
ICARO
Nuestro compromiso es con el futuro.

Bases de Datos

Clase 5

¿Qué veremos hoy?

Hoy vamos a continuar el módulo de **Bases de Datos**, vamos a seguir usando nuestra herramienta **Mysql Workbench**, pero introduciremos algo que ya hemos venido trabajando, a Express. El objetivo sera conectar una aplicacion, nodeJs con Express a nuestra base de datos Mysql. Mas precisamente, las rutas que soliamos crear en el modulo pasado, con Express, ahora iran hasta nuestra base de datos para realizar distintas operaciones en la misma.

Veremos a continuacion como lograrlo.

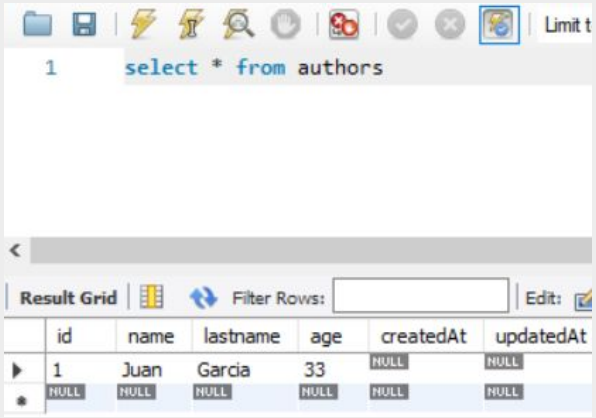
Aplicacion. Express

El módulo pasado vimos cómo desarrollar y crear nuestras propias rutas haciendo uso del modulo de Express.

```
// Controllers
const authorsControllers = require('../controllers/author');

module.exports = (app) => {

    app.get('/api', (req, res) => res.status(200).send({
        message: 'Clase 5 Icaro',
    }));

    app.get('/api/authors/find/id/:id', authorsControllers.find);

};
```

Aplicacion. Express

En la imagen presentada antes, a modo introductorio de lo que se pretende en esta clase, se genera una **ruta o route**, la cual está destinada a buscar informacion de un autor o author.
Este misma ruta, es la que se comunicara con la base de datos que hemos estado utilizando y nos mostrará información de un **author** particular, pasando como dato, su **id**.



Aplicacion. Express

Lo mostrado anteriormente es nuestra tabla, presente en la base de datos que hemos creado en Mysql Workbench.
Lo que sigue a continuación, es establecer la comunicación con nuestra base de datos, esta vez, no de forma directa con Mysql Workbench, sino que quien tiene que establecer esa comunicación, es nuestra aplicación, la que va a estar utilizando Express.

Veamos a Sequelize, la libreria que nos permitira llevar a cabo dicha tarea.

Sequelize

Sequelize es un **ORM** basado en la promesa para Node. js y io. js. Es compatible con los dialectos PostgreSQL , **MySQL** , **MariaDB** , **SQLite** y **MSSQL**. Ahora, que significa el término **ORM**?
Un **ORM** es un modelo de programación que permite mapear las estructuras de una base de datos relacional (SQL Server, Oracle, MySQL, etc.), en adelante RDBMS (Relational Database Management System), sobre una estructura lógica de entidades con el objeto de simplificar y acelerar el desarrollo de nuestras aplicaciones.

Sequelize. Instalacion

En lo que respecta a la instalación, usaremos algo ya conocido, que es nuestro gestor de paquetes NPM.

npm install --save sequelize
npm install -g sequelize-cli

Por un lado tenemos la dependencia de Sequelize y además, podemos hacer uso de una herramienta propia, incluida con Sequelize, de nombre Sequelize CLI que nos ayudará a ejecutar ciertos comandos para por ejemplo crear una representación de la tabla de lado de nuestra aplicación, modificar la misma, etc.

También, debemos instalar mysql.
npm install --save mysql2

Sequelize. Flujo de trabajo

Una vez instalado lo anterior, se debe ejecutar el comando INIT, o mejor expresado:
sequelize init

Al ejecutar este comando, siempre dentro de la carpeta de nuestro proyecto, se crearan:

```
mi-proyecto/
├── config/
│   └── config.json
├── migrations/
├── models/
│   └── index.js
├── node_modules/
├── seeders/
├── app.js
├── package-lock.json
└── package.json
```

Sequelize. Flujo de trabajo

```
{
  "development": {
    "username": "root",
    "password": "123456",
    "database": "ejemplo-sequelize",
    "host": "127.0.0.1",
    "dialect": "mysql",
    "operatorsAliases": false
  },
  "test": {
    "username": "root",
    "password": "123456",
    "database": "ejemplo-sequelize",
    "host": "127.0.0.1",
    "dialect": "mysql",
    "operatorsAliases": false
  },
  "production": {
    "username": "root",
    "password": "123456",
    "database": "ejemplo-sequelize",
    "host": "127.0.0.1",
    "dialect": "mysql",
    "operatorsAliases": false
  }
}
```

Antes se mostró una serie de archivos que son autogenerados por Sequelize. De ellos, se destaca el archivo de configuración, donde se tendra la informacion para conectarnos a la base de datos. Sequelize nos permite diferenciar ambientes, y en cada uno se debe especificar el **usuario, password, nombre de base de datos**, entre otros.

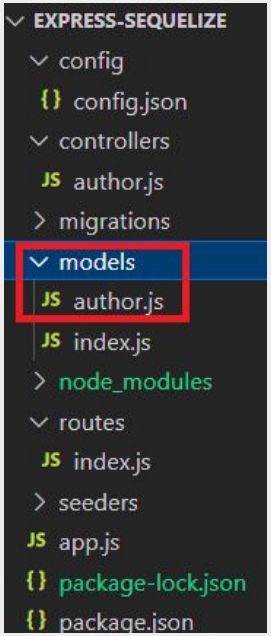
Sequelize. Creación de Modelo

Una vez ejecutado el comando **Sequelize Init** como se mostro antes, podemos crear nuestro modelo. Este representará a la Tabla que está en nuestra base de datos, mas precisamente, a nuestra tabla Authors.
El comando será el siguiente:

```
sequelize model:create --name authors --attributes name:string,lastname:string,age:integer
```

Aqui, le estamos indicando a Sequelize que cree nuestro modelo, de nombre authors, con los campos que se especifican. Al ejecutar esta instrucción, se creará una carpeta de nombre **model** dentro de nuestra aplicación.

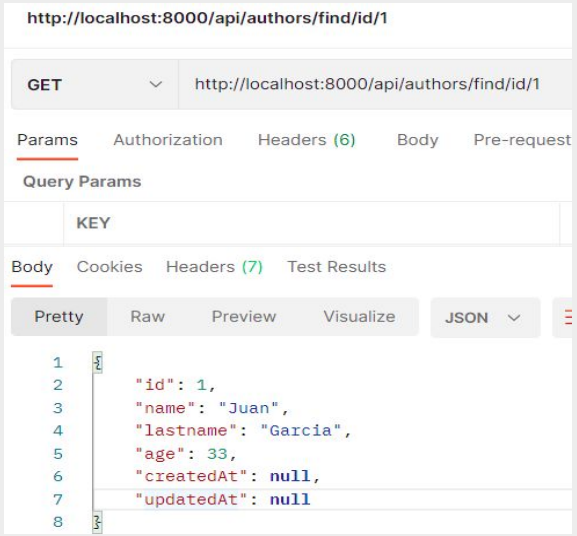
Sequelize. Creación de Modelo



Vemos como Sequelize, al ejecutar el comando se crea la carpeta models con nuestro modelo **authors.js**.

Practica. Express. Sequelize. Mysql

Para unir todos estos conceptos, llevaremos a cabo la creación de una ruta, que nos devolvera, al especificar un id de authors, un **author** presente en nuestra base de datos.



Practica. Express. Sequelize. Mysql

El metodo find, dentro del **controlador o controller**, invocará al siguiente método:

```
onst author = require('../models').author;

odule.exports = {

  find(req, res) {
      return author
          .findOne({
              where: {
                  id: req.params.id
              }
          })
          .then(author => res.status(200).send(author))
          .catch(error => res.status(400).send(error))
  }
```

Practica. Express. Sequelize. Mysql

La ruta, como se mostro sera:
http://localhost:8000/api/authors/find/id/1

Con lo cual se buscara al author de id 1. Nuestro controlador, podria quedar expresado como:

```
// Controllers
const authorsControllers = require('../controllers/author');

module.exports = (app) => {

    app.get('/api', (req, res) => res.status(200).send({
        message: 'Clase 5 Icaro',
    }));

    app.get('/api/authors/find/id/:id', authorsControllers.find);

};
```

Practica. Express. Sequelize. Mysql

Vemos que se importa el modelo que nos creó Sequelize, y que de ahí se ejecuta el método findOne. Este metodo, al recibir el id pasado a través de la ruta, ejecutara una sentencia SELECT sobre nuestra tabla authors, y finalmente nos devolverá al **author** si el mismo está en la base de datos.

```
$ npm start

> clase5@1.0.0 start
> node app.js

Executing (default): SELECT `id`, `name`, `lastname`, `age`, `createdAt`, `updatedAt`
FROM `authors` AS `author` WHERE `author`.`id` = '1';
GET /api/authors/find/id/1 200 61.646 ms - 85
```

No olvidemos que disponemos de clases de consulta!!

ICARO Asociación Civil
CUIT 30716564815
info@icaro.org.ar
www.icaro.org.ar

Muchas gracias!

ICARO Asociación Civil
CUIT 30716564815
info@icaro.org.ar
www.icaro.org.ar