app.use(process.env.BASE_URI + '/peliculas', peliculasRouter); PQ /PELICULAS

PORT=3000

BASE_URI=/api/v1

process.env.BASE_URI viene de tu .env: BASE_URI=/api/v1

Luego se le suma '/peliculas': '/api/v1' + '/peliculas' = '/api/v1/peliculas'

¿Y qué pasa dentro de routes/peliculas.js?

```
router.get('/', (req, res) => {
  res.send("Todas las películas");
});
```

http://127.0.0.1:3000/api/v1/peliculas

LUEGO MI bin/www

**el archivo que arranca tu servidor** Express.

npm install

npm start

node app.js

ejemplo postman http://localhost:3000/api/v1/peliculas

---

Se puede poner:

**components → schemas**

Si vas a reutilizar esa validación en varios sitios.

**Dentro de requestBody → content → schema**

Cuando quieres que el cuerpo de la petición cumpla una u otra forma (por ejemplo: email o teléfono, pero al menos uno).

```
 pelicula:
    allOf:
      - required:
        - titulo
        - directores
```

- actores

- duracion

- resumen

- $ref: "#/components/schemas/pelicula_actualizar"

allOf Cumple **todos** los schemas.

Cumple **al menos uno** de los schemas.

anyOf:

- required:
  - email
- required:
  - telefono
  - pasaporte

Cumple **exactamente uno**.

oneOf:

- required:
  - dni
- required:

---

Los tags en OpenAPI sirven para **organizar las rutas en grupos** con nombres.
👉 **No cambian el funcionamiento de la API**, solo sirven para **ordenar y documentar mejor**.

tags:

- name: Películas
  description: Operaciones relacionadas con películas
- name: Sesiones
  description: Operaciones relacionadas con las sesiones

/peliculas:

get:

tags:

- Películas

summary: GET todas las películas

```
    description: GET todas las películas
    responses:
      "200":
       -------
  post:
    tags:
      - Películas
       get:
    tags:
      - Películas
```

---

**¿Qué es example: en OpenAPI?**

El campo example se usa para **mostrar un ejemplo real de cómo debería ser ese objeto o valor**. Ayuda a Swagger UI a mostrar datos de ejemplo automáticos.

---

✅ **¿Dónde se añade?**

Se añade **dentro de un schema** en components.schemas, justo al final del objeto

```
pelicula:
  allOf:
    - required:
        - titulo
        - duracion
    - $ref: "#/components/schemas/pelicula_actualizar"
  additionalProperties: true
```

**Para añadir un ejemplo completo:**

yaml

CopiarEditar

```
pelicula:
  allOf:
    - required:
        - titulo
        - duracion
```

```
    - $ref: "#/components/schemas/pelicula_actualizar"
  additionalProperties: true
  example:
    titulo: "Titanic"
    directores:
      - nombre: "James Cameron"
    actores:
      - nombre: "Leonardo DiCaprio"
      - nombre: "Kate Winslet"
    duracion: 195
    edad_min: "PG-13"
    resumen: "Un joven y una joven se enamoran a bordo del Titanic."
    genero: ["drama", "romance"]



paths:
  /book:
    post:
      summary: Añadir libro
      requestBody:
        required: true
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/Book'
            example:
              titulo: "Cien años de soledad"
              author: "Gabriel García Márquez"
              year: 1967
```

---

**CUÁNDO TE PIDEN "PON UN example"? ¿DÓNDE Y PARA QUÉ MÉTODO?**

Depende de **qué tipo de ejemplo te pidan**:

---

✅ **CASO 1: Te piden un ejemplo de JSON que se envía**

 “Pon un example de un libro que se va a crear”

🔷 🔷 Esto es para **POST** o **PUT** → porque se **envía** en el cuerpo (body).

🔧 Dónde lo pones:

yaml

CopiarEditar

```
paths:
 /book:
   post:
    requestBody:
     content:
       application/json:
         schema:
           $ref: '#/components/schemas/Book'
         example:
           titulo: "Rayuela"
           author: "Julio Cortázar"
           year: 1963
```

---

✅ **CASO 2: Te piden un ejemplo de parámetro**

 “Pon un ejemplo del parámetro year en query”

🔷 Esto es para métodos **GET** con parameters.

🔧 Dónde lo pones:

yaml

CopiarEditar

```
parameters:
 - name: year
   in: query
   schema:
    type: integer
```

example: 1999

---

✅ **CASO 3: Te piden un ejemplo del ID en path**

 “Pon un ejemplo de ID que se pasa por path (URL)”

🔷 Para **GET /book/{id}**, **PUT /book/{id}**, **DELETE /book/{id}**.

🔧 Dónde lo pones:

yaml

CopiarEditar

components:

 parameters:

  ID:

   name: bookId

   in: path

   required: true

   schema:

    type: string

    example: 6463448ae7684d03f44af30f

---

✅ **CASO 4: Te piden un ejemplo en la respuesta**

 “Pon un ejemplo de la respuesta 200 al crear un libro”

🔷 Se pone dentro del responses > 200 > content.

🔧 Dónde lo pones:

yaml

CopiarEditar

responses:

 '200':

  description: Libro creado correctamente

  content:

   application/json:

    schema:

     $ref: '#/components/schemas/Book'

```
      example:
        titulo: "1984"
        author: "George Orwell"
        year: 1949
```

"Incluye un ejemplo de libro en la creación de un recurso".

👉 Tú sabes que eso va en el POST ➜ requestBody ➜ example.

yaml

CopiarEditar

```
post:
 summary: Crea un nuevo libro
 requestBody:
   required: true
   content:
     application/json:
       schema:
         $ref: '#/components/schemas/Book'
       example:
         titulo: "1984"
         author: "George Orwell"
         year: 1949
```

---

📌 **Otro enunciado:**

"Incluye un ejemplo del objeto libro que devuelve el GET por ID"

👉 Eso va en: GET /book/{id} ➜ responses ➜ example

yaml

CopiarEditar

```
get:
 summary: Obtiene un libro por ID
 responses:
```

```
'200':
  description: OK
  content:
    application/json:
      schema:
        $ref: '#/components/schemas/Book'
      example:
        titulo: "Cien años de soledad"
        author: "Gabriel García Márquez"
        year: 1967
```

---

**□ ¿CÓMO SABER QUÉ MÉTODO USAR?**

🔷 Si el ejemplo es de lo que envías → **POST o PUT**
🔷 Si es de lo que filtras por URL → **GET (con parameters)**
🔷 Si es de ID en la URL → **GET/PUT/DELETE con path**
🔷 Si es de lo que recibes como respuesta → **GET/POST/PUT → responses**

---

🔗 **TAMBIÉN VALE PARA PROPIEDADES**

Dentro de un schema, también puedes poner example: en cada propiedad. Por ejemplo:

yaml

CopiarEditar

```
titulo:
  type: string
  example: "Titanic"
```

**¿Qué son los parameters in: query?**

Son **parámetros que van en la URL después del signo de interrogación ?**.
Por ejemplo, si escribes esto en el navegador o en Postman:

/peliculas?genero=drama

👆 Eso es un **parámetro de tipo query**. Va fuera de la ruta /peliculas y sirve para **filtrar, buscar, paginar, ordenar**, etc. "Haz que el endpoint /peliculas permita filtrar por género con query params"

```
router.get('/', async (req, res) => {
  const dbConnect = dbo.getDb();
  const query = {};

  if (req.query.genero) {
    query.genero = req.query.genero;
  }

  if (req.query.duracionMin) {
    query.duracion = { $gte: parseInt(req.query.duracionMin) };
  }

  try {
    const results = await dbConnect
      .collection(COLLECTION)
      .find(query)
      .toArray();
    res.status(200).json({ results });
  } catch (err) {
    res.status(400).send('Error al buscar películas');
  }
});
```

En postman en el post {

```
  "titulo": "Mi película comedia",
  "directores": [{ "nombre": "Pedro" }],
  "actores": [{ "nombre": "Ana" }],
  "duracion": 120,
  "resumen": "Una comedia divertida",
  "genero": ["comedia"]
}
```

En yaml /peliculas:

```
  get:
    summary: GET todas las películas
    description: GET todas las películas
    parameters:
     - name: genero
       in: query
       description: Filtrar por género
       required: false
       schema:
       type: string
  minimum: 1900 dependiedno de lo q difa  e
    maximum: 2100
```

si poemos lo de min en el router.get

```
if (year >= 1900 && year <= 2100) {
 query.year = year;
} else {
 return res.status(422).json({ error: "El año debe estar entre 1900 y 2100" });
}
```

---

router.get

```
let results = await dbConnect
 .collection(COLLECTION)
 .find(query)
 .sort({ _id: -1 })
 .limit(limit)
 .project({ titulo: 1 })
 .toArray()
 .catch(err => res.status(400).send('Error al buscar películas'));

next = results.length == limit ? results[results.length - 1]._id : null;
```

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

✅ EXPLICACIÓN PASO A PASO:

1. const dbConnect = dbo.getDb();

👉 Obtienes la conexión activa a la base de datos.

2. .collection(COLLECTION)

👉 Accedes a la colección (por ejemplo "books" o "movies").

3. .find(query)

👉 Aplicas el filtro (puede ser vacío {} o con condiciones como { year: 2008 }).

4. .sort({ _id: -1 })

👉 Ordenas los resultados por _id descendente (de más nuevo a más antiguo).

5. .limit(limit)

👉 Limita el número de resultados devueltos. Muy típico en paginación.

6. .project({ titulo: 1 })

👉 Proyección: solo devuelve el campo titulo. El _id siempre viene por defecto salvo que pongas _id: 0.

7. .toArray()

👉 Convierte el cursor en un array de resultados para trabajar con ellos.

8. .catch(...)

👉 Si hay un error en toda la cadena, lo captura y responde con 400 Bad Request.

```
let results = await dbConnect
  .collection("books")
  .find(query)
  .collation({ locale: "en", strength: 2 }) // insensible a mayúsculas
```

```
  .sort({ year: -1 })
  .skip(10)
  .limit(5)
  .project({ title: 1, author: 1, _id: 0 })
  .toArray();
```

.skip() → si haces paginación por número de página

.collation() → si quieres búsquedas insensibles a mayúsculas

.explain() → si necesitas analizar el rendimiento

.countDocuments() → si necesitas saber cuántos hay sin traerlos

---

IMPORTANTE EN EL POST

```
//addPelicula()
router.post('/', async (req, res) => {
  const dbConnect = dbo.getDb();
  console.log(req.body);
  let result = await dbConnect
    .collection(COLLECTION)
    .insertOne(req.body);
  res.status(201).send(result);
}); Y QUE PASA IMPORTNATE EN MI YAML MI nueElemnto estas así     nuevoElemento:
      type: object
      properties:
        id:
          $ref: "#/components/schemas/id"
        url:
          description: Enlace a la sesión o película creada
          type: string
          format: uri
      example: {"id": 123,
        "url": "http://example.com/api/v1/peliculas/123"}
```

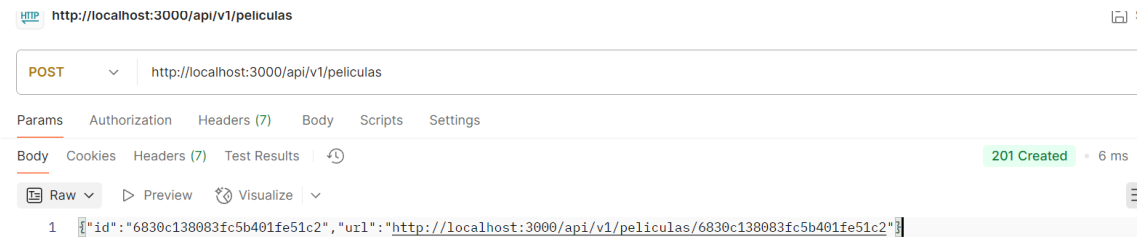
Ahora el pos tno me devuelve con url tenemos q cambiar en pelicular el psot

```
// addPelicula()
router.post('/', async (req, res) => {
 const dbConnect = dbo.getDb();
 try {
  const result = await dbConnect
   .collection(COLLECTION)
   .insertOne(req.body);

  // Construimos la URL del nuevo recurso:
  const nuevaUrl = `${req.protocol}://${req.get('host')}${req.baseUrl}/${result.insertedId}`;

  res.status(201).send({
   id: result.insertedId,
   url: nuevaUrl
  });

 } catch (err) {
  res.status(400).send('Error al insertar película');
 }
});
```




vemos url

**Paginación en APIs REST**

**¿Qué es la paginación?**

La paginación es una técnica que permite dividir una gran cantidad de datos en partes más pequeñas llamadas 'páginas'. Esto se hace para mejorar el rendimiento y la experiencia del usuario al consultar datos en una API REST.

**Ejemplo de paginación en una API REST**

1. Primer GET:

***GET /peliculas?limit=3***

Respuesta:

```
{
  "results": [
    {"titulo": "Titanic"},
    {"titulo": "Avatar"},
    {"titulo": "Shrek"}
  ],
  "next": "66493fa3f84cdb0f3e12d9f7"
}
```

2. Segundo GET:

***GET /peliculas?limit=3&next=66493fa3f84cdb0f3e12d9f7***

**Implementación en el backend (Node.js / Express)**

En tu archivo peliculas.js tienes la lógica para manejar los parámetros de paginación 'limit' y 'next'. Esto permite devolver solo un número limitado de resultados y continuar desde donde se quedó la consulta anterior.

```
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 results = await dbConnect
    .collection(COLLECTION)
    .find(query)
    .sort({ _id: -1 })
    .limit(limit)
    .toArray();
  next = results.length === limit ? results[results.length - 1]._id : null;
```

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

**Configuración en OpenAPI (YAML)**

En tu archivo OpenAPI, debes declarar los parámetros 'limit' y 'next' como parámetros de tipo query:

```
/peliculas:
  get:
    summary: Obtener lista de películas con paginación
    parameters:
      - name: limit
        in: query
        description: Límite máximo de resultados a devolver
        required: false
        schema:
          type: integer
          minimum: 1
          maximum: 50
      - name: next
        in: query
        description: ID de la última película devuelta, para la siguiente página
        required: false
        schema:
          type: string
    responses:
      "200":
        description: OK
        content:
          application/json:
            schema:
              $ref: "#/components/schemas/peliculas"
```

**1. Cambiar el orden de los resultados**

Te pueden pedir que lo devuelvas en orden ascendente (_id: 1) o descendente (_id: -1), y que sea configurable por query:

js

CopiarEditar

// Añadir al principio

let sortOrder = req.query.order === 'asc' ? 1 : -1;

```
// Cambiar esto:
.sort({_id: -1})
// Por esto:
.sort({_id: sortOrder})
```

---

PAGINACION

YAML

```
schemas:
   Books:
     type: object
     properties:
       results:
         $ref: "#/components/schemas/BooksArray"
       next:
         type: string
         description: Book next ID for pagination search
     required:
       - results
       - next
```

Books.js

```
 let next  = req.query.next;


 let query = {}
 if (next){
   query = {_id: {$lt: new ObjectId(next)}}
 }
```

---

Hateous

```
       $ref: "#/components/schemas/BookMin"
   BookMin:
     type: object
     properties:
       _id:
```

```
        $ref: "#/components/schemas/ID"
      title:
        type: string
        description: Book title
      author:
        type: string
        description: Book author
    required:
      - _id
      - title
      - author
      - link
```

```
Router.get
  const baseUrl = `${req.protocol}://${req.get('host')}${req.baseUrl}`;

  const resultsWithLinks = results.map(book => ({
    ...book,
    link: `${baseUrl}/${book._id}`
  }));
  res.status(200).json({ results:resultsWithLinks, next });
}
  catch (err) {
    res.status(400).send('Error al buscar libros');
  }

});
```

---

ROUTER.GET ID

LO PRIBAMOS ASI http://localhost:3011/api/v2/book/683adc2244caa575aea45170 (ID) Q HAYA

```
router.get('/:id', async (req, res) => {
  const dbConnect = dbo.getDb();
  const id = req.params.id;

  // Validación de formato de ID
  if (!ObjectId.isValid(id)) {
    return res.status(400).json({ error: 'ID no válido' }); // Error 400
  }

  try {
    const result = await dbConnect
      .collection(COLLECTION)
      .findOne({ _id: new ObjectId(id) });

    if (!result) {
      return res.status(404).json({ error: 'Libro no encontrado' }); // Error 404
    }

    res.status(200).json(result); // Éxito 200
  } catch (err) {
    res.status(500).json({ error: 'Error al buscar el libro' });
  }
});
```