



Clase 46. Programación Backend

***Introducción a frameworks de
desarrollo backend:
Parte II***



OBJETIVOS DE LA CLASE

- Conocer sobre frameworks para desarrollo backend con Node:
 - Sails
 - Koa

CRONOGRAMA DEL CURSO

Clase 45



**Introducción a
frameworks de desarrollo
backend: Parte I**

Clase 46



**Introducción a
frameworks de
desarrollo backend:
Parte II**

Clase 47



**Deno: el futuro de
NodeJS?**

SAILS



¿De qué se trata?



- **Sails.js** es un framework MVC construido sobre Express.
- Se utiliza para desarrollar API RESTful y aplicaciones web modernas. Cuenta con soporte para los requisitos de aplicaciones modernas.
- Imita el modelo de ruby on rails para la creación de pequeñas o grandes aplicaciones de forma rápida, sencilla y segura.
- No importa qué base de datos utilicemos, Sails provee una capa de abstracción, que hace que la elección de la misma, le sea indiferente.



¿De qué se trata?



- Cuenta con la capacidad de crear RESTfull JSON APIs de forma automática.
- Trae incorporado el modulo Socket.io.
- Genera rutas automáticas para sus controladores.
- Provee sistema de autenticación de usuarios y control de acceso basado en roles.
- Grunt como Task Runner (Tareas automáticas como minificación, compilación, testing, etc)
- Assets: Todos los archivos de sus correspondientes directorios (css,js) son unificados en un único archivo y minificados, para reducir considerablemente la carga de la página y la cantidad limitada de peticiones del navegador.

API RESTful CON SAILS



Instalación



- En primer lugar instalamos el framework con el comando: `npm -g install sails`.
- Con el comando `sails -v` podemos ver la versión del framework que se

```
C:\Cursos\Coderhouse\CursoBackend\Clase46\Sails
λ npm -g install sails
npm WARN deprecated uuid@3.0.1: Please upgrade to version 7 or higher. Older versions may use Math.random() in certain circumstances
. See https://v8.dev/blog/math-random for details.
C:\Users\EducacionIT\AppData\Roaming\npm\sails -> C:\Users\EducacionIT\AppData\Roaming\npm\node_modules\sails\bin\sails.js
+ sails@1.4.3
updated 1 package in 12.028s

C:\Cursos\Coderhouse\CursoBackend\Clase46\Sails
λ sails -v
1.4.3

C:\Cursos\Coderhouse\CursoBackend\Clase46\Sails
λ |
```




Creación del proyecto

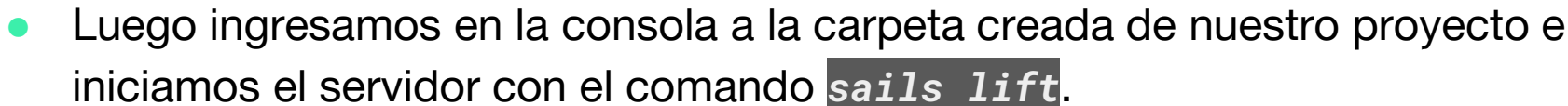


- Para crear un proyecto con Sails usamos el comando: `sails new <name>`.
- Nos da a elegir entre 2 opciones: web app o Empty. En este caso elegimos Empty.

```
Cmder

C:\Cursos\Coderhouse\CursoBackend\Clase46\Sails
λ sails new BookStoreAPI
Choose a template for your new Sails app:
1. Web App · Extensible project with auth, login, & password recovery
2. Empty   · An empty Sails app, yours to configure
(type "?" for help, or <CTRL+C> to cancel)
? 2
info: Installing dependencies...
Press CTRL+C to cancel.
(to skip this step in the future, use --fast)
info: Created a new Sails app `book-store-api`!

C:\Cursos\Coderhouse\CursoBackend\Clase46\Sails
λ |
```

***CODER HOUSE***



Creación del proyecto



- El archivo `app.js` que se nos crea tiene al principio algunas indicaciones de cómo usar el proyecto, algunos comandos.

The screenshot shows the Visual Studio Code interface with the 'app.js' file open. The Explorer sidebar on the left shows a project structure with folders like 'controllers', 'models', 'policies', 'tasks', and 'views'. The 'app.js' file is selected in the Explorer. The main editor area displays the content of 'app.js', which includes comments explaining how to run the application and use the 'sails lift' command. The code is as follows:

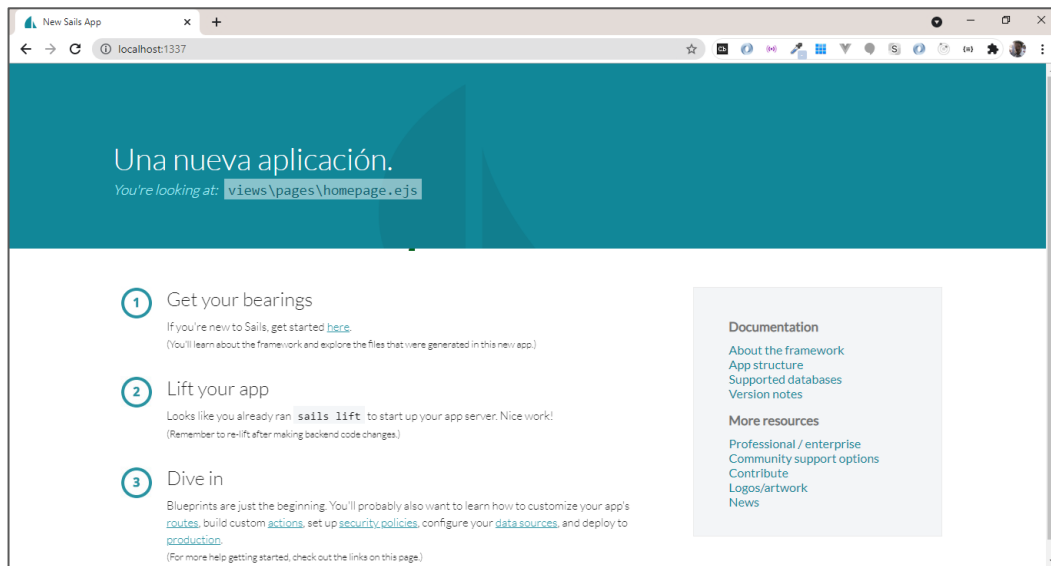
```
1 /**
2  * app.js
3  *
4  * Use `app.js` to run your app without `sails lift`.
5  * To start the server, run: `node app.js`.
6  *
7  * This is handy in situations where the sails CLI is not relevant or useful,
8  * such as when you deploy to a server, or a PaaS like Heroku.
9  *
10 * For example:
11 * => `node app.js`
12 * => `npm start`
13 * => `forever start app.js`
14 * => `node debug app.js`
15 *
16 * The same command-line arguments and env vars are supported, e.g.:
17 * `NODE_ENV=production node app.js --port=80 --verbose`
18 *
19 * For more information see:
20 * https://sailsjs.com/anatomy/app.js
21 */
22
23
24 // Ensure we're in the project directory, so cwd-relative paths work as expected
25 // no matter where we actually lift from.
26 // > Note: This is not required in order to lift, but it is a convenient default.
27 process.chdir(__dirname);
28
```



Creación del proyecto



- El servidor está escuchando en el puerto 1337, por lo que si ingresamos a <http://localhost:1337> vemos lo siguiente:



Creación de la API Rest



Creación del Api



- Sails proporciona la función de rutas de planos para generar rutas RESTful. Podemos crear una API con el comando `sails generate api <name>`. En este caso, creamos una API llamada “book”.

```
C:\Cursos\Coderhouse\CursoBackend\Clase46\Sails\BookStoreAPI
λ sails generate api book
Info: Created a new api!

C:\Cursos\Coderhouse\CursoBackend\Clase46\Sails\BookStoreAPI
λ sails lift
Info: Starting app...

-----
Excuse my interruption, but it looks like this app
does not have a "migrate" setting configured yet.
(perhaps this is the first time you're lifting it with models?)

Tired of seeing this prompt? Edit config/models.js.

In a production environment (NODE_ENV=production) Sails always uses
migrate: 'safe' to protect against inadvertent deletion of your data.
But during development, you have a few different options:

1. FOR DEV:      alter  wipe/drop and try to re-insert ALL my data (recommended)
2. FOR TESTS:    drop   wipe/drop ALL my data every time I lift sails
3. FOR STAGING:  safe   don't auto-migrate my data. I will do it myself

Read more: sailsjs.com/docs/concepts/models-and-orm/model-settings#migrate
-----

What would you like Sails to do this time?
** NEVER CHOOSE "alter" or "drop" IF YOU ARE WORKING WITH PRODUCTION DATA **

prompt: ? 1
-----
Ok! Temporarily using migrate: 'alter'...
To skip this prompt in the future, edit config/models.js.
-----

Info: ** Auto-migrating... (alter)
Info:   Hold tight, this could take a moment.
Info:   ✓ Auto-migration complete.
```

- Entonces, al generar la API de esta forma se crean las rutas de GET, POST, PUT y DELETE por default.
- Luego, ingresamos de nuevo el comando `sails lift`, y vemos que nos da opciones para elegir para configurar la migración. Elegimos la primera opción, “**alter**”, ya que no tenemos, en este caso, una base de datos conectada a nuestra aplicación.



Creación del Api



- Al ejecutar el comando anterior, se crean dos archivos:
 - api/controllers/**BookController.js**,
 - api/models/**Book.js**.

The screenshot shows the Visual Studio Code interface with the Explorer sidebar on the left. The Explorer shows a project structure with folders like .tmp, api, controllers, helpers, models, and policies. Two files are highlighted: **BookController.js** and **Book.js**. The main editor area displays the content of **BookController.js**, which includes a JSDoc comment and a module.exports object.

```
1 /**
2  * BookController
3  *
4  * @description :: Server-side actions for handling incoming requests.
5  * @help       :: See https://sailsjs.com/docs/concepts/actions
6  */
7
8 module.exports = {
9
10
11 };
12
13
```

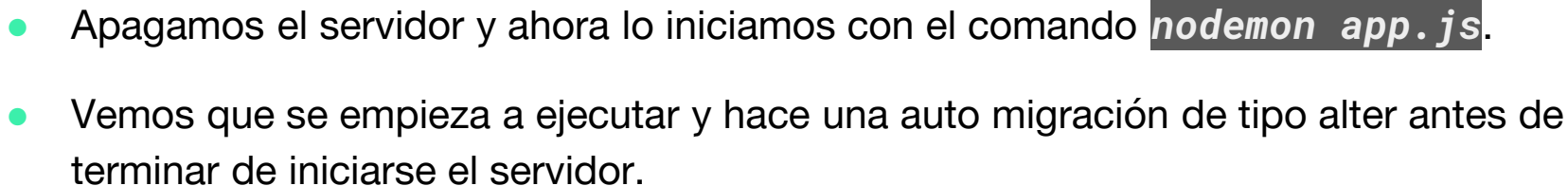


Creación del Api



- Vamos al archivo **config/models.js** , buscamos la línea donde se configura la opción de migración a 'alter', y la descomentamos, como se ve en la imagen:

```
47 * > Note that, when running in a production environment, this will be
48 * > automatically set to 'migrate: 'safe'', no matter what you configure
49 * > here. This is a failsafe to prevent Sails from accidentally running
50 * > auto-migrations on your production database.
51 * >
52 * > For more info, see:
53 * > https://sailsjs.com/docs/concepts/orm/model-settings#migrate
54 *
55 /*****
56 migrate: 'alter',
57
58 /*****
59
60 *
61 * Base attributes that are included in all of your models by default.
62 * By convention, this is your primary key attribute ('id'), as well as two
63 * other timestamp attributes for tracking when records were last created
64 * or updated.
65 *
66 * > For more info, see:
67 * > https://sailsjs.com/docs/concepts/orm/model-settings#attributes
68 *
69 /*****
```


***CODER HOUSE***

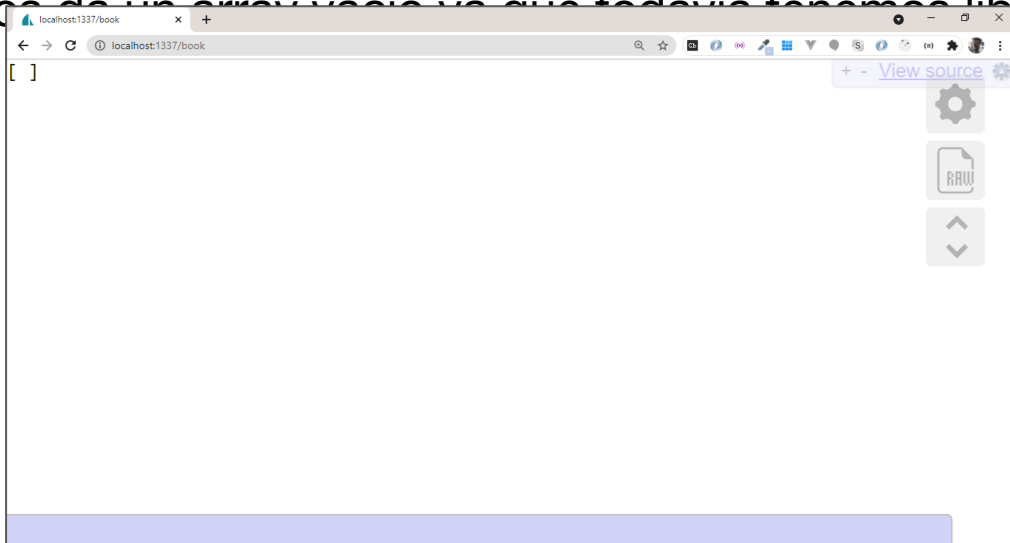
Usando la aplicación



Usando la aplicación



- Podemos ahora ya ingresar en el navegador a nuestra aplicación, en la ruta creada por sails “/book”. Para eso ingresamos a la url <http://localhost:1337/book>.
- Vemos que no da un error, ya que todavía tenemos libros almacenados.

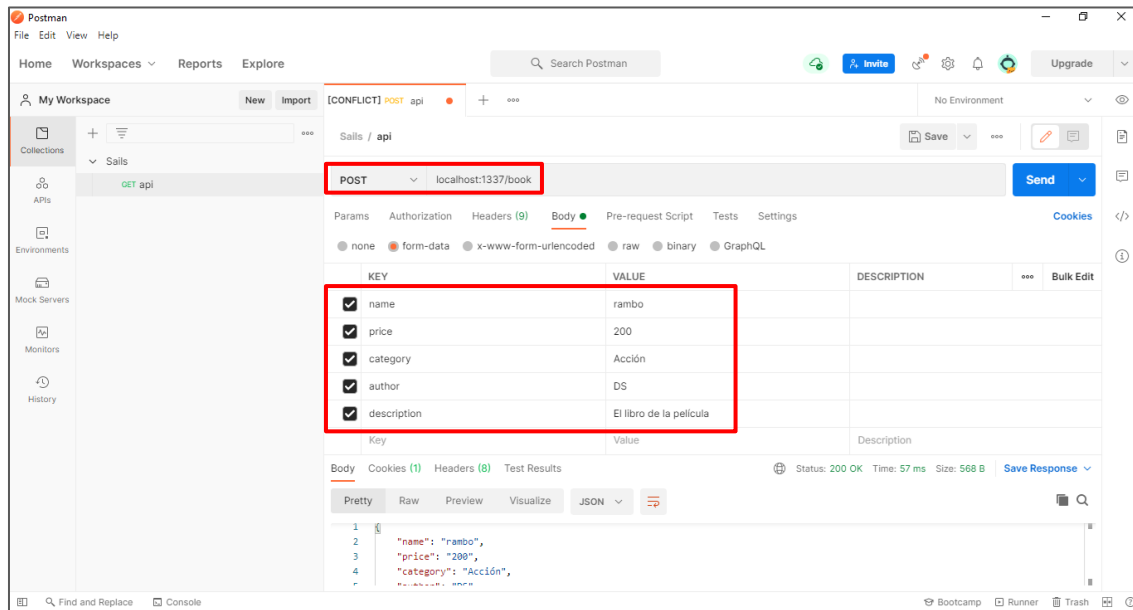




Usando la aplicación



- Podemos ir entonces a Postman a probar todas las rutas.
- Empecemos creando un nuevo libro con la ruta “/book” por POST. Le especificamos el *body* como muestra la imagen.





Usando la aplicación



- Luego, podemos ir nuevamente al navegador y probar la ruta “/book” por GET.
- Podemos observar un JSON con la información del libro creado.

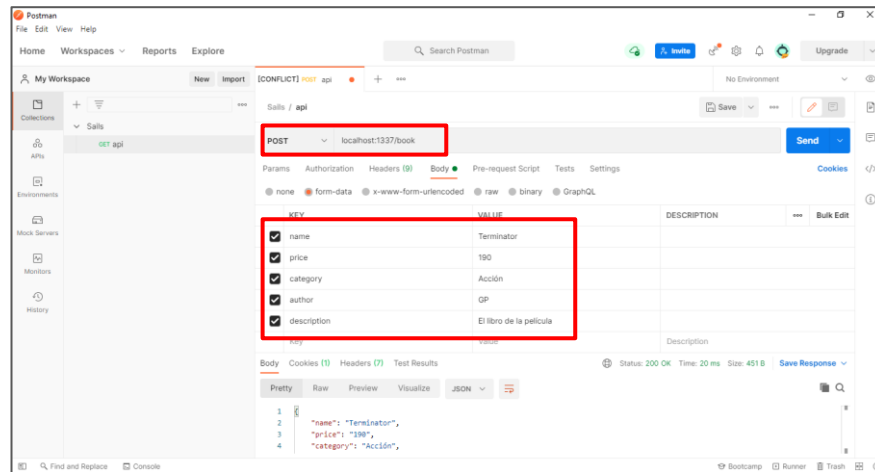
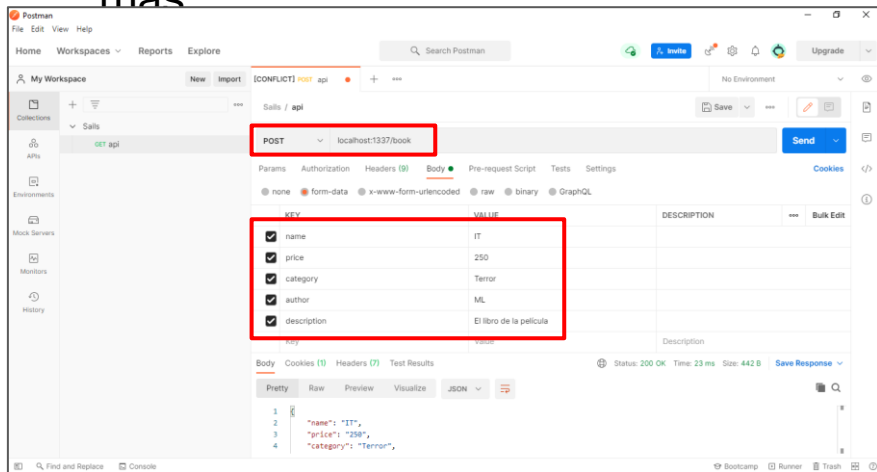
```
{
  "name": "rambo",
  "price": "280",
  "category": "Acción",
  "author": "95",
  "description": "El libro de la película",
  "createdat": 1624924575533,
  "updatedat": 1624924575533,
  "id": 1
}
```



Usando la aplicación



- Creamos ahora nuevamente con la ruta “/book” por POST en Postman dos libros más





Usando la aplicación



- Volviendo al navegador, vemos un JSON con la información de los 3 libros creados.

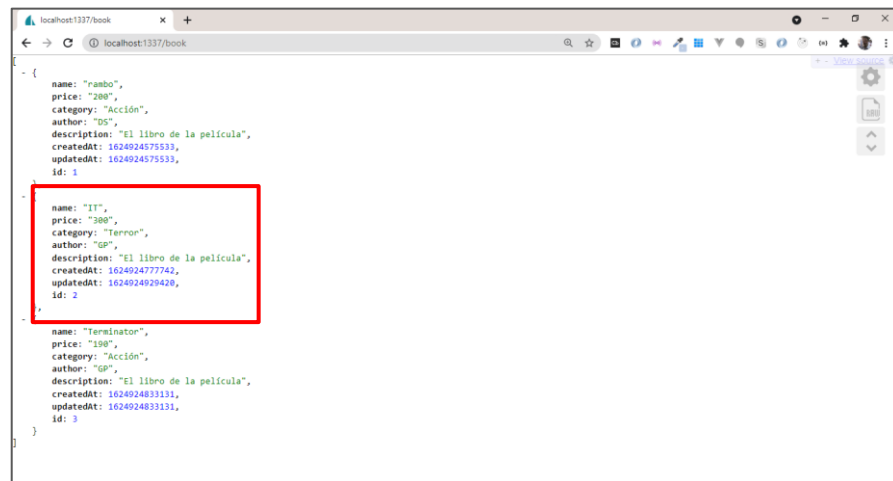
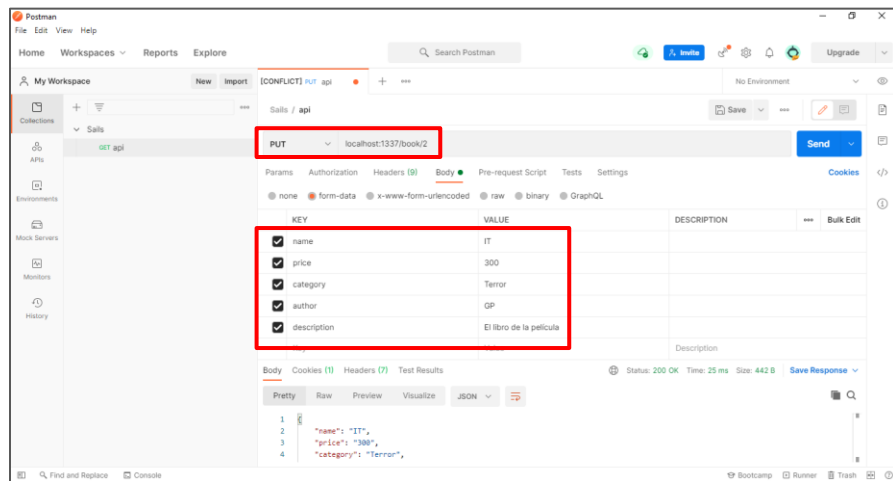
```
[
  {
    name: "rambo",
    price: "200",
    category: "Acción",
    author: "DS",
    description: "El libro de la película",
    createdAt: 1624924575533,
    updatedAt: 1624924575533,
    id: 1
  },
  {
    name: "IT",
    price: "250",
    category: "Terror",
    author: "ML",
    description: "El libro de la película",
    createdAt: 1624924777742,
    updatedAt: 1624924777742,
    id: 2
  },
  {
    name: "Terminator",
    price: "190",
    category: "Acción",
    author: "GP",
    description: "El libro de la película",
    createdAt: 1624924833131,
    updatedAt: 1624924833131,
    id: 3
  }
]
```



Usando la aplicación



- Probemos ahora en Postman la ruta “/book” por PUT. Vamos a modificar la información del libro con id = 2.
- Luego, si volvemos al navegador, vemos que su información se modificó correctamente.

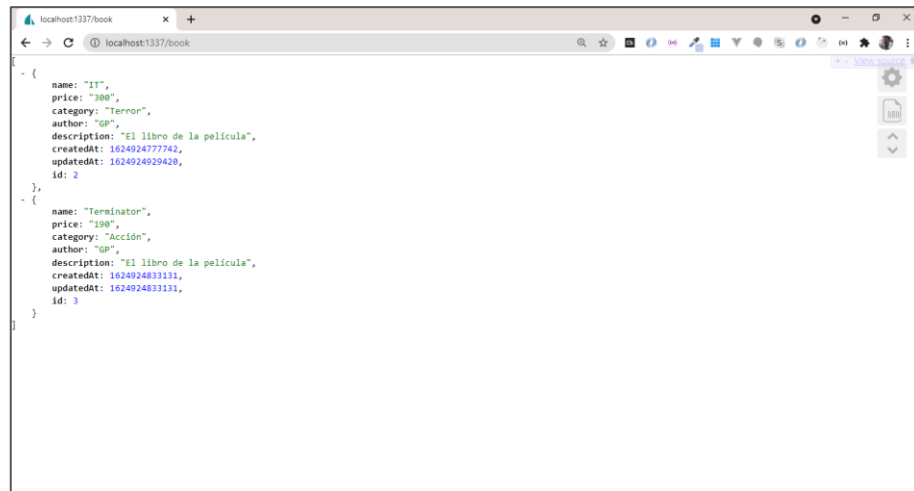
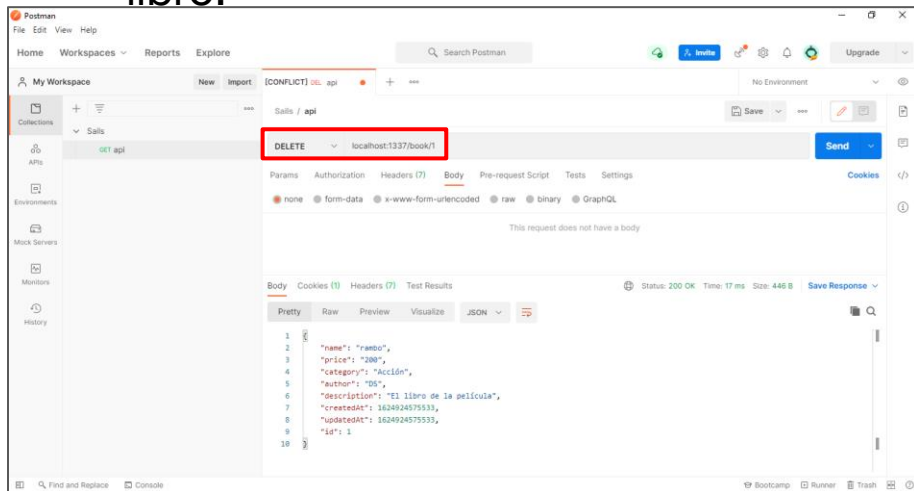




Usando la aplicación



- Probamos ahora la ruta “/book” por DELETE. Vamos a eliminar el libro con id = 1.
- Luego, observamos en el navegador que ya no tenemos la información de ese libro.

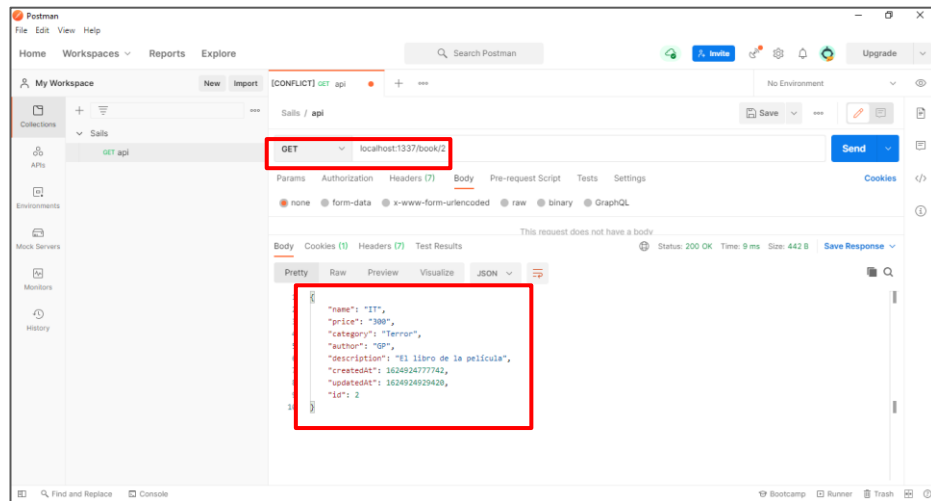
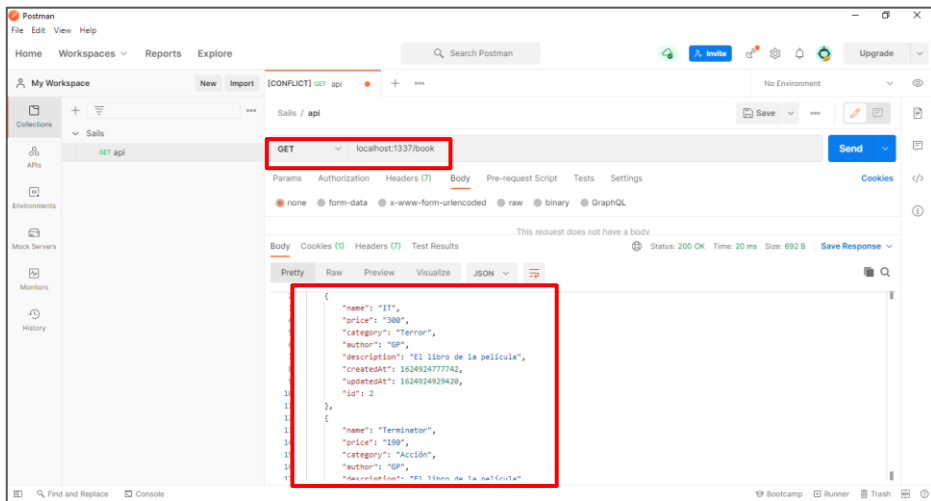




Usando la aplicación



- También podemos probar la ruta por GET en postman. Si la ruta es “/book” nos trae todos los libros almacenados. Si es “/book/id” nos trae solo la información del libro correspondiente al id.





Usando la aplicación



- Para ver dónde se almacenan los datos, vamos a la carpeta del proyecto llamada *.tmp*, y en esta tenemos la carpeta *localDiskDb*. Tenemos en esa carpeta el archivo ***book.db***. Los datos se almacenan allí como se muestra en la imagen.

The screenshot shows the Visual Studio Code interface. On the left, the Explorer sidebar is open, showing the project structure. The file `book.db` is selected under the `.tmp > localDiskDb` path. The main editor area displays the content of `book.db`, which is a JSON array of book records. The records include fields like `name`, `price`, `category`, `author`, `description`, `createdAt`, and `updatedAt`. Some records also have an `id` and a `deleted` flag.

```
1  {"$indexCreated":{"fieldName":"id","unique":true,"sparse":false}}
2  {"name":"rambo","price":"200","category":"Acción","author":"DS","description":"El libro de la
3  película","createdAt":"1624924575533","updatedAt":"1624924575533","id":"1","_id":"1"}
4  {"name":"II","price":"250","category":"Terror","author":"ML","description":"El libro de la película",
5  "createdAt":"1624924777742","updatedAt":"1624924777742","id":"2","_id":"2"}
6  {"name":"Terminator","price":"190","category":"Acción","author":"GP","description":"El libro de la
7  película","createdAt":"1624924833131","updatedAt":"1624924833131","id":"3","_id":"3"}
8  {"$deleted":true,"_id":"2"}
9  {"name":"II","price":"300","category":"Terror","author":"GP","description":"El libro de la película",
10 "createdAt":"1624924777742","updatedAt":"1624924929420","id":"2","_id":"2"}
11 {"$deleted":true,"_id":"1"}
12
```

PERSISTENCIA EN MYSQL

Persistencia de datos en MySQL



- Hasta ahora, la persistencia de datos la teníamos en archivo. Vamos a modificar eso, y persistir los datos en MySQL. Para eso, primero debemos instalar el módulo **sails-mysql** en nuestro proyecto, via npm.
- Luego, seteamos las variables para conectarnos a nuestra base de datos de MySQL. Esto lo hacemos en el archivo **config/datastores.js**.
- Además, debemos iniciar MySQL en nuestra computadora.

```
default: {  
  
  /*****  
  *  
  * Want to use a different database during development?  
  *  
  * 1. Choose an adapter:  
  *   https://sailsjs.com/plugins/databases  
  *  
  * 2. Install it as a dependency of your Sails app.  
  *   (For example: npm install sails-mysql --save)  
  *  
  * 3. Then pass it in, along with a connection URL.  
  *   (See https://sailsjs.com/config/datastores for help.)  
  *  
  *****/  
  adapter: 'sails-mysql',  
  url: 'mysql://user:password@host:port/database',  
  
},
```



Persistencia de datos en MySQL



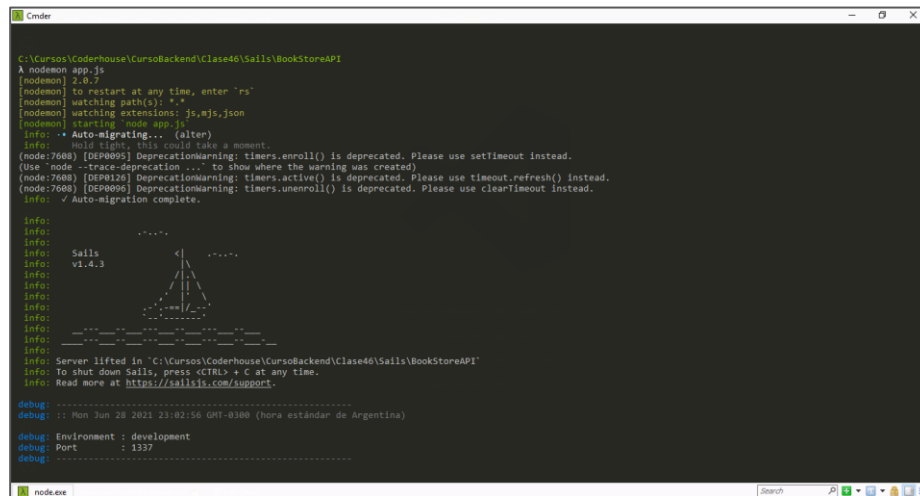
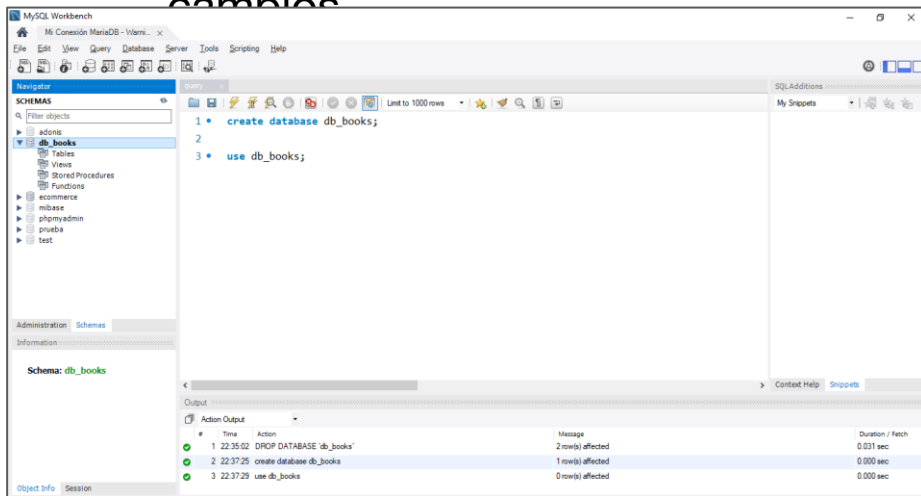
- También, vamos al archivo ***models/Book.js*** y definimos los atributos del modelo de libros.

```
6  // ...
7
8  module.exports = {
9
10   attributes: {
11
12     // PRIMITIVES
13
14     // OBJECTS
15
16     // ASSOCIATIONS
17
18     name: { type: 'string', required: true, },
19     price: { type: 'string', required: true, },
20     category: { type: 'string', required: true, },
21     author: { type: 'string', },
22     description: { type: 'string', },
23   },
24 };
25
```

Persistencia de datos en MySQL



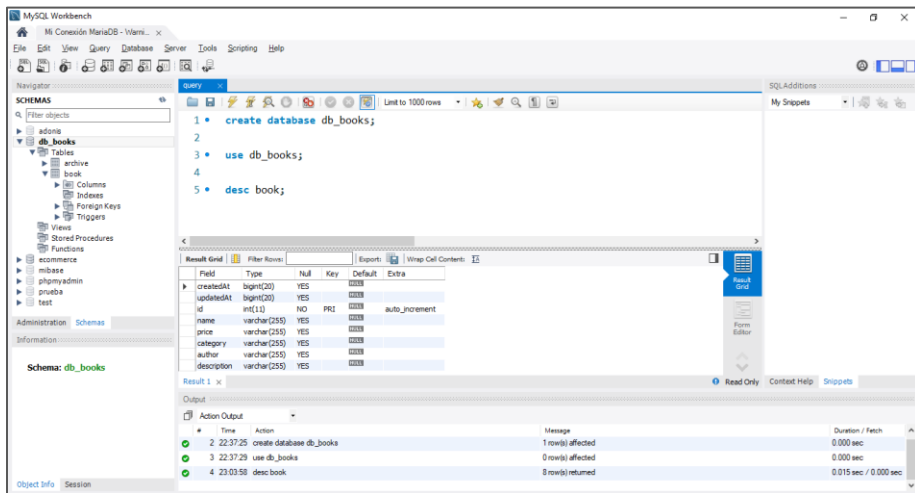
- Desde el Workbench podemos crear nuestra base de datos para usarla en nuestro proyecto.
- Finalmente, reiniciamos el servidor de nuestra aplicación para aplicar los cambios.



Persistencia de datos en MySQL



- Vemos en el Workbench que se creó la tabla de libros con las columnas especificadas en el modelo.
- Si hacemos un *select* de la tabla, vemos que todavía no tiene registros.



MySQL Workbench - Mi Conexión MariaDB - Varni...

Query:

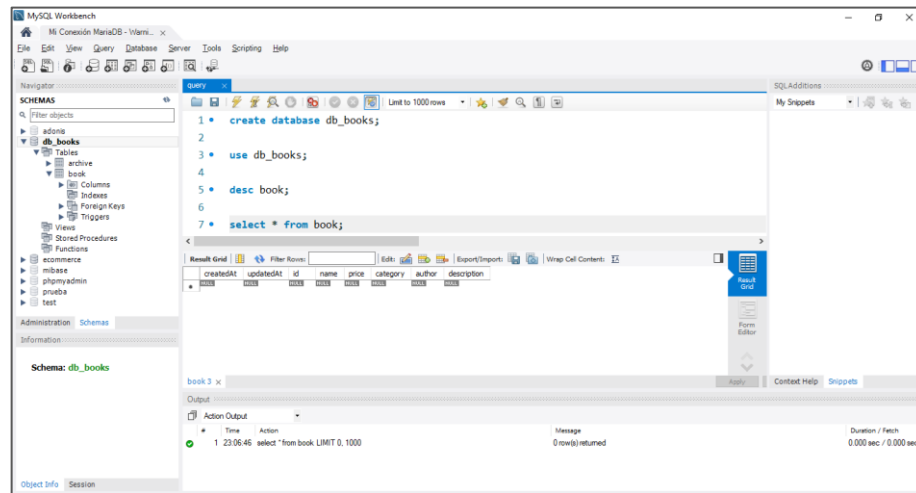
```
1 • create database db_books;
2
3 • use db_books;
4
5 • desc book;
```

Result Grid:

Field	Type	Null	Key	Default	Extra
createdAt	bigint(20)	YES			
updatedAt	bigint(20)	YES			
id	int(11)	NO	PR		auto_increment
name	varchar(255)	YES			
price	varchar(255)	YES			
category	varchar(255)	YES			
author	varchar(255)	YES			
description	varchar(255)	YES			

Action Output:

Time	Action	Message	Duration / Fetch
2 22:37:25	create database db_books	1 row(s) affected	0.000 sec
3 22:37:29	use db_books	0 row(s) affected	0.000 sec
4 23:03:58	desc book	8 row(s) returned	0.015 sec / 0.000 sec



MySQL Workbench - Mi Conexión MariaDB - Varni...

Query:

```
1 • create database db_books;
2
3 • use db_books;
4
5 • desc book;
6
7 • select * from book;
```

Result Grid:

Field	Type	Null	Key	Default	Extra
createdAt	bigint(20)	YES			
updatedAt	bigint(20)	YES			
id	int(11)	NO	PR		auto_increment
name	varchar(255)	YES			
price	varchar(255)	YES			
category	varchar(255)	YES			
author	varchar(255)	YES			
description	varchar(255)	YES			

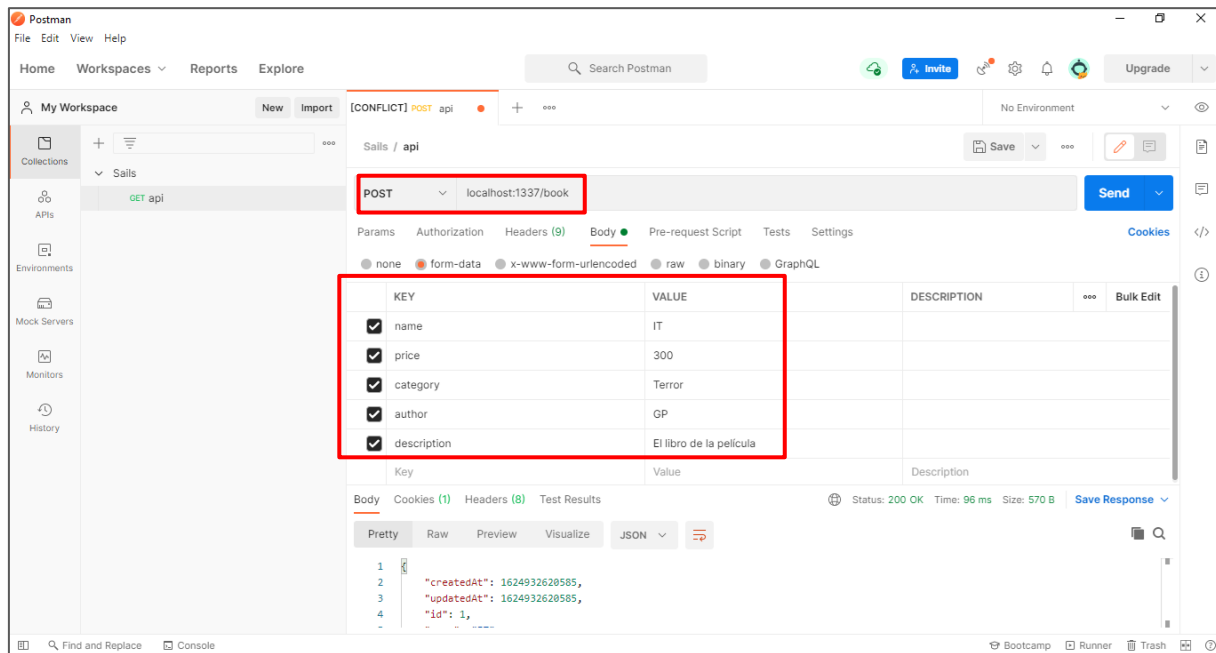
Action Output:

Time	Action	Message	Duration / Fetch
1 23:06:45	select * from book LIMIT 0, 1000	0 row(s) returned	0.000 sec / 0.000 sec

Persistencia de datos en MySQL



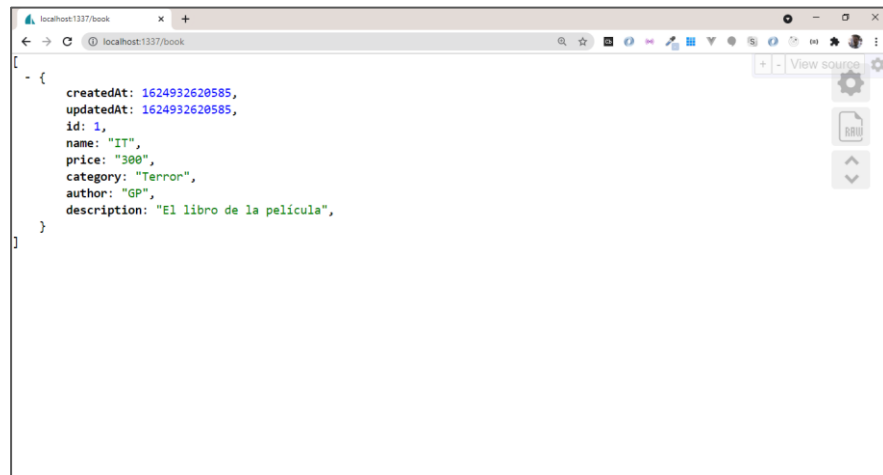
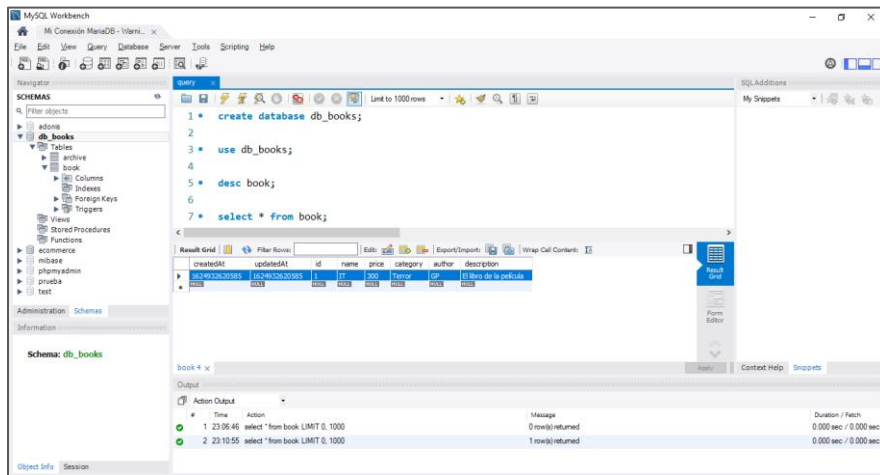
- Si volvemos a Postman, podemos probar la ruta “/book” por POST para crear un libro nuevo. Le pasamos en el body todas las propiedades.



Persistencia de datos en MySQL



- Si volvemos a hacer un select de la tabla de libros en el Workbench, ahora nos muestra el libro que acabamos de agregar.
- Además, lo podemos ver en el navegador, en la URL: <http://localhost:1337/book>.

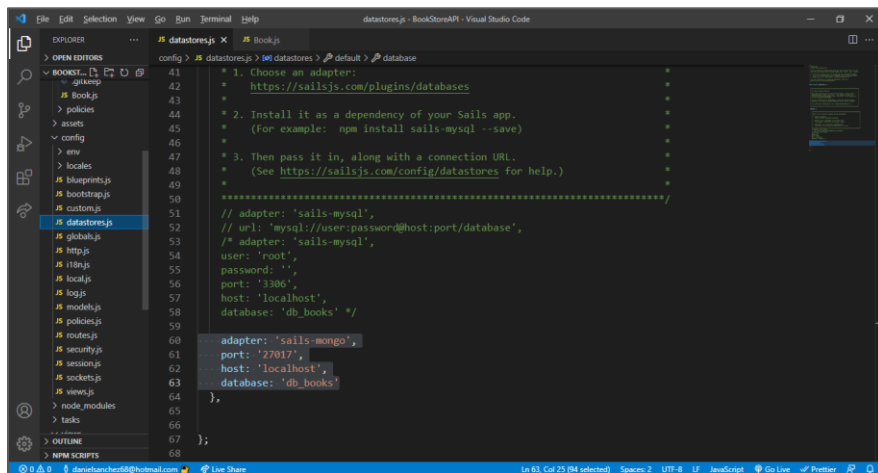


PERSISTENCIA EN MONGODB

Persistencia de datos en MongoDB



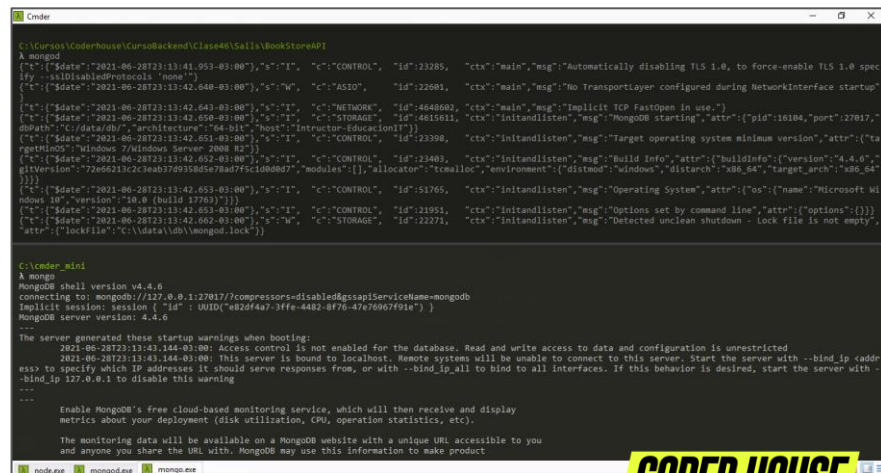
- Realizamos ahora lo mismo pero con la persistencia en una base de datos en MongoDB. Para eso, ingresamos nuevamente al archivo de **datastores.js** y configuramos los datos de la base de datos de Mongo (adapter: sails-mongo)
- Además, en la consola iniciamos Mongo.



```

1  * 1. Choose an adapter:
2  * https://sailsjs.com/plugins/databases
3
4  * 2. Install it as a dependency of your Sails app.
5  * (For example: npm install sails-mysql --save)
6
7  * 3. Then pass it in, along with a connection URL.
8  * (See https://sailsjs.com/config/datastores for help.)
9
10 *****
11 // adapter: 'sails-mysql',
12 // url: 'mysql://user:password@host:port/database',
13 // adapter: 'sails-mysql',
14 user: 'root',
15 password: '',
16 port: '3306',
17 host: 'localhost',
18 database: 'db_books' */
19
20 adapter: 'sails-mongo',
21 port: '27017',
22 host: 'localhost',
23 database: 'db_books'
24 },
25
26 node_modules
27 tasks
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68

```



```

C:\Cursos\Coderhouse\CursoBackend\Class40\Sails\BookStoreAPI
A mongod
{"t":{"$date":"2021-06-28T23:13:41.951-03:00"},"s":"I", "c":"CONTROL", "id":23285, "ctx":{"main","msg":"Automatically disabling TLS 1.0, to force-enable TLS 1.0 spec
ify --sslDisabledProtocols 'none'"
},"$":{"$date":"2021-06-28T23:13:42.640-03:00"},"s":"W", "c":"ASIO", "id":22901, "ctx":{"main","msg":"No TransportLayer configured during NetworkInterface startup"
},"$":{"$date":"2021-06-28T23:13:42.643-03:00"},"s":"I", "c":"NETWORK", "id":4648602, "ctx":{"main","msg":"Implicit TCP Fastopen in use."
},"$":{"$date":"2021-06-28T23:13:42.650-03:00"},"s":"I", "c":"STORAGE", "id":4615611, "ctx":{"initandlisten","msg":"MongoDB starting","attr":{"pid":16104,"port":27017,
"dirpath":"/data/db","architecture":"64-bit","host":"Instructor-Education1"}
},"$":{"$date":"2021-06-28T23:13:42.651-03:00"},"s":"I", "c":"CONTROL", "id":23398, "ctx":{"initandlisten","msg":"Target operating system minimum version","attr":{"ta
rgetMinOS":"Windows 7/Windows Server 2008 R2"}
},"$":{"$date":"2021-06-28T23:13:42.652-03:00"},"s":"I", "c":"CONTROL", "id":23403, "ctx":{"initandlisten","msg":"Build info","attr":{"buildInfo":{"version":"4.4.6","
gitVersion":"72e66213c2c3eab37d9358d578ad7f5c1d0d0d7","modules":[],"allocator":"tcmalloc","environment":{"distmod":"windows","distarch":"x86_64","target_arch":"x86_64"
}}}}
},"$":{"$date":"2021-06-28T23:13:42.653-03:00"},"s":"I", "c":"CONTROL", "id":51765, "ctx":{"initandlisten","msg":"Operating System","attr":{"os":{"name":"Microsoft Wi
ndows 10","version":"10.0 (build 17763)"}}}
},"$":{"$date":"2021-06-28T23:13:42.653-03:00"},"s":"I", "c":"CONTROL", "id":21951, "ctx":{"initandlisten","msg":"Options set by command line","attr":{"options":{"}}}
},"$":{"$date":"2021-06-28T23:13:42.662-03:00"},"s":"W", "c":"STORAGE", "id":22271, "ctx":{"initandlisten","msg":"Detected unclean shutdown - lock file is not empty,
attr":{"lockFile":"/C:/data/db/mongod.lock"}}
}}}

C:\vander_mini
A mongod
MongoDB shell version v4.6
connecting to: mongodb://127.0.0.1:27017/?compressors=disabled&gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("e820f4a7-3ffe-4482-8f76-47e76967f91e") }
MongoDB server version: 4.6.0

The server generated these startup warnings when booting:
2021-06-28T23:13:43.144-03:00: Access control is not enabled for the database. Read and write access to data and configuration is unrestricted
2021-06-28T23:13:43.144-03:00: This server is bound to localhost. Remote systems will be unable to connect to this server. Start the server with --bind_ip addr-
ess to specify which IP addresses it should serve responses from, or with --bind_ip_all to bind to all interfaces. If this behavior is desired, start the server with
--bind_ip 127.0.0.1 to disable this warning

Enable MongoDB's free cloud-based monitoring service, which will then receive and display
metrics about your deployment (disk utilization, CPU, operation statistics, etc).

The monitoring data will be available on a MongoDB website with a unique URL accessible to you
and anyone you share the URL with. MongoDB may use this information to make product

```

Persistencia de datos en MongoDB

- Luego, vamos a la consola e instalamos el módulo sails-mongo.
- Una vez instalado, iniciamos nuevamente el servidor.

```
debug: .....
debug: :: Mon Jun 28 2021 23:02:56 GMT-0300 (hora estándar de Argentina)

debug: Environment : development
debug: Port          : 1337
debug: .....
^C¿Desea terminar el trabajo por lotes (S/N)? s

C:\Cursos\Coderhouse\CursoBackend\Clase46\Sails\BookStoreAPI
λ npm i -s sails-mongo
+ sails-mongo@2.0.0
added 17 packages from 11 contributors in 7.416s

8 packages are looking for funding
  run `npm fund` for details

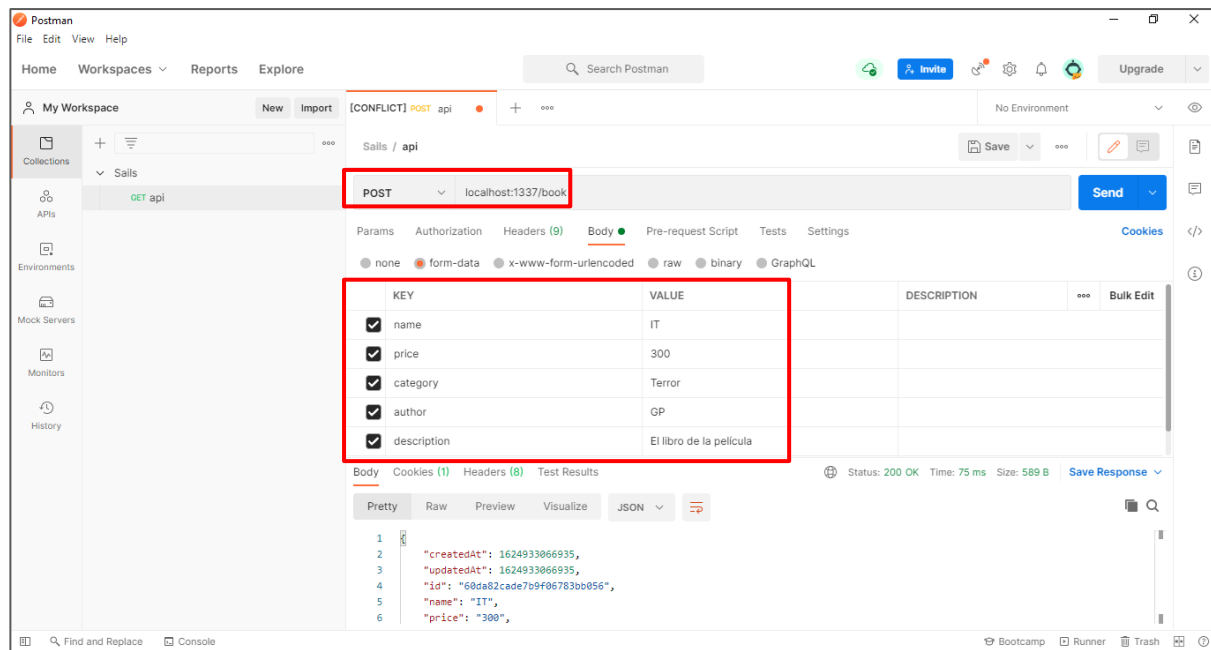
C:\Cursos\Coderhouse\CursoBackend\Clase46\Sails\BookStoreAPI
λ node app.js
[nodemon] 2.0.7
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting 'node app.js'
debug: It looks like the default datastore for this app is 'sails-mongo',
debug: but the default primary key attribute ('id') is not set up correctly.
debug: When using 'sails-mongo', primary keys MUST have 'columnName: '_id'',
debug: and must _not_ have 'autoIncrement: true'.
debug: Also, if 'dontUseObjectIds' is not set to 'true' for the model,
debug: then the 'type' of the primary key must be 'string'.
debug: We'll set this up for you this time...
debug:
info: •• Auto-migrating... (alter)
info: Hold tight, this could take a moment.
info: ✓ Auto-migration complete.

info:
info: .....
info:
```

Persistencia de datos en MongoDB



- Vamos entonces a Postman, y nuevamente agregamos un libro utilizando la ruta “/book” por POST.



Persistencia de datos en MongoDB



- Finalmente, podemos ver el libro agregado ingresando los comandos de Mongo en la consola (`db.book.find()`) o entrando en el navegador a la ruta `"/book"`.

```
C:\Users\CoderHouse\CursosBackend\Clase40\Sails\BookStoreAPI
A mongod
{"t":{"date":"2021-06-28T23:13:41.953-03:00"},"s":"I", "c":"CONTROL", "id":23285, "ctx":"main","msg":"Automatically disabling TLS 1.0, to force-enable TLS 1.0 spec
ify --sslDisabledProtocols 'none'"}
{"t":{"date":"2021-06-28T23:13:42.640-03:00"},"s":"W", "c":"ASIO", "id":22681, "ctx":"main","msg":"No TransportLayer configured during NetworkInterface startup"}
{"t":{"date":"2021-06-28T23:13:42.643-03:00"},"s":"I", "c":"NETWORK", "id":4648002, "ctx":"main","msg":"Implicit TCP Fastopen in use."}
{"t":{"date":"2021-06-28T23:13:42.650-03:00"},"s":"I", "c":"STORAGE", "id":4615611, "ctx":"initandlisten","msg":"MongoDB starting", "attr":{"pid":16184,"port":27017,
dbPath":"C:/data/db/","architecture":"64-bit","host":"Instructor-EducacionIT"}}
{"t":{"date":"2021-06-28T23:13:42.651-03:00"},"s":"I", "c":"CONTROL", "id":23398, "ctx":"initandlisten","msg":"Target operating system minimum version","attr":{"ta
rgetOS":"Windows /Windows Server 2008 R2"}}
{"t":{"date":"2021-06-28T23:13:42.652-03:00"},"s":"I", "c":"CONTROL", "id":23403, "ctx":"initandlisten","msg":"Build info","attr":{"buildInfo":{"version":"4.4.6",
platform":"726e6213c23eab3709358d5e78ad7f5c1d080d7","modules":[],"allocator":"tcmalloc","environment":{"distro":"windows","distarch":"x86_64","target_arch":"x86_64"
}}}}
{"t":{"date":"2021-06-28T23:13:42.653-03:00"},"s":"I", "c":"CONTROL", "id":51765, "ctx":"initandlisten","msg":"Operating System","attr":{"os":{"name":"Microsoft Wi
ndows NT","version":"10.0 (build 17763)}}}
{"t":{"date":"2021-06-28T23:13:42.653-03:00"},"s":"I", "c":"CONTROL", "id":21951, "ctx":"initandlisten","msg":"Options set by command line","attr":{"options":{}}}
{"t":{"date":"2021-06-28T23:13:42.662-03:00"},"s":"W", "c":"STORAGE", "id":22271, "ctx":"initandlisten","msg":"Detected unclean shutdown - lock file is not empty",
"attr":{"lockFile":"C:/data/vdb/\\mongod.lock"}}

To enable free monitoring, run the following command: db.enableFreeMonitoring()
To permanently disable this reminder, run the following command: db.disableFreeMonitoring()

---
> show dbs
admin 0.000GB
config 0.000GB
local 0.000GB
mibase 0.000GB
> show dbs
admin 0.000GB
config 0.000GB
db_books 0.000GB
local 0.000GB
mibase 0.000GB
> use db_books
switched to db db_books
> db.book.find()
{ "_id" : ObjectId("60da82cade7b9f06783bbe856"), "name" : "IT", "price" : "300", "category" : "Terror", "author" : "GP", "description" : "El libro de la pelicula", "crea
tedAt" : 1624933066935, "updatedAt" : 1624933066935 }
> }
```

```
localhost:1337/book
localhost:1337/book

[
  {
    createdAt: 1624933066935,
    updatedAt: 1624933066935,
    id: "60da82cade7b9f06783bbe856",
    name: "IT",
    price: "300",
    category: "Terror",
    author: "GP",
    description: "El libro de la pelicula"
  }
]
```

SAILS CON REST BLUEPRINT



Rutas BluePrint



Cuando corremos el comando `sails lift` con BluePrint habilitado, el framework inspecciona sus modelos y configuración para vincular ciertas rutas automáticamente. Estas rutas de BluePrint implícitas permiten que nuestra aplicación responda a ciertas solicitudes sin que tengamos que vincular esas rutas manualmente.



Rutas BluePrint



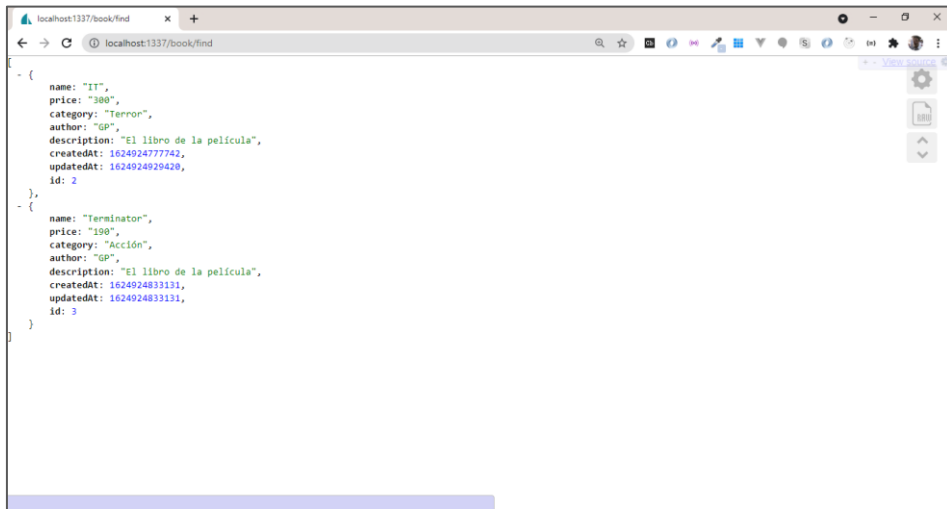
- **REST BluePrint** son las rutas generadas automáticamente que utiliza Sails para exponer una **API REST** convencional para un modelo, incluidas las acciones de búsqueda, creación, actualización y destrucción. Estas rutas son todas por **GET**.
- Dependiendo del estilo de la aplicación que generemos, las rutas de REST BluePrint pueden estar habilitadas de manera predeterminada y podrían ser adecuadas para su uso en un escenario de producción, siempre que estén protegidas por políticas para evitar el acceso no autorizado.



Rutas BluePrint



- Estas rutas de BluePrint que nos proporciona Sails son rutas de acceso directo, casi siempre exclusivas para el ambiente de desarrollo.



```
- {
  name: "IT",
  price: "300",
  category: "Terror",
  author: "GP",
  description: "El libro de la película",
  createdAt: 1624924777742,
  updatedAt: 1624924929420,
  id: 2
},
- {
  name: "Terminator",
  price: "190",
  category: "Acción",
  author: "GP",
  description: "El libro de la película",
  createdAt: 1624924833131,
  updatedAt: 1624924833131,
  id: 3
}
```

- Entonces, para probar cada una de ellas, como son todas por GET, podemos ir al navegador e ingresar las distintas URLs.
- Para **buscar** todos los libros almacenados, podemos ingresar a la URL:

<http://localhost:1337/book/find>



Rutas BluePrint



- Para buscar un libro por su id ingresamos a la URL:
<http://localhost:1337/book/find/2>.

A screenshot of a web browser window. The address bar shows the URL `localhost:1337/book/find/2`. The page content displays a JSON object representing a book:

```
{
  name: "IT",
  price: "300",
  category: "Terror",
  author: "GP",
  description: "El libro de la película",
  createdAt: 1624924777742,
  updatedAt: 1624924929420,
  id: 2
}
```



Rutas BluePrint



- También podemos mediante estas rutas crear, modificar o eliminar elementos.
- En el caso de crear nuevo libro, le pasamos como queries en la url todos los atributos del body.

```
{
  "name": "Nico",
  "price": "400",
  "category": "Accion",
  "author": "FD",
  "description": "Mucha Accion!",
  "createdAt": 1624926076437,
  "updatedAt": 1624926076437,
  "id": 4
}
```

- Para **crear** un nuevo libro, la URL que usamos es como:
<http://localhost:1337/book/create?name=Nico&price=400&category=Accion&author=FD&Mucha%20Accion!&createdAt=fecha&updatedAt=fecha>.



Rutas BluePrint



- Si vamos nuevamente a la ruta “/book/find” ahora tenemos también el nuevo libro creado.

```
[
  {
    name: "IT",
    price: "300",
    category: "Terror",
    author: "GP",
    description: "El libro de la película",
    createdAt: 1624924777742,
    updatedAt: 1624924929420,
    id: 2
  },
  {
    name: "Terminator",
    price: "190",
    category: "Acción",
    author: "GP",
    description: "El libro de la película",
    createdAt: 1624924833131,
    updatedAt: 1624924833131,
    id: 3
  },
  {
    name: "Nico",
    price: "400",
    category: "Accion",
    author: "FD",
    description: "Mucha Accion!",
    createdAt: 1624926076437,
    updatedAt: 1624926076437,
    id: 4
  }
]
```



Rutas BluePrint



- Para **modificar** un libro con las rutas de acceso directo, la ruta de la URL es `"/book/update/id?ATRIBUTOS"`. Le pasamos también por query todos los atributos con sus correspondientes valores modificados.

```
localhost:1337/book/find
localhost:1337/book/update/2?name=Terminator 2&price=230&category=Accion&author=GP&description=El libro de la p

{
  name: "IT",
  price: "300",
  category: "Terror",
  author: "GP",
  description: "El libro de la película",
  createdAt: 1624024777742,
  updatedAt: 1624024029420,
  id: 2
},
{
  name: "Terminator",
  price: "190",
  category: "Acción",
  author: "GP",
  description: "El libro de la película",
  createdAt: 1624024833133,
  updatedAt: 1624024833133,
  id: 3
},
{
  name: "Nico",
  price: "400",
  category: "Accion",
  author: "FD",
  description: "Mucha Accion!",
  createdAt: 1624026076437,
  updatedAt: 1624026076437,
  id: 4
}
```

```
localhost:1337/book/update/2?name=Terminator 2&price=230&category=Accion&author=GP&description=El libro de la p
{
  name: "Terminator 2",
  price: "230",
  category: "Accion",
  author: "GP",
  description: "El libro de la película",
  createdAt: 1624024777742,
  updatedAt: 1624026708019,
  id: 2
}
```



Rutas BluePrint



- Si volvemos a la ruta “/book/find” vemos ahora primer libro modificado con los nuevos valores de los atributos que le pasamos.

```
[
  {
    name: "Terminator 2",
    price: "230",
    category: "Accion",
    author: "GP",
    description: "El libro de la película",
    createdAt: 1624924777742,
    updatedAt: 1624926708018,
    id: 2
  },
  {
    name: "Terminator",
    price: "190",
    category: "Acción",
    author: "GP",
    description: "El libro de la película",
    createdAt: 1624924833131,
    updatedAt: 1624924833131,
    id: 3
  },
  {
    name: "Nico",
    price: "400",
    category: "Accion",
    author: "FD",
    description: "Mucha Accion!",
    createdAt: 1624926076437,
    updatedAt: 1624926076437,
    id: 4
  }
]
```




Rutas BluePrint



- Para **eliminar** un libro, la ruta que usamos es “/book/**destroy**/id”.
- Si luego volvemos a “/book/**find**” vemos que ya no está el libro que eliminamos.

```
{
  name: "Terminator",
  price: "590",
  category: "Acción",
  author: "GP",
  description: "El libro de la película",
  createdAt: 1624924833133,
  updatedAt: 1624924833133,
  id: 3,
}
```

```
- {
  name: "Terminator 2",
  price: "230",
  category: "Acción",
  author: "GP",
  description: "El libro de la película",
  createdAt: 1624924777742,
  updatedAt: 1624926780010,
  id: 2
},
- {
  name: "Nico",
  price: "400",
  category: "Acción",
  author: "EP",
  description: "Mucha Acción!",
  createdAt: 1624920876437,
  updatedAt: 1624920876437,
  id: 4
}
}
```



Rutas BluePrint



En resumen, las rutas de BluePrint que nos proporciona Sails son:

Ruta	Acción
GET <code>/:modelIdentity/find</code>	Find (buscar).
GET <code>/:modelIdentity/find/:id</code>	findOne (buscar uno).
GET <code>/:modelIdentity/create?.....</code>	Create (crear). Atributos por query.
GET <code>/:modelIdentity/update/:id?...</code>	Update(modificar). Atributos por query.
GET: <code>/:modelIdentity/destroy/:id</code>	Destroy (eliminar uno).



USANDO SAILS

Tiempo: 15 minutos

USANDO SAILS

Desafío
generico



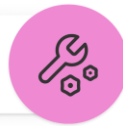
Tiempo: 15 minutos

Crear un proyecto backend basado en el framework Sails para realizar consultas y almacenamiento de películas por nombre, género y año de estreno. El sistema de persistencia será el default del framework (sistema de archivos).

- Estableceremos las rutas API Rest (Blueprint) para poder:
 - ◆ Listar todas y cada una de las películas por su id
 - ◆ Incorporar una película
 - ◆ Actualizar una película por su id
 - ◆ Borrar una película por su id
- Dichos request serán realizados mediante Postman, comprobando que todos las funciones trabajen correctamente.

USANDO SAILS

Desafío
generico



Tiempo: 15 minutos

Luego, probar la API Rest a través de requests get que nos permiten las rutas blueprint de Sails utilizando los parámetros:

- /find
- /find/:id
- /create?nombre=...&genero=...&anio=...
- /update/:id?nombre=...&genero=...&anio=...
- /destroy/:id

Reiniciar el servidor y verificar la persistencia de los datos.



BREAK

¡5/10 MINUTOS Y VOLVEMOS!

KOA



¿De qué se trata?

- **Koa** es otro framework para Node.
- Podemos decir que es Express mejorado, ya que es mucho más ligero, más rápido y mucho más sólido.
- Es desarrollado por el mismo equipo de Express, sin embargo esta vez tomaron un enfoque diferente aprovechándose de las nuevas características de JavaScript.
- Tiene características que ayudan a los desarrolladores de JavaScript que desean usar y aprovechar Node para acelerar el desarrollo de API y aplicaciones web.

Características

- **Libre de middlewares:** Muchas veces no necesitamos el paquete entero de características, esto nos permite tener un servidor rápido y podemos ir añadiendo características a medida que las necesitamos.
- **Adiós callbacks:** Funciona completamente bien con las características de ES6, lo que permite usar promesas, async/await, que mejoran la lectura del código, y nos permite olvidarnos de los callbacks.

Características

- **Middlewares en cascada:** Aplica los middlewares de manera descendente, es decir los aplica a medida que los encuentra, y cuando devuelve respuestas vuelve a retornar por cada uno de los middlewares.
- **Manejo de errores:** Podemos tener un control más fino cuando suceden excepciones, un solo middleware es suficiente para ayudar a solventar esto.
- **Gran rendimiento:** Se ubica entre los frameworks más rápidos. Sin un core simple y ligero no podría ser posible.



Empezando con Koa



- Para empezar a desarrollar con este framework, primero debemos instalarlo. Usamos el comando: `npm install koa`.
- Podemos crear un archivo llamado `app.js` en donde crearemos una ruta que imprima “Hello World” usando Koa.

```
1 // app.js
2 const Koa = require('koa');
3 const app = new Koa();
4
5 // Our First Route
6 app.use(async ctx => {
7     ctx.body = 'Hello World';
8 });
9
10 // Bootstrap the server
11 app.listen(3000);
```

- Primero, requerimos Koa y luego, definimos la instancia `app`. Esta tiene acceso a todos los métodos que incluye la API de Koa (como `use()` y `listen()`). Es como se define la función de middleware. La estamos usando como ruta en este caso.
- Estamos usando `ctx` como argumento para la función de middleware asincrónico. Se llama Contexto en Koa y encapsula los objetos de solicitud y respuesta en un solo objeto.



Empezando con Koa



- Si iniciamos entonces el servidor en la terminal con el comando `node app.js` y vamos al navegador en <http://localhost:3000>, veremos el texto “Hello World”
- En Koa, se crea un contexto para cada solicitud que llega al servidor y siempre se hace referencia a él como middleware.

```
1  // request and response as context in Koa
2  app.use(async ctx => {
3      ctx.request;
4      ctx.response;
5  });
```

API REST CON KOA



Instalación



- Además de instalar el módulo koa, debemos instalar los módulos koa-body y koa-router. Para eso usamos los comandos de npm:

```
npm install koa    npm install koa-body    npm install koa-router
```

- koa-body** es un middleware de body-parser. Soporta solicitudes urlencoded, multi-part y json. Ayuda a crear y responder a solicitudes POST que están disponibles como un campo de formulario, o una carga de archivo, etc. Le dice al servidor que la solicitud entrante del cliente tiene un cuerpo de datos.
- koa-router** es el middleware de enrutamiento que proporciona enrutamiento estilo Express utilizando verbos HTTP. Tiene métodos que podemos usar directamente en la aplicación, como `app.get()`, `app.post()`, etc.



Archivo *app.js*



```
JS app.js x
serverKoa > JS app.js > ...
1 // app.js
2 const Koa = require('koa');
3 const koaBody = require('koa-body');
4
5 const app = new Koa();
6
7 // Set up body parsing middleware
8 app.use(koaBody());
9
10 // Require the Router we defined in books.js
11 let books = require('./books.js');
12
13 // Use the Router on the sub route /books
14 app.use(books.routes());
15
16 // Server listen
17 const PORT = 8080
18 const server = app.listen(PORT, () => {
19   console.log(`Servidor Koa escuchando en el puerto ${server.address().port}`)
20 })
21 server.on('error', error => console.log('Error en Servidor Koa:', error))
```

- Creamos nuestro archivo principal llamado ***app.js***.
- Creamos la instancia `app` de Koa. Luego seteamos el middleware de `body-parser`.
- Requerimos las rutas que vamos a crear en el archivo `books.js`. Y usamos esas rutas con `app.use()`.
- Finalmente, creamos el servidor con `app.listen()`.

Archivo books.js



JS books.js X

serverKoa > JS books.js > ...

```
1  // books.js
2  const Router = require('koa-router');
3
4  // Prefix all routes with /books
5  const router = new Router({
6    prefix: '/books'
7  });
8
9  let books = [
10   { id: 101, name: 'Fight Club', author: 'Chuck Palahniuk' },
11   { id: 102, name: 'Sharp Objects', author: 'Gillian Flynn' },
12   { id: 103, name: 'Frankenstein', author: 'Mary Shelley' },
13   { id: 104, name: 'Into The Wild', author: 'Jon Krakauer' }
14 ];
```

- Primero, en el archivo **books.js**, importamos la dependencia necesaria para crear rutas: koa-router.
- Luego creamos una instancia de esta dependencia. Vemos que el prefijo usado es: “/libros”. De esta forma categorizamos las rutas.
- El array de libros es la información mockeada. Cada objeto de libro tiene campo de id, nombre y autor.



Archivo *books.js*



```
/* API REST Get All */
router.get('/', ctx => {
  ctx.body = {
    status: 'success',
    message: books,
  }
})

/* API REST Get x ID */
router.get('/:id', ctx => {
  const getCurrentBook = books.filter(function (book) {
    if (book.id == ctx.params.id) {
      return true
    }
  })

  if (getCurrentBook.length) {
    ctx.body = getCurrentBook[0]
  } else {
    ctx.response.status = 404
    ctx.body = {
      status: 'error!',
      message: 'Book Not Found with that id!',
    }
  }
})
```

- Luego, en el mismo archivo creamos las rutas.
- Empezamos con los métodos **get** a la ruta “/” para traer todos los libros y **get** a la ruta “/:id” para traer un solo libro por su id.
- En ambos casos, en el callback de router.get() tenemos el parámetro *ctx* que reemplaza al req y res que usamos en Express.
- En la ruta de GET por id, usamos `ctx.params.id` para recuperar el id pasado por parámetro.

Archivo *books.js*



```
/* API REST Post */
router.post('/', ctx => {
  // Check if any of the data field not empty
  if (
    !ctx.request.body.id ||
    !ctx.request.body.name ||
    !ctx.request.body.author
  ) {
    ctx.response.status = 400
    ctx.body = {
      status: 'error',
      message: 'Please enter the data',
    }
  } else {
    const newBook = books.push({
      id: ctx.request.body.id,
      name: ctx.request.body.name,
      author: ctx.request.body.author,
    })
    ctx.response.status = 201
    ctx.body = {
      status: 'success',
      message: `New book added with id:
${ctx.request.body.id} & name: ${ctx.request.body.name}`,
    }
  }
})
```

- Con el método **post** a la ruta “/” creamos nuevos libros.
- Con `ctx.request.body` recuperamos los datos del libro a agregar, que llegan en el body.
- Si todo está bien, el middleware de enrutamiento acepta los datos y devuelve un mensaje de éxito con el status 201.

Archivo *books.js*



```
/* API REST Put */
router.put('/update/:id', ctx => {
  // Check if any of the data field not empty
  if (
    !ctx.request.body.id ||
    !ctx.request.body.name ||
    !ctx.request.body.author
  ) {
    ctx.response.status = 400
    ctx.body = {
      status: 'error',
      message: 'Please enter the data',
    }
  } else {
    const id = ctx.params.id
    const index = books.findIndex(book => book.id === id)
    books.splice(index, 1, ctx.request.body)
    ctx.response.status = 201
    ctx.body = {
      status: 'success',
      message: `New book updated with id:
    ${ctx.request.body.id} & name: ${ctx.request.body.name}`,
    }
  }
})
```

- Con el método ***put*** a la ruta “***/:id***” podemos modificar la información de un libro.



Archivo books.js



```
/* API REST Delete */
router.delete('/delete/:id', ctx => {
  const id = ctx.params.id
  const index = books.findIndex(book => book.id == id)
  books.splice(index, 1)
  ctx.response.status = 200
  ctx.body = {
    status: 'success',
    message: `Book deleted with id: ${id}`,
  }
})
```

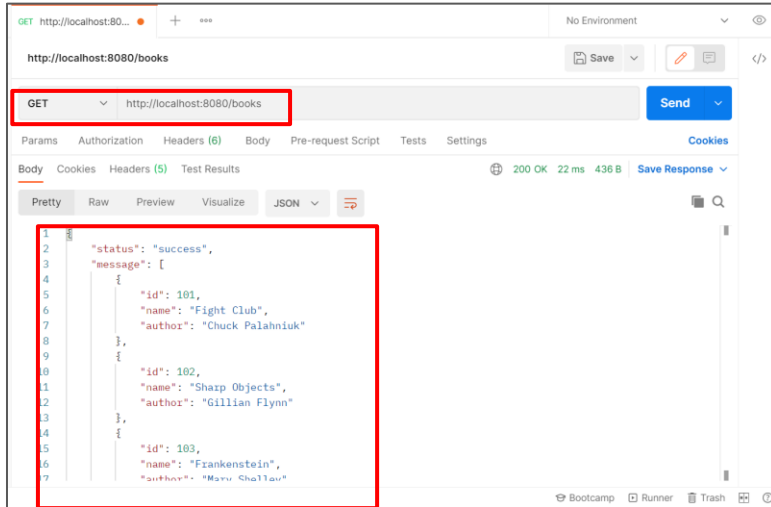
- Y finalmente, con el método ***delete*** a la ruta “***/:id***” podemos eliminar un libro del array.

Probando las rutas

Probando las rutas



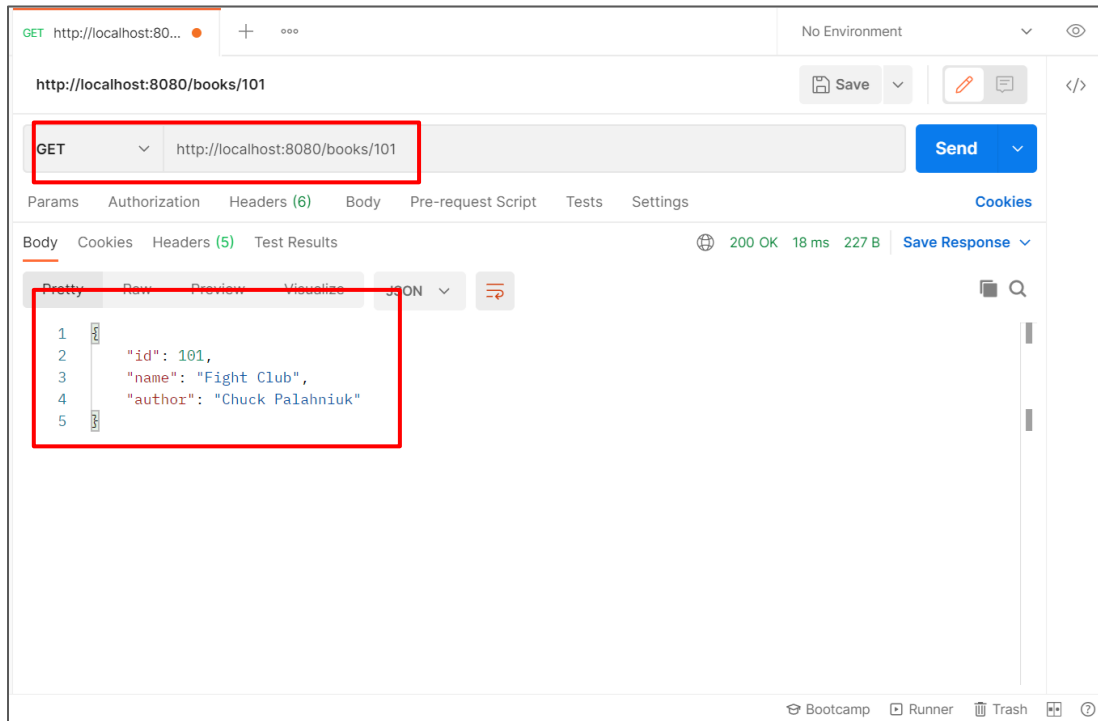
- Para probar las rutas de la API REST que creamos con Koa podemos usar Postman.
- Primero iniciamos el servidor con el comando `node app.js`. Podemos sino configurar el `package.json` para iniciar con `npm start` como siempre.
- Vamos a Postman y probamos la ruta por GET **“/books”** para traer todos los libros.



Probando las rutas



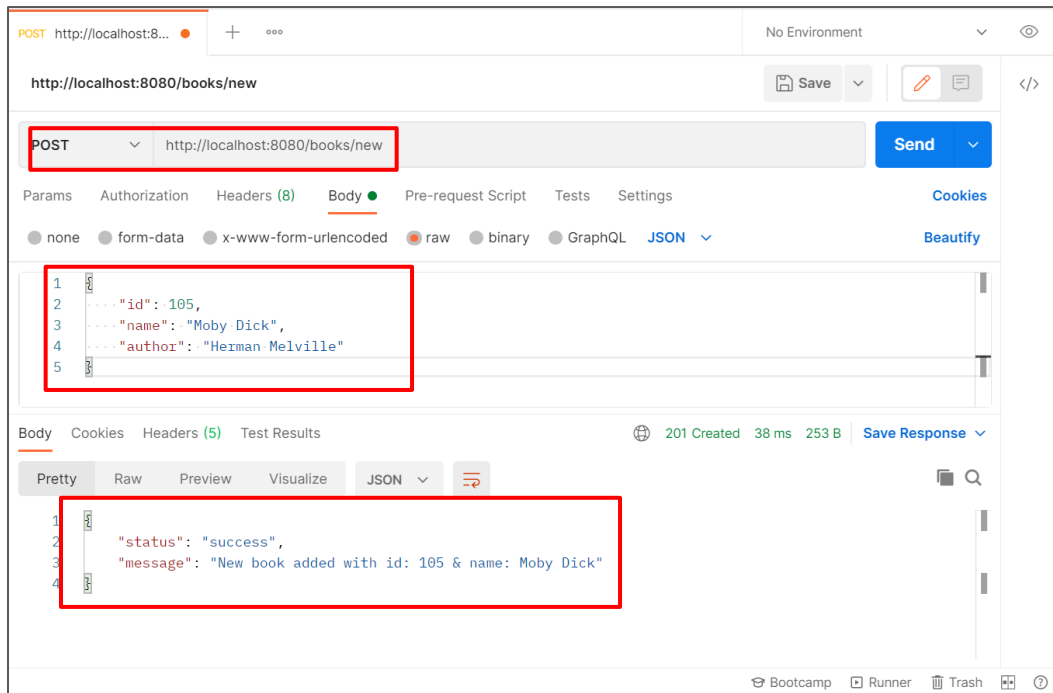
- Seguimos con la ruta por GET “/books/:id” para traer un solo libro por su id.



Probando las rutas



- Luego, agregamos un nuevo libro con la ruta por POST **“/books”**. Para eso, debemos pasarle en el body las propiedades del libro a agregar.





Probando las rutas



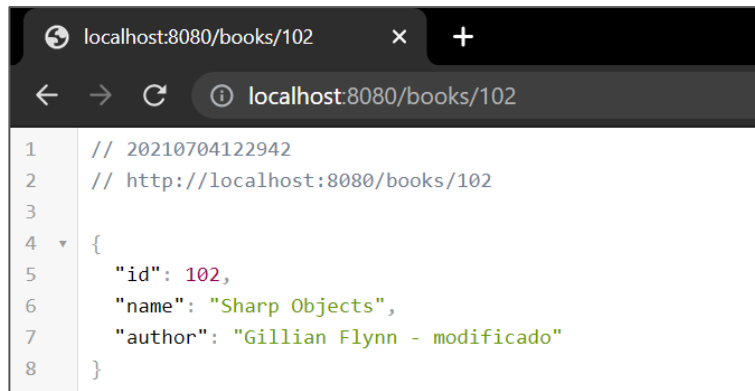
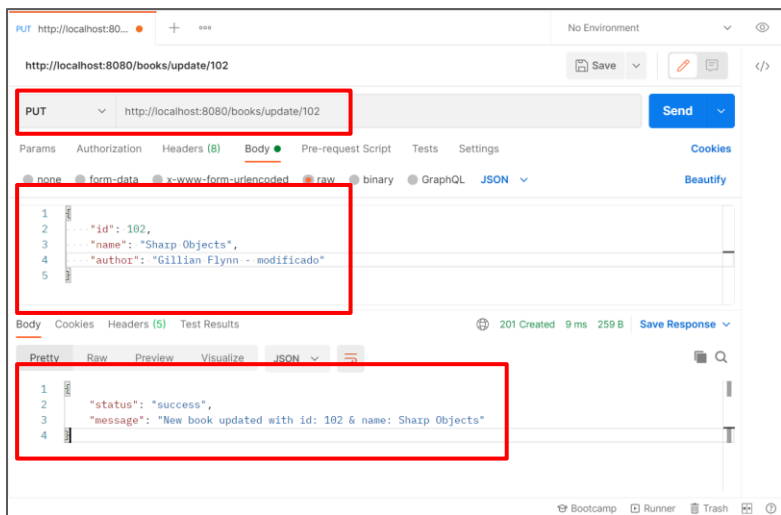
- Podemos ver entrando en el navegador, la lista de libros con el nuevo agregado.

```
localhost:8080/books
// http://localhost:8080/books
{
  "status": "success",
  "message": [
    {
      "id": 101,
      "name": "Fight Club",
      "author": "Chuck Palahniuk"
    },
    {
      "id": 102,
      "name": "Sharp Objects",
      "author": "Gillian Flynn"
    },
    {
      "id": 103,
      "name": "Frankenstein",
      "author": "Mary Shelley"
    },
    {
      "id": 104,
      "name": "Into The Wild",
      "author": "Jon Krakauer"
    },
    {
      "id": 105,
      "name": "Moby Dick",
      "author": "Herman Melville"
    }
  ]
}
```

Probando las rutas



- Probamos ahora la ruta por PUT “/books/:id” para modificar los datos de un libro. Pasamos en el body los datos del libro a modificar.
- Luego, entramos al navegador y vemos los cambios en la ruta por GET.

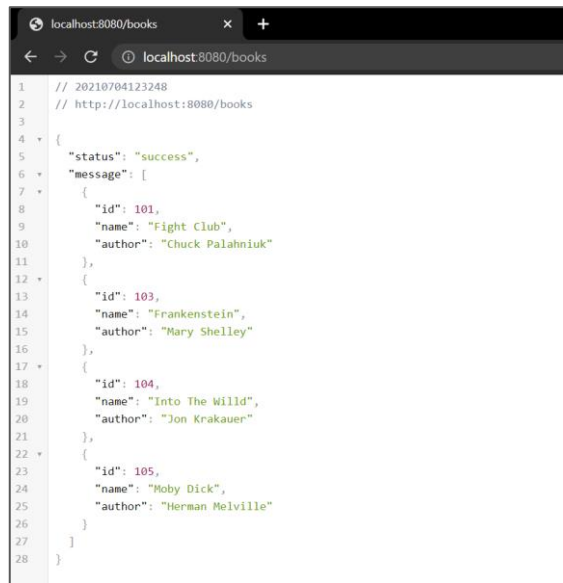
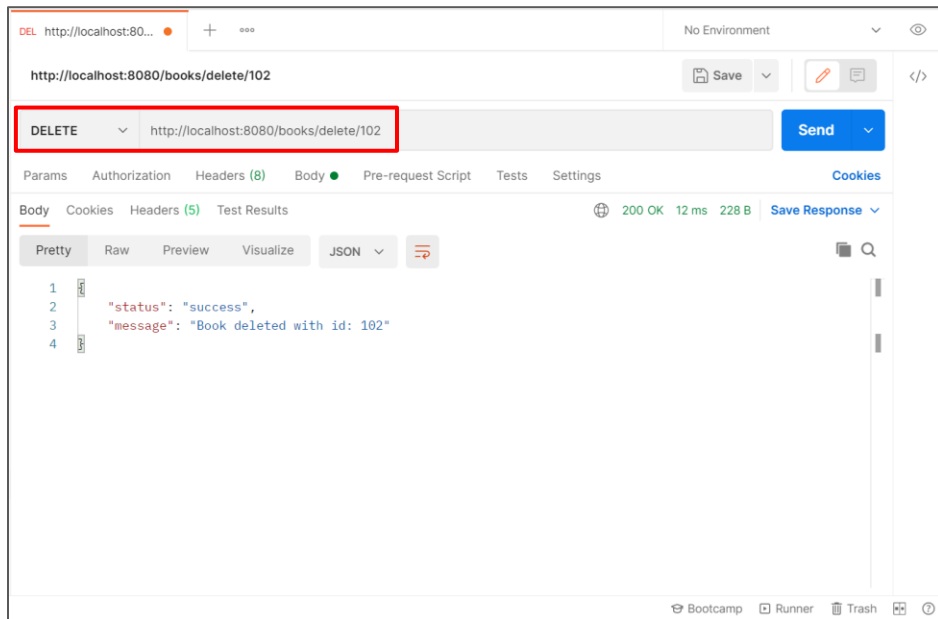




Probando las rutas



- Finalmente probamos la ruta DELETE ***“/books/:id”*** para eliminar un libro.
- Luego, vemos en el navegador, probando la ruta por GET que ya no muestra el libro eliminado.





USANDO KOA

Tiempo: 15 minutos



Tiempo: 15 minutos

Crear una aplicación API Rest con Koa, para ingresar notas de alumnos junto a su nombre, apellido, DNI y materia cursada.

- La aplicación debe ser capaz de responder:
 - ◆ El listado de todos los alumnos, con sus datos completos
 - ◆ Los datos de un alumno seleccionado por su DNI
 - ◆ El promedio que han alcanzado los alumnos de una determinada materia
- Además se podrá actualizar y borrar los datos de un alumno por su DNI
- Probar la funcionalidad con postman.

NOTA1: Para seleccionar la materia de la cual se calculará el promedio alcanzado por sus alumnos utilizar query params en Koa. Si dicha materia no existe, retornar un mensaje de error.

NOTA2: En cada request de ingreso ó modificación, validar sencillamente los datos, y en caso de no ser consistentes, dar el mensaje de error correspondiente.



REFORMAR PARA USAR OTRO FRAMEWORK

Retomar el proyecto con el que vemos trabajando para trasladarlo a uno de los frameworks presentados

REFORMAR PARA USAR OTRO FRAMEWORK

Formato: link a un repositorio en Github con el proyecto cargado.

Sugerencia: no incluir los node_modules

Desafío
entregable



>> Consigna: Elegir uno de los frameworks vistos en clase y trasladar a esta nueva plataforma el último proyecto entregable (con GraphQL) o al anterior (sin GraphQL).

→ Verificar el correcto funcionamiento del servidor a nivel de sus rutas, vistas, lógica de negocio y persistencia.

¿PREGUNTAS?





¡MUCHAS GRACIAS!

Resumen de lo visto en clase hoy:

- SailsJS
 - Koa
- 



OPINA Y VALORA ESTA CLASE

#DEMOCRATIZANDO LA EDUCACIÓN