

# Simulacro


---

---

---

---

---



## 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.

**! .ENV**

```
PORT=3000 3010
BASE_URI=/api /v2

# Database
MONGODB_URI=mongodb://127.0.0.1:27020/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:
  /book:
    get:
      summary: GET all books
      description: GET all books
      responses:
        "200":
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/Books'
```

```
schemas:
  Books:
    type: object
    properties:
      results:
        type: array
        items:
          type: object
          properties:
            _id:
              type: string
              description: Book next ID for pagination search
            next:
              type: string
              description: Book next ID for pagination search
            required:
              - results
              - next
            BookArray:
              type: array
              items:
                type: object
                properties:
                  _id:
                    type: string
                    description: '#/components/schemas/BookId'
                  $ref: '#/components/schemas/Books'
```

**! 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;
  if (next) {
    query = { _id: { $lt: new ObjectId(next) } }
  }
  const dbConnect = db.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 });
});
```

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

**! hay q añadir una proyección**

**devuelve un array de BookIds**

**\_id, title, author**

## 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. Modificar schemas/library.schema.yaml → ? BookId link
- link type: string

1

```
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();
  .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); → En vez de enviarlo, añadimos el link
  results = results.map(book => {
    ...book, // mantenemos los campos existentes: _id, title, author
    link: `${BASE_URI}/${book._id}` // añadimos el campo "link" con la URL del recurso
  });
  next = results.length == limit ? results[results.length - 1]._id : null;
  res.status(200).json({ results, next });
});
```



**BASE\_URI**

```
// Contraseña la base de datos usando variables de entorno
const DB = process.env.DB || 'books'
const BASE_URI = process.env.BASE_URI || 'http://localhost:3000'
const MONGODB_URI = `mongodb://${DB}:${DB_PASSWORD}@localhost:27020/${DB}`
```

## 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.

🔗 [schemas/library.schemas.yam1](https://schemas.library.schemas.yam1)

solución : si falta, q de código 400

🔗 routes/book.js

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

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

🔗 routes/book.js

```
//deleteBookById()
router.delete('/:id', async (req, res) => {
  const query = { _id: new ObjectId(req.params.id) };
  const dbConnect = db.getConnection();
  let result = await dbConnect
    .collection(COLLECTION)
    .deleteOne(query);
  res.status(200).send(result); // código 200
});
```

falta código 400

## 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 } }
])
```