

# EJS SWEB

EJS SWEB.....	1
JSON EJS.....	2
MongoDB Ejercicios .....	7
ENTREGA 1 .....	7
ENTREGA 2 .....	8
Entrega 3 – EJ3.....	11
PARCIAL .....	12
<b>Si solo te piden el número total combinado (todas las propiedades que tienen 2, 3, 4 o 5 camas):.....</b>	<b>14</b>
Apartado: Usa una variable de entorno para configurar el puerto.....	16
🔍 Paso 1: Mira qué campos permite OpenAPI .....	17
🔧 Paso 2: Modifica tu <code>routes/book.js</code> .....	17
🎯 Resultado esperado .....	18
<b>1. MODIFICACIÓN DEL CÓDIGO EN <code>routes/book.js</code> .....</b>	<b>19</b>
✅ <b>2. MODIFICACIÓN DEL OpenAPI (<code>library.schema.yaml</code>) .....</b>	<b>20</b>
✅ <b>EJEMPLO DE SALIDA ESPERADA AL REINICIAR EL SERVER .....</b>	<b>20</b>
✅ Paso 1: Modificar la ruta en <code>routes/book.js</code> .....	21
✅ Paso 2: Añadir la definición a <code>library.schema.yaml</code> .....	23
🖋️ ¿Cómo comprobarlo? .....	23
DTD EJS .....	23
DTD Ejercicio 1 .....	23
DTD Ejercicio 2.....	24
2. 📄 Apartado: No se puede validar el cuerpo de la petición .....	25
3. 🔄 Apartado: La API no sigue lo que dice el <code>schema.yaml</code> de OpenAPI ..	26
4. 🔒 Apartado: Añade CORS para que la API sea accesible desde el frontend	26
5. 🔍 Apartado: Necesitas hacer logs de peticiones.....	26
✅ 1. Instala apiDoc (una sola vez).....	27
✅ 2. Estructura básica de comentarios apiDoc .....	27
✅ 3. Genera la documentación .....	27
✅ 4. Ver la documentación en el navegador.....	28
✅ APARTADO 6: Añadir POST /book con validación.....	28
1. <code>routes/book.js</code> .....	28

2. library.schema.yaml.....	29
✓ APARTADO 7: Añadir previous o count total en GET /book.....	29
1. routes/book.js (modifica el GET / existente) .....	29
✓ APARTADO 8: Filtro por autor .....	30
1. En el mismo GET /, añade justo después de let query = {} lo siguiente:.....	30
* Posibles comandos para probar: .....	30
✓ Apartado 13: Validar API Key (x-api-key) .....	30
1. Crea archivo middleware/auth.js .....	30
2. Añade al .env .....	31
3. Aplica middleware en routes/book.js .....	31
✓ Apartado 14: Validar límite con MAX_RESULTS .....	31
1. En routes/book.js, modifica esta parte de tu GET /book.....	31
✓ Apartado 15: Añadir self y collection en HATEOAS.....	31
1. En GET /book, añade:.....	31

## JSON EJS

Crea un JSON que sea válido con el siguiente JSON Schema:

**Pag 37:**

```
{
  "$schema": "https://json-schema.org/draft/2019-09/schema#",
  "$id": "http://my-paintings-api.com/schemas/painting-schema.json",
  "type": "object",
  "title": "Painting",
  "description": "Painting information",
  "additionalProperties": true,
  "required": ["name", "artist", "dimension", "description", "tags"],
  "properties": {
    "name": {
      "type": "string",
      "description": "Painting name"
    },
    "artist": {
      "type": "string",
      "maxLength": 50,
      "description": "Name of the artist"
    },
    "description": {
      "type": ["string", "null"],
```

```

    "description": "Painting description"
  },
  "dimension": { "$ref": "#/$defs/dimension" },
  "tags": {
    "type": "array",
    "items": { "$ref": "#/$defs/tag" }
  }
},
"$defs": {
  "tag": {
    "type": "string",
    "enum": ["oil", "watercolor", "digital", "famous"]
  },
  "dimension": {
    "type": "object",
    "title": "Painting dimension",
    "description": "Describes the dimension of a painting in cm",
    "additionalProperties": true,
    "required": ["width", "height"],
    "properties": {
      "width": { "type": "number", "description": "Width of the product",
"minimum": 1 },
      "height": { "type": "number", "description": "Height of the product",
"minimum": 1 }
    }
  }
}
}

```

### Solución

```

{
  "name": "prueba",
  "artist": "Bisbal",
  "description": "Caca",
  "dimension": { "width": 1.1, "height": 2.2 },
  "tags": "oil"
}

```

**EJ PAG 38:** Crea un JSON que sea válido con el siguiente JSON Schema:

//JSON enunciado

```

{
  "squadName": "Super hero squad",
  "homeTown": "Metro City",
  "formed": 2016,
  "secretBase": "Super tower",
  "active": true,

```

```

"members": [
  {
    "name": "Molecule Man",
    "age": 29,
    "secretIdentity": "Dan Jukes",
    "powers": ["Radiation resistance", "Turning tiny", "Radiation blast"]
  },
  {
    "name": "Madame Uppercut",
    "age": 39,
    "secretIdentity": "Jane Wilson",
    "powers": ["Million tonne punch", "Damage resistance"]
  }
]
}

```

**SOLUCIÓN: //JSON Schema válido**

```

{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "type": "object",
  "properties": {
    "squadName": { "type": "string" },
    "homeTown": { "type": "string" },
    "formed": { "type": "integer" },
    "secretBase": { "type": "string" },
    "active": { "type": "boolean" },
    "members": {
      "type": "array",
      "items": {
        "type": "object",
        "properties": {
          "name": { "type": "string" },
          "age": { "type": "integer", "minimum": 0 },
          "secretIdentity": { "type": "string" },
          "powers": {
            "type": "array",
            "items": { "type": "string" }
          }
        }
      }
    },
    "required": ["name", "age", "secretIdentity", "powers"]
  }
}
{
  "required": ["squadName", "homeTown", "formed", "secretBase", "active", "members"]
}

```

```

{
  "$schema": "https://json-schema.org/draft/2020-12/schema",

```

```
"title": "Superhero Squad",
"type": "object",
"required": ["squadName", "homeTown", "formed", "secretBase", "active",
"members"],
"properties": {
  "squadName": {
    "type": "string",
    "description": "Name of the squad"
  },
  "homeTown": {
    "type": "string",
    "description": "City of origin"
  },
  "formed": {
    "type": "integer",
    "description": "Year of formation"
  },
  "secretBase": {
    "type": "string"
  },
  "active": {
    "type": "boolean"
  },
  "members": {
    "type": "array",
    "items": {
      "type": "object",
      "required": ["name", "age", "secretIdentity", "powers"],
      "properties": {
        "name": {
          "type": "string"
        },
        "age": {
          "type": "integer",
          "minimum": 0
        },
        "secretIdentity": {
          "type": "string"
        },
        "powers": {
          "type": "array",
          "items": {
            "type": "string"
          }
        }
      }
    }
  }
}
```

### EJ39:

//JSON de película

```
{
  "title": "Inception",
  "year": 2010,
  "director": "Christopher Nolan",
  "genre": ["Science Fiction", "Action"],
  "rating": 8.8,
  "actors": [
    {
      "name": "Leonardo DiCaprio",
      "role": "Dom Cobb"
    },
    {
      "name": "Joseph Gordon-Levitt",
      "role": "Arthur"
    }
  ]
}
```

// JSON Schema para validar el JSON

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "type": "object",
  "properties": {
    "title": { "type": "string" },
    "year": { "type": "integer", "minimum": 1888 },
    "director": { "type": "string" },
    "genre": {
      "type": "array",
      "items": { "type": "string" }
    },
    "rating": { "type": "number", "minimum": 0, "maximum": 10 },
    "actors": {
      "type": "array",
      "items": {
        "type": "object",
        "properties": {
          "name": { "type": "string" },
          "role": { "type": "string" }
        },
        "required": ["name", "role"]
      }
    }
  },
  "required": ["title", "year", "director", "genre", "rating", "actors"]
}
```

# MongoDB Ejercicios

## ENTREGA 1

1. En `sample_training.zips` ¿Cuántas colecciones tienen menos de 1000 personas en el campo `pop`? (sol. 8065)

(base) MacBook-Pro-Pere:mongodb-sample-dataset maripele\$ mongosh

test> show dbs

sample\_training> show tables

companies

grades

inspections

posts

routes

stories

trips

tweets

zips

sample\_training> db.zips.find({ pop: { \$lt: 1000 } }).count()

Rdo: 8065

2. En `sample_training.trips` ¿Cuál es la diferencia entre la gente que nació en 1998 y la que nació después de 1998? (sol 6)

Solución profe (+ FACIL):

Muy fácil hago consulta de una `db.trips.find({ "birth year": 1998 }).count(); //12`

Y luego hace la otra independiente restando.

`db.trips.find({ "birth year": 1998 }).count() - db.trips.find({ "birth year": { $gt: 1998 } }).count()`

Si le aplicamos el Valor absoluto:

`Math.abs(  
 db.trips.find({ "birth year": 1998 }).count() -  
 db.trips.find({ "birth year": { $gt: 1998 } }).count()  
)`

3. En `sample_training.routes` ¿Cuántas rutas tienen al menos una parada? (sol. 11)

sample\_training> db.routes.find({ stops: { \$gte: 1 } }).count()

11

sample\_training>

4. En `sample_training.inspections` ¿Cuántos negocios tienen un resultado de inspección "Out of Business" y pertenecen al sector "Home Improvement Contractor - 100"? (sol. 4)

```
sample_training> db.inspections.find({ result: "Out of Business", sector: "Home Improvement Contractor - 100" }).count()
```

4

```
sample_training>
```

5. En `sample_training.inspections` ¿Cuántos documentos hay con fecha de inspección "Feb 20 2015" o "Feb 21 2015" y cuyo sector no sea "Cigarette Retail Dealer - 127"? (sol. 204)

El \$in se comporta como un or...

```
db.inspections.find({
  date: { $in: ["Feb 20 2015", "Feb 21 2015"] },
  sector: { $ne: "Cigarette Retail Dealer - 127" }
}).count()
```

Otra alternativa mia propia:

```
db.inspections.find({date:"Feb 20 2015", sector:{ $ne:"Cigarette Retail Dealer - 127"}}).count() + db.inspections.find({date:"Feb 21 2015", sector:{ $ne:"Cigarette Retail Dealer - 127"}}).count()
```

204

## ENTREGA 2

MongoDB – EJ2

1. En `sample_training.companies`, ¿cuántas empresas tienen más empleados que el año en el que se fundaron? (sol. 324)

EMPLEADOS > AÑO DE FUNDACIÓN

```
db.companies.find({
  $expr: { $gt: ["$number_of_employees", "$founded_year"] }
}).count()
```

2. En `sample_training.companies`, ¿en cuántas empresas coinciden su `permalink` con su `twitter_username`? (sol. 1299)

EQUIVALENTES

```
db.companies.find({
  $expr: { $eq: ["$permalink", "$twitter_username"] }
}).count()
```



3. En sample\_airbnb.listingsAndReviews, ¿cuál es el nombre del alojamiento en el que pueden estar más de 6 personas alojadas y tiene exactamente 50 reviews? (sol. Sunset Beach Lodge Retreat)

```
db.listingsAndReviews.find({ accommodates: { $gt: 6 }, number_of_reviews: 50 }, { name: 1, _id: 0 })
```

**Se quita el id para que solo salga el nombre**

4. En sample\_airbnb.listingsAndReviews, ¿cuántos documentos tienen el "property\_type" "House" e incluyen "Changing table" como una de las "amenities"? (sol. 11)

```
db.listingsAndReviews.find({property_type: "House", amenities: "Changing table" }).count()
```

5. En sample\_training.companies, ¿Cuántas empresas tienen oficinas en Seattle? (sol. 117)

```
offices: [
  {
    description: '',
    address1: '710 - 2nd Avenue',
    address2: 'Suite 1100',
    zip_code: '98104',
    city: 'Seattle',
```

Offices es un array que tiene el atributo city

```
db.companies.find({"offices.city": "Seattle" }).count()
```

6. En sample\_training.companies, haga una query que devuelva únicamente el nombre de las empresas que tengan exactamente 8 "funding\_rounds"

```
funding_rounds: [
  {
    id: 888,
    round_code: 'a',
    source_url: 'http://seattlepi.nwsources.com/business/246734_wiki02.html',
    source_description: '',
    raised_amount: 5250000,
    raised_currency_code: 'USD',
    funded_year: 2005,
    funded_month: 10,
    funded_day: 1,
    investments: [
      {
        company: null,
        financial_org: {
          name: 'Frazier Technology Ventures',
          permalink: 'frazier-technology-ventures'
        },
        person: null
      },
      {
        company: null,
        financial_org: { name: 'Trinity Ventures', permalink: 'trinity-ventures' },
        person: null
      }
    ]
  }
]
```

Size: 8 objetos en total dentro de ese atributo.

```
db.companies.find({funding_rounds: { $size: 8 }}, { name: 1 })
```

 //tmb podemos quitar\_id=0

7. En sample\_training.trips, ¿cuántos viajes empiezan en estaciones que están al oeste de la longitud -74? (sol. 1928)

Nota 1: Hacia el oeste la longitud decrece

Nota 2: el formato es <field\_name>: [ <longitud>, <latitud> ]

```
sample_training> db.trips.findOne()
{
  _id: ObjectId('572bb8222b288919b68abf5c'),
  tripduration: 589,
  'start station id': 489,
  'start station name': '10 Ave & W 28 St',
  'end station id': 284,
  'end station name': 'Greenwich Ave & 8 Ave',
  bikeid: 21997,
  usertype: 'Subscriber',
  'birth year': 1982,
  gender: 2,
  'start station location': { type: 'Point', coordinates: [ -74.00176802, 40.75066386 ] },
  'end station location': { type: 'Point', coordinates: [ -74.0026376103, 40.7390169121 ] },
  'start time': ISODate('2016-01-01T00:00:48.000Z'),
  'stop time': ISODate('2016-01-01T00:10:37.000Z')
}
sample_training>
```

Coordinates es un atributo interno que accedemos con el . y luego el 0 (1ra posición array)

```
db.trips.find({"start station location.coordinates.0": { $lt: -74 }}).count()
```

8. En sample\_training.inspections, ¿cuántas inspecciones se llevaron a cabo en la ciudad de "NEW YORK"? (sol. 18279)

```
sample_training> db.inspections.findOne()
{
  _id: ObjectId('56d61033a378eccde8a83551'),
  id: '10084-2015-ENF0',
  certificate_number: 9278914,
  business_name: 'MICHAEL GOMEZ RANGHALL',
  date: 'Feb 10 2015',
  result: 'No Violation Issued',
  sector: 'Locksmith - 062',
  address: { city: 'QUEENS VLG', zip: 11427, street: '214TH ST', number: 8823 }
}
```

```
db.inspections.find({"address.city": "NEW YORK"}).count()
```

9. En sample\_airbnb.listingsAndReviews, haga una query que devuelva el nombre y la dirección de los alojamientos que tengan "Internet" como primer elemento de "amenities"

```
amenities: [
  'Internet',
  'Wifi',
  'Air conditioning',
  'Kitchen',
  'Buzzer/wireless intercom',
  'Heating',
  'Smoke detector',
  'Carbon monoxide detector',
  'Essentials',
  'Lock on bedroom door'
],
```

Solución profe mas corta :

`db.listingsAndReviews.find({"amenities.0":"Internet"},{name:1,address:1,_id:0})` solo se puede mezclar poner un 0 con ID.

## Entrega 3 – EJ3

### 1. En sample\_airbnb.listingsAndReviews, ¿qué "room types" existen?

`$group`. Sirve para **agrupar documentos** según el valor de un campo.

```
db.listingsAndReviews.aggregate([
  { $group: { _id: "$room_type" } }
])
```

**Más facil:** (para obtener los "room\_type" únicos(sin repetición o duplicados))  
`db.listingsAndReviews.distinct("room_type")`

### 2. En sample\_training.companies, haga una query que devuelva el nombre y el año en el que se fundaron las 5 compañías más antiguas.

Muestra los años mas bajos de todo (antiguo 1800 en vez de 2000) (de mayor a menor sort 1.

El Project muestra solo esos campo y los demás no.

```
db.companies.aggregate([ { $match: { founded_year: { $ne: null } } }, { $project: {
name: 1, founded_year: 1, _id: 0 } },
  { $sort: { founded_year: 1 } }, { $limit: 5 } a esto le falta el not equal...
])
```

`db.companies`

```
.find(
  { founded_year: { $ne: null } }, // evita resultados nulos
  { name: 1, founded_year: 1, _id: 0 } // proyección: solo nombre y año
)
.sort({ founded_year: 1 }) // más antiguos primero
.limit(5) // solo los 5 primeros
```

### 3. En sample\_training.trips, ¿en qué año nació el ciclista más joven? (sol. 1999)

No es del todo correcta...

```
db.trips.aggregate([
  { $group: { _id: null, maxBirthYear: { $max: "$birth year" } } }
])
```

```
db.trips.find(
  {
    "birth year": { $ne: null, $ne: "" }, // excluye null y ""
    "birth year": { $type: "number" } // asegura que es número
  }
)
```

```

    },
    {
      "birth year": 1, _id: 0
    }
  ).sort({ "birth year": -1 }).limit(1) //(-1 para que saque el más joven o año más alto)

```

## PARCIAL

**Apartado 1.** En la colección `listingAndReviews` indique el/los nombre(s) del alojamiento con más reviews.

```
db.listingsAndReviews.find({}, { name: 1, number_of_reviews: 0, _id: 0 }).sort({
number_of_reviews: -1 }).limit(2) //he puesto 2 para comprobar si hay varios
iguales
```

Una vez descubierto que no tienen el mismo número pues dejo solo el nombre borrando el número de reviews:

```
sample_airbnb> db.listingsAndReviews.find({}, { name: 1, _id: 0 }).sort({
number_of_reviews: -1 }).limit(2)
```

🔍 ¿Qué hace esta consulta?

`{}` → no filtra nada, busca en todos los documentos.

`{ name: 1, number_of_reviews: 1, _id: 0 }` → muestra solo el nombre y número de reviews.

`.sort({ number_of_reviews: -1 })` → ordena de mayor a menor por número de reviews.

`.limit(1)` → devuelve solo el alojamiento con más reviews.

**Apartado 2.** En la colección `listingAndReviews` indique el/los nombre(s) del alojamiento con más amenities.

```

amenities: [
  'Internet',
  'Wifi',
  'Air conditioning',
  'Kitchen',
  'Buzzer/wireless intercom',
  'Heating',
  'Smoke detector',
  'Carbon monoxide detector',
  'Essentials',
  'Lock on bedroom door'
],

```

```

db.listingsAndReviews.aggregate([
  { $project: {
    name: 1,
    totalAmenities: { $size: "$amenities" }
  }},
  { $sort: { totalAmenities: -1 } },
  { $limit: 2 }
])

```

🔍 ¿Qué hace esta consulta?

\$project: crea un campo calculado totalAmenities con el número de elementos en el array amenities.

\$sort: ordena de mayor a menor por totalAmenities.

\$limit: devuelve solo el alojamiento con más amenities.

**Apartado 3. En la colección listingsAndReviews indique para cada tipo de property\_type el número de alojamientos de ese tipo.**

```

db.listingsAndReviews.aggregate([
  {
    $group: {
      _id: "$property_type", //aquí se dice el campo por el que se agrupan
      total: { $sum: 1 } //cuenta 1x1 cuantos doc hay por cada grupo de property type.
    }
  },
  {
    $sort: { total: -1 }
  }
])

```

🔍 Explicación:

\$group: agrupa los documentos por el campo property\_type.

`_id: "$property_type"`: cada grupo será un tipo de propiedad (como "Apartment", "House", etc.).

`total: { $sum: 1 }`: cuenta cuántos alojamientos hay en cada grupo.

**Apartado 4. En la colección `listingAndReviews` indique el número de alojamientos que tienen 2, 3, 4 o 5 beds.**

```
db.listingsAndReviews.aggregate([
  { $match: { beds: { $in: [2, 3, 4, 5] } } },
  { $group: { _id: "$beds", total: { $sum: 1 } } },
  { $sort: { _id: 1 } } // Orden opcional: por número de camas
])
```

**Si solo te piden el número total combinado (todas las propiedades que tienen 2, 3, 4 o 5 camas):**

```
db.listingsAndReviews.find({ beds: { $in: [2, 3, 4, 5] } }).count()
```

#### Nota:

`.find()` solo sirve para:

- Buscar documentos.
- Filtrar campos con `.projection`.
- Ordenar con `.sort()`.
- Limitar con `.limit()`.

Pero **no te permite hacer cosas más complejas** como:

- Agrupar (`$group`)
- Contar (`$count`)
- Calcular promedios o sumas (`$avg`, `$sum`)
- Reestructurar campos (`$project`)
- Hacer joins con otras colecciones (`$lookup`)

## OPENAPI

### Problema 1

La carpeta api contiene un servidor web que arranca una API REST que está incompleta. La especificación OpenAPI está en schema/library.schema.yaml. Tenga en cuenta las siguientes consideraciones:

- Vamos a dejar que la base de datos gestione las id, con lo que usaremos `_id` como nuestras id tratándola como un string.
- Por simplicidad no se permite editar la información de los libros. Complete los apartados que aparecen a continuación.

**Apartado 1. Actualmente la API no se está ejecutando en la ruta que está especificada en el documento OpenAPI. Modifique el servidor para que coincidan.**

El archivo `library.schema.yaml` indica (por convención OpenAPI) en qué ruta debería estar expuesta tu API. Observando su contenido, seguramente aparece algo

```
servers:  
  - url: /api/v2
```


Esto significa que **todas las rutas de la API deben estar colgando bajo `/api/v2`**

**Por lo que:**

Solución en `app.js`

Modifica este fragmento de código:

```
var bookRouter = require('./routes/book');  
var usersRouter = require('./routes/users');  
  
app.use('/books', bookRouter);  
app.use('/users', usersRouter);
```

 Por esto:

```
app.use('/api/books', bookRouter);  
app.use('/api/users', usersRouter);
```

Esto asegura que las rutas coincidan con `/api/books` y `/api/users`, como define el esquema OpenAPI (`/api/v2` es el prefijo base).

En `conn.js` se ve definido:

```
const connectionString = process.env.MONGODB_URI;
```

Añadimos en app.js la librería con el mismo nombre no sirve (MONGO\_URI) tiene que coincidir :

```
# Database (ERA EL PROBLEMA)
MONGODB_URI=mongodb://127.0.0.1/sw2
```

**Y si eso, añadir modulo si hace falta en app y demás:**

```
require('dotenv').config();
```

## APARTADO EXTRA:

*Apartado: Usa una variable de entorno para configurar el puerto*

Ejemplo:

```
const PORT = process.env.PORT || 3000;
```

**Apartado 2. Actualmente la ruta GET /book está devolviendo la información completa de cada libro, pero eso no debería ser así. Modifique el servidor para que de cada libro se devuelva sólo la información especificada en el documento OpenAPI.**

```
mongodb://127.0.0.1/sw2
Successfully connected to database
GET /book 404 65.186 ms - 1073
GET /api/book 404 8.796 ms - 1073
GET /api/v2/book 404 12.397 ms - 1073
```

Hay que meter en el dotenv

```
BASE_URI=/api # Las rutas están montadas bajo /api
```

Esto por si acaso:

```
PORT=3000
MAX_RESULTS=5
```

**Ahora si funciona:**

<http://localhost:3000/api/book>

Ya que en el app.js:

estás haciendo algo como esto:

```
app.use('/', indexRouter);
app.use(process.env.BASE_URI + '/book', booksRouter);
```



Lo cual **monta correctamente la ruta GET /api/book**,

Ahora:

Que la ruta GET /book devuelva **solo los campos definidos** en `library.schema.yaml`, **no toda la información completa** de cada libro.

---

## Paso 1: Mira qué campos permite OpenAPI

Abre `library.schema.yaml`, y localiza la definición del objeto `Book`. Verás en **schemas** algo como:

```
Book:
  type: object
  properties:
    _id:
      type: string
    title:
      type: string
    author:
      type: string
    published:
      type: string
      format: date
    pages:
      type: integer
```

Eso significa que **solo debes devolver estos campos**:

- `_id`
  - `title`
  - `author`
  - `published`
  - `pages`
- 

## Paso 2: Modifica tu `routes/book.js`

Edita tu ruta GET /book para que haga una **proyección** de solo esos campos.

Fichero origen:

```
//getBooks()
router.get('/', async (req, res) => {
  let limit = MAX_RESULTS;
  if (req.query.limit){
    limit = Math.min(parseInt(req.query.limit), MAX_RESULTS);
  }
}
```

```

let next = req.query.next;
let query = {}
if (next){
  query = {_id: {$lt: new ObjectId(next)}}
}
const dbConnect = dbo.getDb();
let results = await dbConnect
  .collection(COLLECTION)
  .find(query)
  .sort({_id: -1})
  .limit(limit)
  .toArray()
  .catch(err => res.status(400).send('Error searching for books'));
next = results.length == limit ? results[results.length - 1]._id : null;
res.json({results, next}).status(200);
});

```

**Proyección** de solo esos campos.

```

let results = await dbConnect
  .collection(COLLECTION)
  .find(query, { projection: { _id: 1, title: 1, author: 1, published: 1, pages: 1 } }) // 👉
  .sort({_id: -1})
  .limit(limit)
  .toArray()

```

## Resultado esperado

Al ir a <http://localhost:3000/api/book>, deberías ver solo objetos como este:

```

{
  "_id": "660bce7e96bc7d6563d627f8",
  "title": "Learning JavaScript Design Patterns",
  "author": "Addy Osmani",
  "published": "2012-08-30T00:00:00.000Z",
  "pages": 254
}

```

**Apartado 3. Queremos hacer nuestra API restful y para eso nos falta una parte muy importante, HATEOAS. Vamos a empezar a implementarlo en alguna de las rutas, pero no queremos modificar los datos que tenemos en la base de datos.**

En GET /book añade a cada libro del array results un atributo link que enlace a la ruta completa de ese libro: /book/{id} De forma que por ejemplo se devuelva lo siguiente (por simplicidad sólo se muestra un libro en los resultados y puede ser que la ruta no sea correcta del todo):

```
{
  "results": [
    {
```

```
      "_id": "646332b5b3767c0bcb5d4b3b",
      "title": "Speaking JavaScript",
      "author": "Axel Rauschmayer",
      "link": "localhost:3000/api/book/646332b5b3767c0bcb5d4b3b"
    }
  ],
  "next": null
}
```

Modifica el archivo OpenAPI para tener en cuenta esta modificación.

## 1. MODIFICACIÓN DEL CÓDIGO EN `routes/book.js`

Después de obtener los resultados, recórrelos y añade la propiedad `link` a cada libro en la api. Puedes usar `req.get('host')` y `req.baseUrl` para construir la URL correctamente (sin hardcodear `localhost:3000`).

🔧 Sustituye este bloque:

```
res.json({results, next}).status(200);
```

🧑 Por esto:

```
// Añadir campo 'link' a cada libro
results = results.map(book => ({
  ...book,
  link:
`${req.protocol}://${req.get('host')}${req.baseUrl}/${book._id}`
}));

res.status(200).json({ results, next });
```

Esto devolverá cada libro con un atributo `link` como:

`http://localhost:3000/api/book/646332b5b3767c0bcb5d4b3b`

Recorre cada `book` del array `results` y:

1. Usa `...book` para **copiar todos los datos del libro** (como título, autor, etc.).

2. Añade un nuevo campo `link`, que contiene la URL HATEOAS del estilo:

`http://localhost:3000/api/book/ID_DEL_LIBRO`

🔧 Se construye así:

- `req.protocol` → `http` o `https`
- `req.get('host')` → Ej: `localhost:3000`
- `req.baseUrl` → Si tu ruta es `/api/book`, aquí será `/api/book`
- `/${book._id}` → Añade el ID del libro al final

---

## ✅ 2. MODIFICACIÓN DEL OpenAPI

(`library.schema.yaml`)

Edita la definición del objeto "libro" (`BooksArray` o similar) y añade el campo `link`. Ejemplo:

```
BooksArray:
  type: array
  items:
    type: object
    properties:
      _id:
        type: string
      title:
        type: string
      author:
        type: string
      published:
        type: string
      pages:
        type: integer
      link:
        type: string
        description: "Enlace directo al recurso indiv.. de este libro"
    required:
      - _id
      - title
      - author
      - link
```

---

## ✅ EJEMPLO DE SALIDA ESPERADA AL REINICIAR EL SERVER

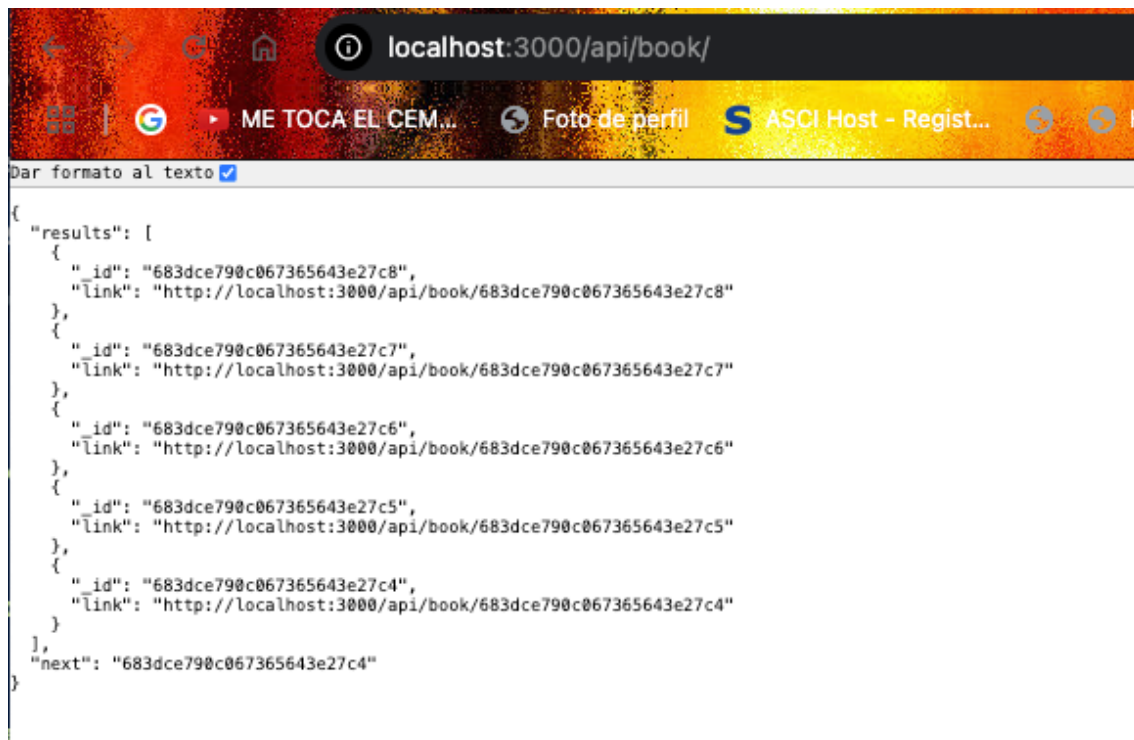
```
{
  "results": [
    {
      "_id": "646332b5b3767c0bcb5d4b3b",
      "title": "Speaking JavaScript",

```

```

      "author": "Axel Rauschmayer",
      "link":
"http://localhost:3000/api/book/646332b5b3767c0bcb5d4b3b"
    }
  ],
  "next": null
}

```



**Apartado 4.** En la ruta **DELETE /book/{id}** no se están aplicando todas las respuestas definidas en la especificación OpenAPI. Modifique el servidor para que se tengan en cuenta todos los casos definidos.

Para resolver el **Apartado 4**, necesitas **modificar la ruta DELETE /book/:id** para que cumpla con **todas las respuestas HTTP** especificadas en `library.schema.yaml`, que normalmente incluyen:

- 204 No Content si se elimina correctamente.
- 404 Not Found si el libro no existe.
- 400 Bad Request si el `id` no tiene un formato válido.

### ✓ Paso 1: Modificar la ruta en `routes/book.js`

Tenía esto:

```

//deleteBookById()
router.delete('/:id', async (req, res) => {
  const query = { _id: new ObjectId(req.params.id) };
  const dbConnect = dbo.getDb();

```

```
let result = await dbConnect
  .collection(COLLECTION)
  .deleteOne(query);
res.status(200).send(result);
});
```

Ahora mismo **especifica solo códigos 200 y 400**, pero **no contempla 404** (recurso no encontrado), lo cual sería más completo.

- **Tu código de servidor responde siempre con 200 OK**, incluso si no se borró ningún libro (es decir, el ID era válido, pero no existía).

Cuando haces una petición como:

```
DELETE /api/book/11111
```

El backend intenta hacer:

```
new ObjectId('11111')
```

**⚠ Pero '11111' no es un ObjectId válido** (debe tener exactamente 24 caracteres hexadecimales), y entonces Mongo lanza este error:

```
BSONError: Argument passed in must be a string of 12 bytes or a string
of 24 hex characters or an integer
```

Nuevo código

```
//deleteBookById()
router.delete('/:id', async (req, res) => {
  //const query = { _id: new ObjectId(req.params.id) };
  const id = req.params.id;

  // ✅ Validar que el ID sea un ObjectId válido
  if (!ObjectId.isValid(id)) {
    return res.status(400).send({ error: 'Invalid book ID' });
  }

  const query = { _id: new ObjectId(id) };
  const dbConnect = dbo.getDb();
  let result = await dbConnect
    .collection(COLLECTION)
    .deleteOne(query);

  // ✅ Comprobar si se eliminó algo
```

```
if (result.deletedCount === 0) {
  return res.status(404).send({ error: 'Book not found' });
}
res.status(200).send(result);
});

module.exports = router;
```

## ✓ Paso 2: Añadir la definición a `library.schema.yaml`

Busca la sección `paths:` y asegúrate de que `DELETE /book/{id}` tenga esta estructura:

```
/book/{id}:
  delete:
    summary: Delete a book by ID
    parameters:
      - in: path
        name: id
        required: true
        schema:
          type: string
    responses:
      '204':
        description: Book deleted successfully
      '400':
        description: Invalid ID format
      '404':
        description: Book not found
```

## ✏️ ¿Cómo comprobarlo?

Haz peticiones `DELETE` a:

- Un id **válido y existente** → te devuelve 204.
- Un id **válido pero inexistente** → te devuelve 404.
- Un id con formato incorrecto (123, por ejemplo) → te devuelve 400.

## DTD EJS

### DTD Ejercicio 1

```
<!DOCTYPE TVSCHEDULE [
<!ELEMENT TVSCHEDULE (CHANNEL+)>
<!ELEMENT CHANNEL (BANNER,DAY+)>
<!ELEMENT BANNER (#PCDATA)>
<!ELEMENT DAY (DATE,(HOLIDAY|PROGRAMSLOT+)+)>
<!ELEMENT HOLIDAY (#PCDATA)>
<!ELEMENT DATE (#PCDATA)>
```

```

<!ELEMENT PROGRAMSLOT (TIME,TITLE,DESCRIPTION?)>
<!ELEMENT TIME (#PCDATA)>
<!ELEMENT TITLE (#PCDATA)>
<!ELEMENT DESCRIPTION (#PCDATA)>
<!ATTLIST TVSCHEDULE NAME CDATA #REQUIRED>
<!ATTLIST CHANNEL CHAN CDATA #REQUIRED>
<!ATTLIST PROGRAMSLOT VTR CDATA #IMPLIED>
<!ATTLIST TITLE RATING CDATA #IMPLIED>
<!ATTLIST TITLE LANGUAGE CDATA #IMPLIED>
]>

```

## SOLUCIÓN:

### CASO SIMPLE:

```

<TVSCHEDULE NAME= "MI HORARIO">
  <CHANNEL CHAN="TELECINCO">
    <BANNER> Publi 1 </BANNER>
    <DAY>
      <DATE>2023-10-01</DATE>
      <HOLIDAY>VERANO</HOLIDAY>
    </DAY>
  </CHANNEL>
</TVSCHEDULE>

```

### COMPLEJO:

```

<TVSCHEDULE NAME= "MI HORARIO">
<CHANNEL CHAN="TELECINCO">
  <BANNER> Publi 1 </BANNER>
  <DAY>
    <DATE>2023-10-01</DATE>
    <HOLIDAY>VERANO</HOLIDAY>
  </DAY>
  <DAY>
    <DATE>2023-10-02</DATE>
    <PROGRAMSLOT>
      <TIME>12:00</TIME>
      <TITLE RATING="12+" LANGUAGE="en">Serie: Misterios del
Espacio</TITLE>
    </DAY>
  </CHANNEL>
</TVSCHEDULE>

```

## DTD Ejercicio 2

**Crea un DTD que valide el siguiente XML:**

```

<articles>
<article id="x34675">
<name>Apache Spark Architecture</name>
<month>december</month>

```



```
<author name="kay vennisla"/>
<reviews lang=""/>
<feedback > high rating</feedback>
<reviews lang="de">The best content with diagrams</reviews>
</article>
</articles>
```


### SOLUCIÓN:

```
<!DOCTYPE ARTICLES[
<!ELEMENT ARTICLES (ARTICLE+)>
<!ELEMENT ARTICLE(NAME, MONTH, AUTHOR, REVIEWS+, FEEDBACK?)>
<!ELEMENT NAME (#PCDATA)>
<!ELEMENT MONTH (#PCDATA)>
<!ELEMENT AUTHOR EMPTY>
<!ELEMENT FEEDBACK(#PCDATA)>
<!ELEMENT REVIEWS (#PCDATA)>

<!ATTLIST ARTICLE ID CDATA #REQUIRED>
<!ATTLIST AUTHOR NAME CDATA #REQUIRED>
<!ATTLIST REVIEWS LANG CDATA #REQUIRED>

]>
```

## 2. Apartado: No se puede validar el cuerpo de la petición

 **Librería necesaria:** `express.json()` ya está en Express  $\geq 4.16$ , pero puede pedirte usar:

- `body-parser`

 Instalar:

```
bash
CopiarEditar
npm install body-parser
```

Y añadir:

```
js
CopiarEditar
const bodyParser = require('body-parser');
app.use(bodyParser.json());
```

---

### 3. Apartado: La API no sigue lo que dice el `schema.yaml` de OpenAPI

⚠ **Librería necesaria:** `swagger-ui-express + yamljs`

📦 **Instalar:**

```
bash
CopiarEditar
npm install swagger-ui-express yamljs
```

En `app.js`:

```
js
CopiarEditar
const swaggerUi = require('swagger-ui-express');
const YAML = require('yamljs');
const swaggerDocument = YAML.load('./schema/library.schema.yaml');

app.use('/api-docs', swaggerUi.serve,
swaggerUi.setup(swaggerDocument));
```

---

### 4. Apartado: Añade CORS para que la API sea accesible desde el frontend

⚠ **Librería necesaria:** `cors`

📦 **Instalar:**

```
bash
CopiarEditar
npm install cors
```

Usar en el servidor:

```
js
CopiarEditar
const cors = require('cors');
app.use(cors());
```

---

### 5. Apartado: Necesitas hacer logs de peticiones

⚠ **Librería típica:** `morgan`

📦 **Instalar:**

```
bash
CopiarEditar
npm install morgan
```

Y usar:

```
js
CopiarEditar
const morgan = require('morgan');
app.use(morgan('dev'));
```

Para hacer la documentación con **apiDoc** (<https://apidocjs.com/>), se utiliza un sistema de **comentarios especiales en tu código fuente**, generalmente en los archivos `.js`, que luego apiDoc convierte en una documentación HTML navegable.

---

## ✓ 1. Instala apiDoc (una sola vez)

```
bash
CopiarEditar
npm install -g apidoc
```

---

## ✓ 2. Estructura básica de comentarios apiDoc

Debes escribir comentarios encima de cada ruta como este ejemplo:

```
js
CopiarEditar
/**
 * @api {get} /api/book Obtener todos los libros
 * @apiName GetBooks
 * @apiGroup Book
 *
 * @apiQuery {Number} [limit] Número máximo de libros a devolver
 * @apiQuery {String} [next] ID del último libro devuelto (para
paginación)
 *
 * @apiSuccess {Object[]} results Lista de libros.
 * @apiSuccess {String} results._id ID del libro.
 * @apiSuccess {String} results.title Título del libro.
 * @apiSuccess {String} results.author Autor del libro.
 * @apiSuccess {String} results.link Enlace HATEOAS del libro.
 * @apiSuccess {String} next ID del siguiente libro (si hay
paginación).
 *
 * @apiExample {curl} Ejemplo de uso:
 *   curl -i http://localhost:3000/api/book
 */
router.get('/', async (req, res) => {
  // Tu código de la ruta
});
```

Puedes documentar todos los métodos: `@api {post}`, `@api {put}`, `@api {delete}`...

---

## ✓ 3. Genera la documentación

Si tu código está en `./routes`, ejecuta:

```
bash
CopiarEditar
apidoc -i routes/ -o apidoc/
```

Esto generará una carpeta `apidoc/` con los archivos HTML.

---

## ✓ 4. Ver la documentación en el navegador

Sirve la carpeta generada con Express (opcional):

```
js
CopiarEditar
app.use('/docs', express.static(path.join(__dirname, 'apidoc')));
```

Ahora accedes en el navegador a:

```
bash
CopiarEditar
http://localhost:3000/docs
```

## ✓ APARTADO 6: Añadir POST /book con validación

**Objetivo:** Insertar libros y validar los campos requeridos del `schema.yaml`.

---

### 1. `routes/book.js`

Añade al final:

```
js
CopiarEditar
// Crear libro
router.post('/', async (req, res) => {
  const dbConnect = dbo.getDb();
  const { title, author } = req.body;

  if (!title || !author) {
    return res.status(400).json({ error: "Faltan campos requeridos:
'title' y 'author'" });
  }

  const nuevoLibro = {
    title,
    author,
    publishedDate: req.body.publishedDate || null
  };

  try {
    const result = await
dbConnect.collection(COLLECTION).insertOne(nuevoLibro);
    res.status(201).json(result.ops?.[0] || nuevoLibro);
  } catch (error) {
    res.status(500).json({ error: 'Error al insertar libro' });
  }
});
```

---

## 2. *library.schema.yaml*

Dentro de `paths > /book` añade:

```
yaml
CopiarEditar
post:
  summary: Añade un nuevo libro
  requestBody:
    required: true
    content:
      application/json:
        schema:
          type: object
          required:
            - title
            - author
          properties:
            title:
              type: string
            author:
              type: string
            publishedDate:
              type: string
  responses:
    '201':
      description: Libro creado correctamente
    '400':
      description: Faltan campos requeridos
```

---

## APARTADO 7: Añadir `previous` `o count` `total` en `GET /book`

---

### 1. *routes/book.js (modifica el GET / existente)*

```
js
CopiarEditar
// GET libros con total y previous
router.get('/', async (req, res) => {
  let limit = MAX_RESULTS;
  if (req.query.limit) {
    limit = Math.min(parseInt(req.query.limit), MAX_RESULTS);
  }

  let next = req.query.next;
  let query = {};
  if (next) {
    query = { _id: { $lt: new ObjectId(next) } };
  }

  const dbConnect = dbo.getDb();
  const totalCount = await
dbConnect.collection(COLLECTION).countDocuments();

  let results = await dbConnect.collection(COLLECTION)
    .find(query)
    .sort({ _id: -1 })
    .limit(limit)
```

```
    .toArray();

    const previous = results.length ? results[0]._id : null;
    next = results.length === limit ? results[results.length - 1]._id :
    null;

    results = results.map(book => ({
      title: book.title,
      author: book.author,
      _id: book._id,
      link:
`${req.protocol}://${req.get('host')}${req.baseUrl}/${book._id}`
    }));

    res.status(200).json({ results, next, previous, totalCount });
  });
}
```

---

## ✓ APARTADO 8: Filtro por autor

---

1. En el mismo GET /, añade justo después de `let query = {}` lo siguiente:

```
js
CopiarEditar
if (req.query.author) {
  query.author = req.query.author;
}
```

Con eso, si haces `/book?author=Asimov` solo saldrán libros de ese autor.

---

## \* Posibles comandos para probar:

- POST:

```
bash
CopiarEditar
curl -X POST http://localhost:3000/api/book \
-H "Content-Type: application/json" \
-d '{"title":"Dune", "author":"Frank Herbert"}'
```

- GET filtrado:

```
bash
CopiarEditar
curl http://localhost:3000/api/book?author=Frank%20Herbert
```

## ✓ Apartado 13: Validar API Key (`x-api-key`)

1. Crea archivo `middleware/auth.js`

```
js
CopiarEditar
```

```
// middleware/auth.js
require('dotenv').config();

module.exports = (req, res, next) => {
  const apiKey = req.headers['x-api-key'];
  if (!apiKey || apiKey !== process.env.API_KEY) {
    return res.status(403).json({ error: 'Forbidden: Invalid or missing API key' });
  }
  next();
};
```

## 2. Añade al .env

```
ini
CopiarEditar
API_KEY=123456
```

## 3. Aplica middleware en routes/book.js

```
js
CopiarEditar
const express = require('express');
const router = express.Router();
const dbo = require('../db/conn');
const { ObjectId } = require('mongodb');
const authMiddleware = require('../middleware/auth'); // ← NUEVO

router.use(authMiddleware); // ← Aplica a todas las rutas

// Resto de rutas...
```



## Apartado 14: Validar límite con MAX\_RESULTS

### 1. En routes/book.js, modifica esta parte de tu GET /book

```
js
CopiarEditar
let limit = MAX_RESULTS;
if (req.query.limit) {
  const userLimit = parseInt(req.query.limit);
  if (userLimit > MAX_RESULTS) {
    return res.status(400).json({ error: `Max allowed results: ${MAX_RESULTS}` });
  }
  limit = userLimit;
}
```

Ya estás usando MAX\_RESULTS del .env, así que con este if lanzas el 400 si se excede.



## Apartado 15: Añadir self y collection en HATEOAS

### 1. En GET /book, añade:

```
js
CopiarEditar
results = results.map(book => ({
  ...book,
```

```
    link:
    `${req.protocol}://${req.get('host')}${req.baseUrl}/${book._id}`,
    self:
    `${req.protocol}://${req.get('host')}${req.baseUrl}/${book._id}`,
    collection: `${req.protocol}://${req.get('host')}${req.baseUrl}`
  }));
```

**Así devuelves:**

```
json
CopiarEditar
{
  "_id": "...",
  "title": "...",
  "link": "http://localhost:3000/api/book/ID",
  "self": "http://localhost:3000/api/book/ID",
  "collection": "http://localhost:3000/api/book"
}
```