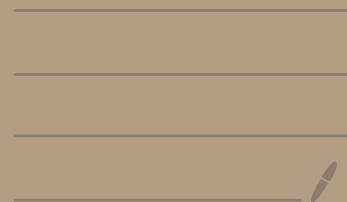


Simulacro



Problema 1

La carpeta api contiene un servidor web que arranca una API REST que está incompleta. La especificación OpenAPI está en `schemas/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.

1. ENV
PORT=3010
BASE_URL=/api/v2

Database
MONGODB_URI=mongodb://127.0.0.1/sw2
MAX_RESULTS=5

!schemas /library. schema.yaml

servers:
 - url: localhost:3010/api/v2

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.

!schemas /library. schema.yaml

```
paths:  
  /books  
    get:  
      summary: GET all books  
      description: GET all books  
      responses:  
        "200":  
          description: "OK"  
          content:  
            application/json:  
              schema:  
                $ref: '#/components/schemas/Books'
```

Devuelve un array de Books

`_id, title, author`

!routes /book.js

```
//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: new ObjectId(next)}  
  }  
  const dbConnect = dbo.getDb();  
  let results = await dbConnect  
    .collection(COLLECTION)  
    .find(query)  
    .sort({_id: -1})  
    .limit(limit)  
    .toArray()  
  if (results.length == res.status(400).send("Error searching for books"))  
    next = results.length == limit ? results.length - 1 : null;  
  res.json(results, next);  
});
```

Hay que añadir una
proyección

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 Modificar `routes/book.js` → GET book ② Link a `get book/{id}.book`

② Modificar `schemas /library. schema.yaml` → `!bookLink` ③ `Properties: { required: [link] type: string }`

link
type: string

```
router.get('/', async (req, res) => {  
  let results = await dbConnect  
    .collection(COLLECTION)  
    .find(query)  
    .project({_id: 1, title: 1, author: 1})  
    .sort({_id: -1})  
    .limit(limit)  
    .toArray()  
  if (err) res.status(400).send("Error searching for books");  
  // next = results.length == limit ? results.length - 1 : null;  
  // res.json(results, next); status(200); --> En vez de enviarlo, añadimos el link  
  
  results = results.map(book => {  
    return (  
      ...book, // mantenemos los campos existentes: _id, title, author  
      link: `${BASE_URL}/${book._id}` // añadimos el campo "link" con la URL del recurso  
    );  
  });  
  
  next = results.length == limit ? results.length - 1 : null;  
  res.status(200).json({ results, next });  
});  
  
// Construimos la base de la URL usando variables de entorno  
const PORT = process.env.PORT || 3010;  
const BASE_URL = process.env.BASE_URL || 'http://127.0.0.1';  
const bookLink = `http://localhost:${PORT}${BASE_URL}/book`;
```

!BASE_URL

Apartado 3.

(2)

```

components:
schemas:
  BookMin:
    type: object
    properties:
      _id:
        $ref: "#/components/schemas/ID"
      title:
        type: string
        description: Book title
      author:
        type: string
        description: Book author
      link:
        type: string
        description: URL to the full book resource (HATEOAS)
    required:
      - _id
      - title
      - author
      - link

```



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.

9 schemas/library.schema.yaml

```

delete:
  tags:
    - book
  summary: Deletes a book
  description: ''
  operationId: deleteBook
  responses:
    '200':
      description: Successful operation
    '400':
      description: Invalid book ID

```

9 routes/book.js

```

//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); → código 200
});

```

falta código 400

Solución : si falla, q de código 400

9 routes/book.js

```

router.delete('/:id', async (req, res) => {
  try {
    const query = { _id: new ObjectId(req.params.id) };
    const dbConnect = dbo.getDb();
    const result = await dbConnect.collection(COLLECTION).deleteOne(query);
    res.status(200).send(result);
  } catch (error) {
    // Captura errores como ObjectId inválido y responde 400
    res.status(400).send({ message: 'Invalid book ID' });
  }
});

```

Problemas de MongoDB

Apartado 1.

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

```

db.listingAndReviews.aggregate([
  { $group: { _id: "$name", totalReviews: { $sum: "$reviews" } } },
  { $sort: { totalReviews: -1 } },
  { $limit: 1
])

```

Apartado 2.

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

```

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

```

Apartado 3.

En la colección listingAndReviews indique para cada tipo de property_type el número de alojamientos de ese tipo.

```

db.listingAndReviews.aggregate([
  { $group: { _id: "$property_type", count: { $sum: 1 } } }
])

```

Apartado 4.

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

```

db.listingAndReviews.aggregate([
  { $match: { beds: { $in: [ 2, 3, 4, 5 ] } } },
  { $group: { _id: "$beds", count: { $sum: 1 } } },
  { $sort: { _id: 1 } }
])

```

Uso de "map" en Mongoose

```
const { MongoClient } = require('mongodb');

async function main() {
  const uri = 'mongodb://localhost:27017';
  const client = new MongoClient(uri);

  try {
    await client.connect();
    const db = client.db('miBaseDeDatos');
    const collection = db.collection('usuarios');

    // Buscar todos los documentos
    const usuarios = await collection.find().toArray();

    // Usar .map() para transformar los datos
    const nombresMayus = usuarios.map(u => u.nombre.toUpperCase());

    console.log(nombresMayus);

  } catch (error) {
    console.error(error);
  } finally {
    await client.close();
  }
}

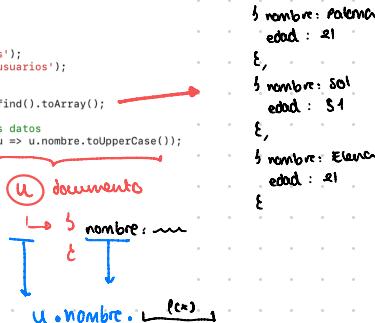
main()
  .catch((err) => {
    console.error(err);
  })

```

↑ nombre: Paloma
edad: 21
E,
↑ nombre: Sol
edad: 31
E,
↑ nombre: Elena
edad: 21
E,

(u) documentos

↳ ↑ nombre: ...
E & T



Ejemplo del simulacro:

```
results = results.map(book => {
  return {
    ...book, // mantenemos los campos existentes: _id, title, author
    link: `${BASE_URL}/${book._id}` // añadimos el campo "link" con la URL del recurso
  };
});
```

map(book) \Rightarrow 3 return: 3
... book, \Rightarrow mantener campos existentes
link: $\$book$ BASE-URL / \$book.id
); t: \downarrow
; addimos campo
; \$book: {
; name
; } \Rightarrow referenciamos