está hecho en TypeScript.

El proceso que se sigue para visualizar todos los datos es:

1. Un evento desencadena la ejecución de una función para mostrar los datos, este evento puede ser, por ejemplo, la carga inicial de la propia página o una acción de selección hecha por el usuario.
2. Se llama con una llamada Http a un endpoint del backend para recuperar los datos deseados.
3. Se recibe respuesta del backend y se procede a mostrar los datos que se han recuperado de la base de datos.

URLS

Para poder probar la aplicación y ver las consultas realizadas es necesario acceder a la url de la APP Web, dicha APP tiene incluida la Url de la BBDD de Mongo que usamos para almacenar todas las colecciones.

URL BBDD MONGODB

Enlaces:

mongodb+srv://mongo:mongo@ttbbdd-fgu5s.mongodb.net/grupo6?retryWrites=true&w=majority

Log in to your account

- **User**: practicattbbdd@outlook.es
- **Password**:team102020

URL APP WEB CONSULTAS

http://ec2-15-236-90-35.eu-west-3.compute.amazonaws.com:3000/#/home

CÓDIGO EN GITHUB

https://github.com/daniel097541/TTBBDD

BIBLIOGRAFÍA

MONGODB OFFICIAL DOCUMENTATION

MONGODB CLOUD

SWAGGER OFFICIAL DOCUMENTATION

NODE.JS OFFICIAL DOCUMENTATION

STACKOVERFLOW

MONGOOSE

AWS-EC2

Y para todo lo demás… GOOGLE

APÉNDICE A

Consulta 1 - ¿Quién es el más gordo?

NOMBRE QUERY	¿Quién es el más gordo?	COMPLEJIDAD
DEFINICIÓN	Personaje con el mayor peso	Sencilla

CÓDIGO

```
db.characters.aggregate([
   { $group: {_id: "$$ROOT", weight: {$max: "$info.weight"}} },
   { $sort: {weight: -1} },
   { $limit: 1 }
]);
```

COMENTARIOS

El campo _id devuelve todos los datos del personaje. El peso es de 855 (unidad desconocida)

Consulta 2 - ¿Quién es el más alto?

NOMBRE QUERY	¿Quién es el más alto?	COMPLEJIDAD
DEFINICIÓN	Dame el personaje más alto de todos.	Sencilla

CÓDIGO

```
db.characters.aggregate([
   { $group: {_id: "$$ROOT", height: {$max: "$info.height"}} },
   { $sort: {height: -1} },
   { $limit: 1 }
]);
```

COMENTARIOS

El campo _id devuelve todos los datos del personaje. La altura del personaje más alto es de 975 (unidad desconocida)

Consulta 3 - ¿Quién es el héroe más poderoso?

NOMBRE QUERY	¿Quién es el héroe más poderoso?	COMPLEJIDAD
DEFINICIÓN	El héroe con más poderes	Sencilla

CÓDIGO

```
db.characters.aggregate([
   { $match: { 'info.alignment': 'good' } },
   {$project: {_id: "$$ROOT", len: {"$size": "$powers"}}},
   {$sort: {len: -1}},
   {$limit: 1}
]);
```

COMENTARIOS

El personaje héroe con más poderes

Consulta 4 - ¿Quién nos salvará del Coronavirus?

NOMBRE QUERY	¿Quién nos salvará del Coronavirus?	COMPLEJIDAD
DEFINICIÓN	Todos los personajes que tienen <cadena> en el nombre.	Sencilla

CÓDIGO

```
const regex = new RegExp(name, 'i');
db.characters.find({'name': regex})
```

COMENTARIOS

Buscador de string contenidas en el nombre de un personaje.

Consulta 5 - Las chicas más populares

NOMBRE QUERY	Las chicas más populares	COMPLEJIDAD
DEFINICIÓN	Top 10 de personajes femeninos con más apariciones	Sencilla

CÓDIGO

```
db.characters.aggregate(
   {$match: {"info.gender": {$eq: "female"}}},
   { $project: { char_name: "$name",
           numberOfAppearances: {$cond: {if: {$isArray: "$comics"}, then: {$size:
                           "$comics"}, else: 0} } } },
   {$sort: {numberOfAppearances: -1}},
   {$limit: 10}
]);
```

COMENTARIOS

Filtramos primero por género “female” para después contar el número de apariciones si tenemos la información. Después ordenamos por número de apariciones de forma descendente. Devolvemos los 10 primeros personajes.

Consulta 6 - Las chicas son guerreras

NOMBRE QUERY	Las chicas son guerreras	**COMPLEJIDAD**
DEFINICIÓN	Todos los personajes femeninos ordenadas por la cantidad de poderes que tienen.	Sencilla

CÓDIGO

```
db.characters.aggregate([
   // 1 - Filter by gender == female
   { $match: { 'info.gender': 'female'} },
   // 2 - Project to retrieve only id, name and power amount
   { $project: {
       _id: '$_id',
       name: '$name',
       powers_amount: {
           $size: { $ifNull: ['$powers', []] }
   } } },
   // 3 - Sort by amount of powers, descending
   { $sort: {'powers_amount': -1 } }
]);
```

COMENTARIOS

Una vez filtrados los personaje por género femenino, nos quedamos sólo con los campos que nos interesan usando `$project` y ordenamos en orden descendente por el número de superpoderes.

Consulta 7 - Malos héroes

NOMBRE QUERY	Malos héroes	COMPLEJIDAD
DEFINICIÓN	Todos los héroes que han sido villanos en un crossover	Sencilla

CÓDIGO

```
db.characters.aggregate([
   // 1 - Find heroes first
   { $match: { 'info.alignment': 'good' } },
   // 2 - Filter by those that have a crossover where they are villains
   { $match: { crossovers: { $elemMatch: {alignment: 'bad'} } } }
]);
```

COMENTARIOS

Con esta consulta queremos averiguar qué personajes héroes han cambiado de bando en alguna de sus apariciones cruzadas en otros cómics. **Esta consulta es muy rápida ya que en nuestro modelo decidimos incluir los crossovers como parte de la colección characters.**

Consulta 8 - El villano más listo

NOMBRE QUERY	El villano más listo	COMPLEJIDAD
DEFINICIÓN	Busca al personaje villano con más puntos de inteligencia	Sencilla

CÓDIGO

```
db.characters.aggregate([
   { $match: { 'info.alignment': 'bad' } },
   { $sort: { 'stats.intelligence': -1 } },
   { $limit: 1 },
   { $project: { _id: '$_id', name: '$name' } }
]);
```

COMENTARIOS

Esta consulta devuelve los personajes “malvados” ordenados en orden descendente por sus puntos de inteligencia.

Consulta 9 - El héroe más tonto

NOMBRE QUERY	El héroe más tonto	COMPLEJIDAD
DEFINICIÓN	Busca al personaje héroe con menos puntos de inteligencia	Sencilla

CÓDIGO

```
db.characters.aggregate([
   { $match: { 'info.alignment': 'good' } },
   { $sort: { 'stats.intelligence': 1 } },
   { $limit: 1 },
   { $project: { _id: '$_id', name: '$name' }}
]);
```

COMENTARIOS

Esta consulta devuelve ofrece los personajes “buenos” ordenados en orden ascendente por sus puntos de inteligencia.

Consulta 10 - Los más poderosos

NOMBRE QUERY	Los más poderosos	COMPLEJIDAD
DEFINICIÓN	Top 10 de personajes más poderosos	Sencilla

CÓDIGO

```
db.characters.aggregate([
   { $match: { powers: { $ne: [] } } },
   { $project: { char_name: "$name",
            numberOfPowers: { $cond: { if: { $isArray: "$powers" },
                    then: { $size: "$powers" }, else: 0} } } },
   {$sort: {numberOfPowers: -1}},
   {$limit: 10}
]);
```

COMENTARIOS

Filtramos primero por los personajes de los que tenemos información de poderes. Acumulamos la “aparición” de poderes en la variable `numberOfPowers` y luego ordenamos descendentemente y nos limitamos a las 10 primeros resultados.

Consulta 11 – Los que más aparecen según alignment

NOMBRE QUERY	Los que más aparecen según alignment	COMPLEJIDAD
DEFINICIÓN	Top 5 de personajes que más apariciones tienen separados por su alignment	Sencilla

CÓDIGO

```
db.characters.aggregate([
   {$match: {"info.alignment": {$eq: alignment}}},
   { $project: { char_name: "$name",
           numberOfAppearances: {$cond: {if: {$isArray: "$comics"},
                   then: {$size: "$comics"}, else: 0} } } },
   { $sort: { numberOfAppearances: -1 } },
   { $limit: 5 }
]);
```

COMENTARIOS

Con esta consulta queríamos hacer un buscador para hallar el top 5 de apariciones según su naturaleza.

Consulta 12 –¿Quién es el héroe con mejores estadísticas del cómic?

NOMBRE QUERY	¿Quién es el héroe con mejores estadísticas del cómic?	COMPLEJIDAD
DEFINICIÓN	Dado un cómic, busca el héroe con mejores estadísticas	Alta

CÓDIGO

```
db.comics.aggregate([
   // 1 - Filter by comic id
   { $match: { _id: comicId } },
   // 2 - Join characters that appear in this comic
   { $lookup: {
            from: "characters",
            localField: "_id",
            foreignField: "comics",
            as: "comic_characters" } },
   // 3 - Unwind characters to iterate over the array
   { $unwind: '$comic_characters '},
   // 4 - Replace root to return a character instead of a comic
   { $replaceRoot: { newRoot: '$comic_characters' } },
   // 5 - Filter characters that are heroes (alignment = good)
   { $match: { 'info.alignment': 'good' } },
   // 6 - Make a projection to not return all fields of the character and get the total stats
   { $project: {
            _id: '$$ROOT._id',
            name: '$$ROOT.name',
            info: '$$ROOT.info',
            stats: '$$ROOT.stats',
            total_stats: {
                $add: ['$stats.combat', '$stats.durability', '$stats.intelligence',
                    '$stats.power', '$stats.speed', '$stats.strength']
            } } },
   // 7 - Sort by total stats descending
   { $sort: { total_stats: -1 } },
   // 8 - Get the best one
   { $limit: 1 }
]);
```

COMENTARIOS

Filtramos primero por comicID y posteriormente hacemos el “join” con la colección characters. Necesitamos hacer una proyección para obtener todos los personajes y sacar las `total_stats`

Consulta 13 - El comic de los vigoréxicos

NOMBRE QUERY	El comic de los vigoréxicos	COMPLEJIDAD
DEFINICIÓN	De una **muestra aleatoria de 1000 cómics**, obtiene el cómic que tiene los héroes más fuertes	Alta

CÓDIGO

```
db.comics.aggregate([
   { $sample: {  size: 1000 } },
   { $lookup: {
            from: 'characters',
            let: {
                comic_id: '$_id',
                comic_name: '$name' },
            as: 'total_comic_str',
            pipeline: [
                { $match: {
                        $expr: {
                            $and: [
                                // the character is from this comic
                                {$in: ['$$comic_id', '$comics']},
                                // the character is a hero
                                {$eq: ['$info.alignment', 'good']},
                                {$ne: [null, '$stats.strength']}
                            ] } } },
                { $group: {  // group by comic id and accumulate strength of each hero
                        _id: '$$comic_name',
                        total_str: {  $sum: '$stats.strength' }
                } } ]
   } },
   { $unwind: { path: '$total_comic_str', preserveNullAndEmptyArrays: false } },
   { $replaceRoot: { newRoot: '$total_comic_str'} },
   { $sort: { total_comic_str: -1 } },
   { $limit: 1 }
]);
```

COMENTARIOS

Esta query tuvimos que adaptar a un sample del 1000 cómics porque tardaba demasiado. Este tipo de consultas quizás requeriría de un cambio en la "estructura" de la base de datos para ser más eficiente en tiempo.
Esta consulta necesito hacer un "join" entre las dos colecciones para poder acceder a los datos de los personajes que están en ese cómic y aculumar en la variable `total_str` el número de

Consulta 14 - Buscador de villanos

NOMBRE QUERY	Buscador de villanos	COMPLEJIDAD
DEFINICIÓN	Todos los villanos que comparten cómic con un personaje	Alta

CÓDIGO

```
db.characters.aggregate([
   { $match: { _id: characterId } },
   { $unwind: '$comics' },
   { $lookup: {
            from: "characters",
            as: "comic_characters",
            let: {
                comic_id: '$comics',
                me: '$_id'
            },
            pipeline: [
                { $match: {
                        $expr: {
                            $and: [
                                { $in: ['$$comic_id', '$comics'] },
                                { $eq: ['$info.alignment', 'bad'] },
                                { $ne: ['$_id', '$$me'] } ]
                } } },
                { $project: { _id: '$_id',  name: '$name' } }
            ]
        } },
   { $unwind: '$comic_characters' },
   { $group: {
            _id: '$_id',
            villains: { $addToSet: '$comic_characters' } }
   }
]);
```

COMENTARIOS

Dado un personaje (búsqueda por id: 1009393) buscamos todos los personajes villanos que comparten cómic con dicho personaje. En esta consulta también es necesario hacer un “join” entre las dos colecciones. Hemos tenido en cuenta que si el personaje que buscamos es villano, no se cuente a sí mismo `$ne: ['$_id', '$$me']`