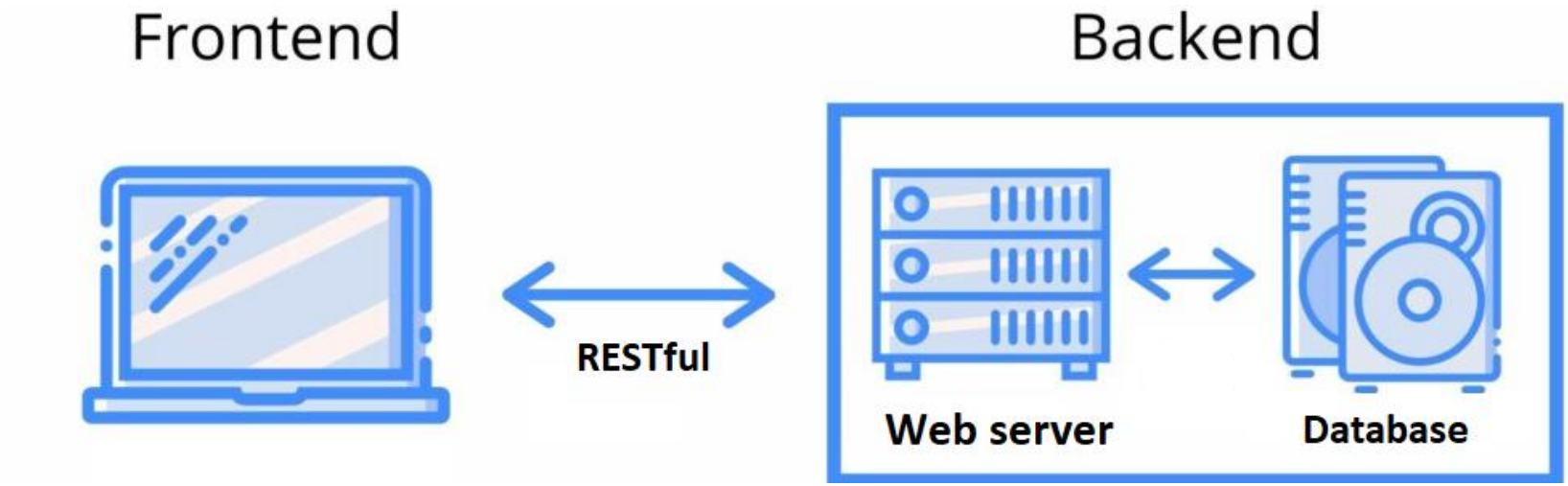
Introducción a Express + Sequelize

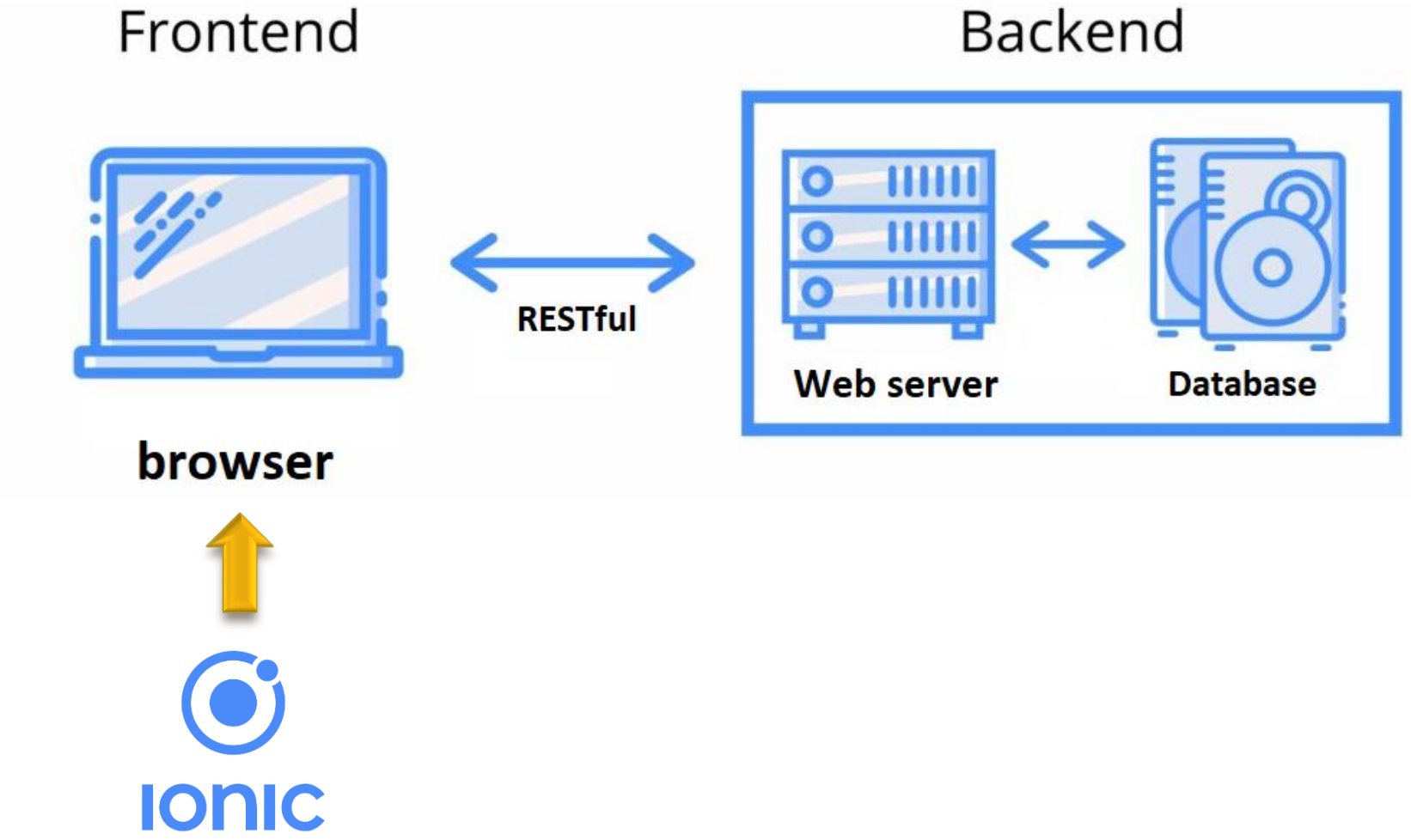
Resumen de pasos basado en la web:

https://www.bezkoder.com/node-js-express-sequelize-mysql/

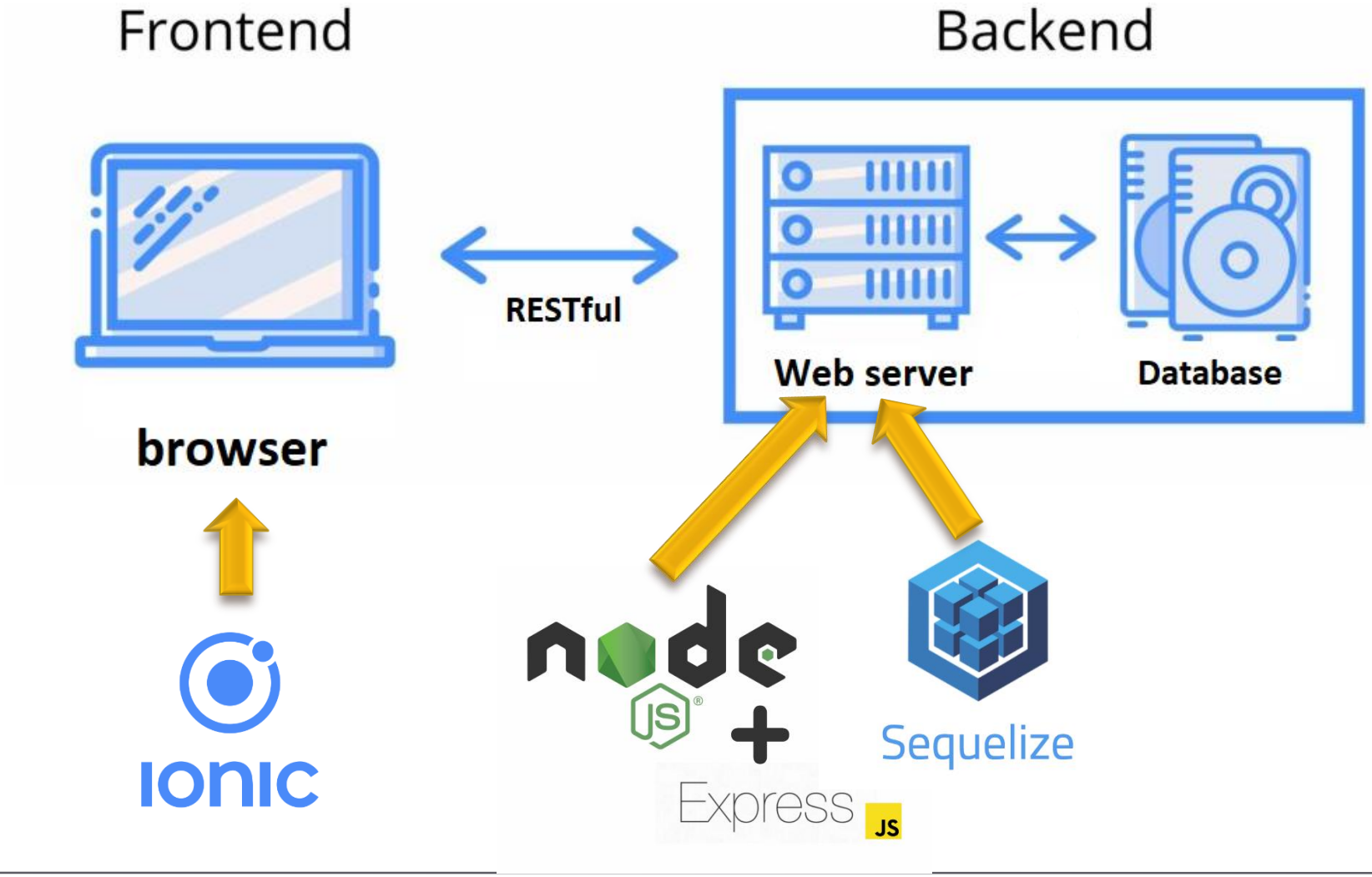
Dónde corre Ionic?



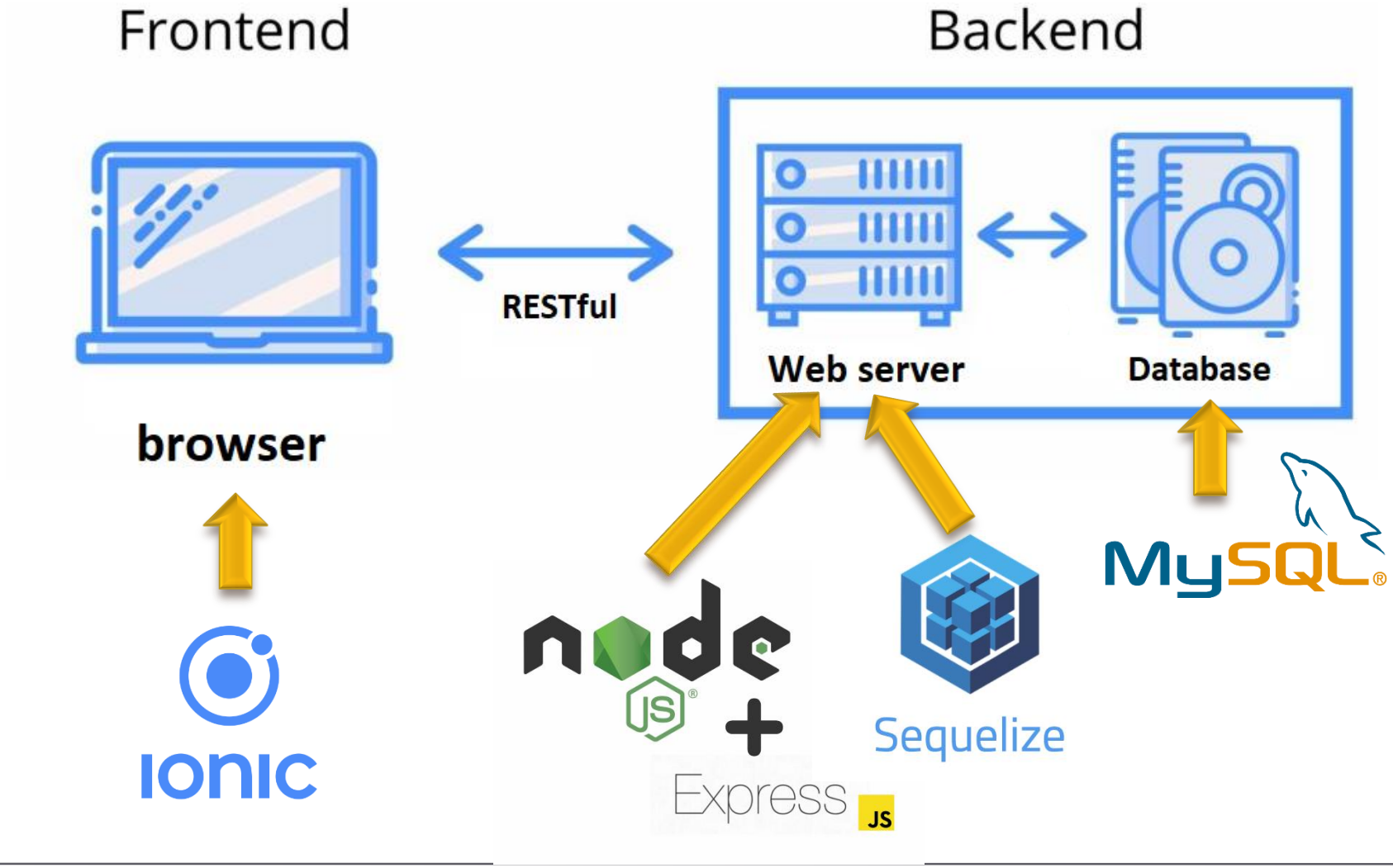
Dónde corren Express & Sequelize?



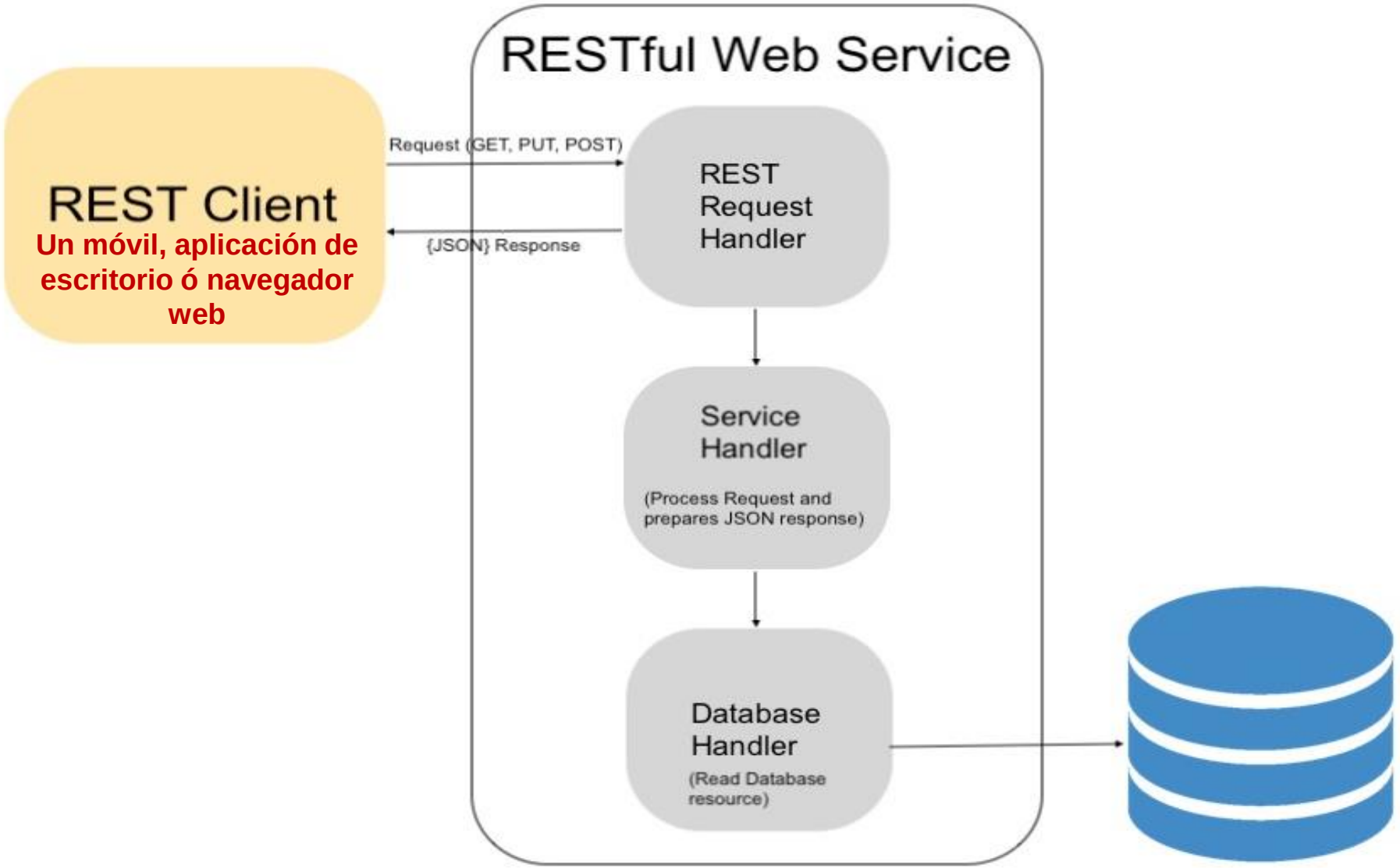
Dónde corre mysql?



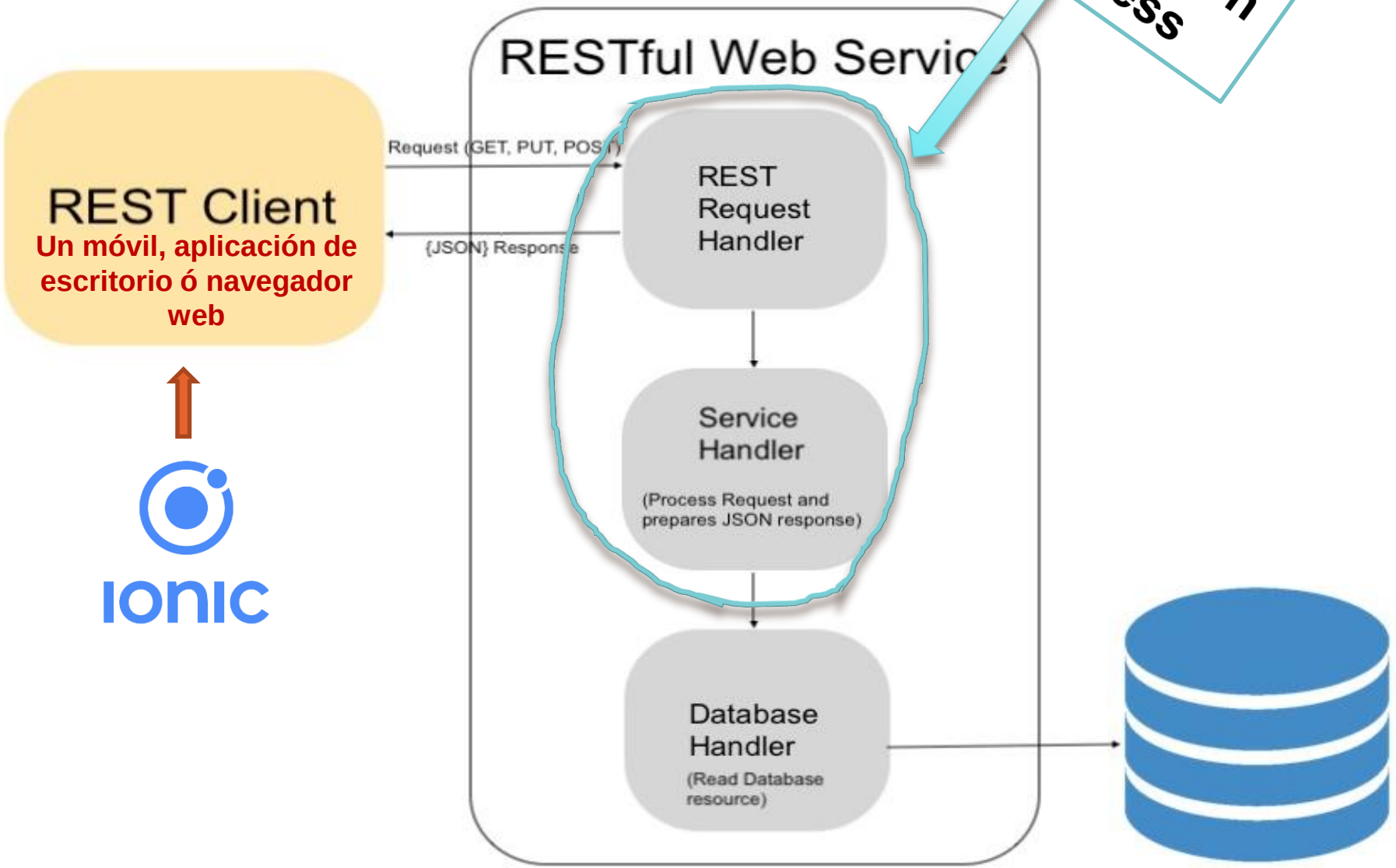
Visión global



No perdamos nunca la visión global que perseguimos…



No perdamos nunca la vis[illegible] global que perseguimos..



Al lío... vamos a hacer nuestra primera API con NodeJS...

¿Qué tenemos ahora?

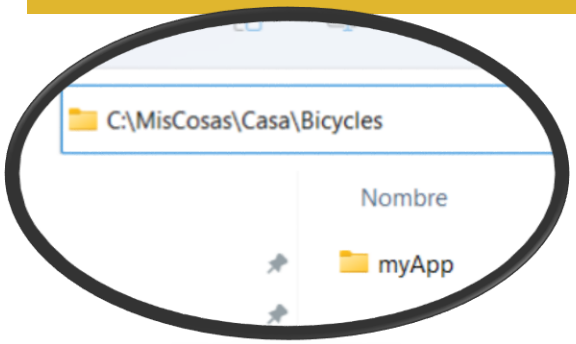
¿Tenemos instalado lo visto anteriormente?

- **NodeJS**, que es lo que ha hecho que JavaScript pueda ejecutarse fuera del navegador web.
- **npm**, que es el gestor de paquetes de node. (Parecido a apt para Linux)
- Los comandos de abajo permiten ver la versión instalada.

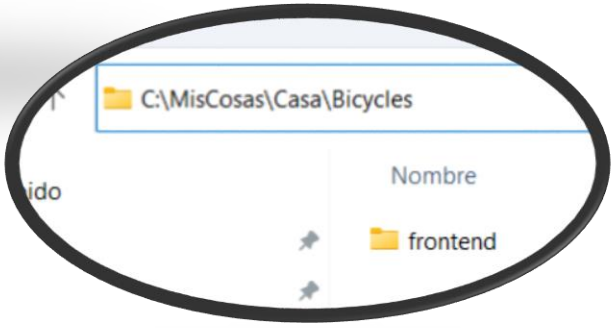
```
$ node --version
$ npm --version
```

Al lío... vamos a hacer nuestra primera API con NodeJS...

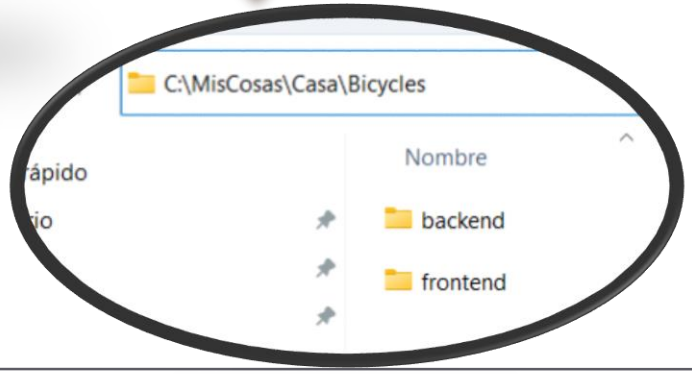
Nuestras carpetas



En el primer tutorial de Ionic la carpeta del proyecto se llamaba myApp. Vamos a llamarla frontend.



Y ahora vamos a crear una nueva carpeta para trabajar con nodeJS + Express + Sequelize. Vamos a llamarla backend.

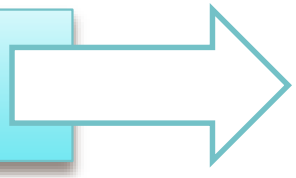


Un simple Get con NodeJS
npm init

Crea un directorio para tu backend e inicia un proyecto con node

```
tibur@DESKTOP-O2362TM MINGW64 /c/MisCosas/Casa/Bicycles/backend
$ npm init
```

Te irá preguntando varias cosas… vamos a verlo…



Un simple Get con NodeJS
node init

```
tibur@DESKTOP-O2362TM MINGW64 /c/MisCosas/Casa/Bicycles/backend
$ npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults.

See `npm help init` for definitive documentation on these fields
and exactly what they do.

Use `npm install <pkg>` afterwards to install a package and
save it as a dependency in the package.json file.

Press ^C at any time to quit.
package name: (backend)
```

Indica el nombre de tu API. Por defecto es el nombre del directorio en el que creas tu proyecto… En este caso yo he pulsado ENTER para la opción por defecto…

Un simple Get con NodeJS
node init

```
$ npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults.

See `npm help init` for definitive documentation on these fields
and exactly what they do.

Use `npm install <pkg>` afterwards to install a package and
save it as a dependency in the package.json file.

Press ^C at any time to quit.
package name: (backend)
version: (1.0.0)
description:
entry point: (index.js)
test command:
git repository:
keywords:
author:
license: (ISC)
About to write to C:\MisCosas\Casa\Bicycles\backend\package.json:
```

Te preguntará una serie de preguntas más que se entienden bien… yo he respondido en este ejemplo con la opción por defecto simplemente pulsando ENTER….

Un simple Get con NodeJS
node init

{} package.json > ...

```
1   {
2     "name": "backend",
3     "version": "1.0.0",
4     "description": "",
5     "main": "index.js",
      Depuración de ▷
6     "scripts": {
7       "test": "echo \"Error: no test specified\" && exit 1"
8     },
9     "author": "",
10    "license": "ISC"
11  }
12
```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL

```
Is this OK? (yes)

tibur@DESKTOP-O2362TM MINGW64 /c/MisCosas/Casa/Bicycles/backend
$
```

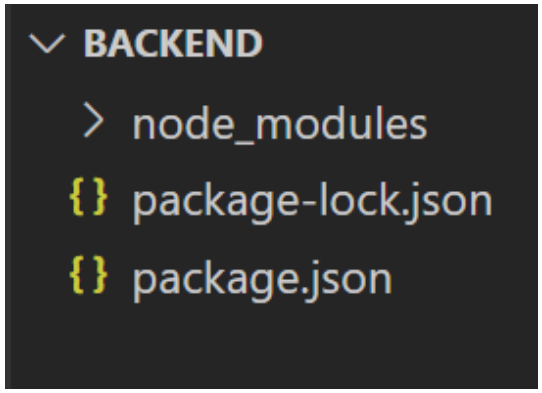
En la última pregunta responde de nuevo ENTER… para responder que está todo OK con lo que se te habrá creado el fichero **package.json** que recoge toda la información introducida en formato JSON.

Un simple Get con NodeJS
Ahora instalamos Express

Instalamos ahora el paquete **Express** que me permitirá crear los **end-points** de mi **API**.

Después de instalar el paquete **Express** te aparecerá la dependencia en **package.json**

Un simple Get con NodeJS
Nuestro directorio ahora mismo



Observa lo que tienes en tu proyecto ahora mismo

package.json

Al inicializar tu proyecto de node con npm init se creó el fichero package.json

node_modules

Al instalar el paquete express se ha creado el directorio node_modules que a partir de ahora albergará todos los paquetes que vayas instalando

Un simple Get con NodeJS
Creemos ahora index.js

```
const express = require("express");

const app = express();

// parse requests of content-type - application/json
app.use(express.json());

// parse requests of content-type - application/x-www-form-urlencoded
app.use(express.urlencoded({ extended: true }));

// simple route
app.get("/", (req, res) => {
  res.json({ message: "Welcome to bicycles application." });
});

// set port, listen for requests
const PORT = process.env.PORT || 8080;
app.listen(PORT, () => {
  console.log(`Server is running on port ${PORT}.`);
});
```

Crea el fichero **index.js** y copia el siguiente código

Un simple Get con NodeJS
Creemos ahora index.js

```
const express = require("express");

const app = express();

// parse requests of content-type - application/json
app.use(express.json());

// parse requests of content-type - application/x-www-form-urlencoded
app.use(express.urlencoded({ extended: true }));

// simple route
app.get("/", (req, res) => {
  res.json({ message: "Welcome to bicycles application." });
});

// set port, listen for requests
const PORT = process.env.PORT || 8080;
app.listen(PORT, () => {
  console.log(`Server is running on port ${PORT}.`);
});
```

Importa la librería express

Un simple Get con NodeJS
Creemos ahora index.js

```
const express = require("express");

const app = express();

// parse requests of content-type - application/json
app.use(express.json());

// parse requests of content-type - application/x-www-form-urlencoded
app.use(express.urlencoded({ extended: true }));

// simple route
app.get("/", (req, res) => {
  res.json({ message: "Welcome to bicycles application." });
});

// set port, listen for requests
const PORT = process.env.PORT || 8080;
app.listen(PORT, () => {
  console.log(`Server is running on port ${PORT}.`);
});
```

Empezamos a usar express usando la constante **app**

Un simple Get con NodeJS
Creemos ahora index.js

```
const express = require("express");

const app = express();

// parse requests of content-type - application/json
app.use(express.json());

// parse requests of content-type - application/x-www-form-u
app.use(express.urlencoded({ extended: true }));

// simple route
app.get("/", (req, res) => {
  res.json({ message: "Welcome to bicycles application." });
});

// set port, listen for requests
const PORT = process.env.PORT || 8080;
app.listen(PORT, () => {
  console.log(`Server is running on port ${PORT}.`);
});
```

Podemos usar la librería express para contenidos application/json y application/x-www-form-urlencoded

Lo veremos un poquito más adelante

Un simple Get con NodeJS
Creemos ahora index.js

```
const express = require("express");

const app = express();

// parse requests of content-type - application/json
app.use(express.json());

// parse requests of content-type - application/x-www-form-urlencoded
app.use(express.urlencoded({ extended: true }));

// simple route
app.get("/", (req, res) => {
  res.json({ message: "Welcome to bicycles application." });
});

// set port, listen for requests
const PORT = process.env.PORT || 8080;
app.listen(PORT, () => {
  console.log(`Server is running on port ${PORT}.`);
});
```

Con estas líneas arrancamos nuestra API que escuchará en el puerto 8080

Un simple Get con NodeJS
Creemos ahora index.js

```
const express = require("express");

const app = express();

// parse requests of content-type - application/json
app.use(express.json());

// parse requests of content-type - application/x-www-form-urle
app.use(express.urlencoded({ extended: true }));

// simple route
app.get("/", (req, res) => {
  res.json({ message: "Welcome to bicycles application." });
});

// set port, listen for requests
const PORT = process.env.PORT || 8080;
app.listen(PORT, () => {
  console.log(`Server is running on port ${PORT}.`);
});
```

Y lo más importante:

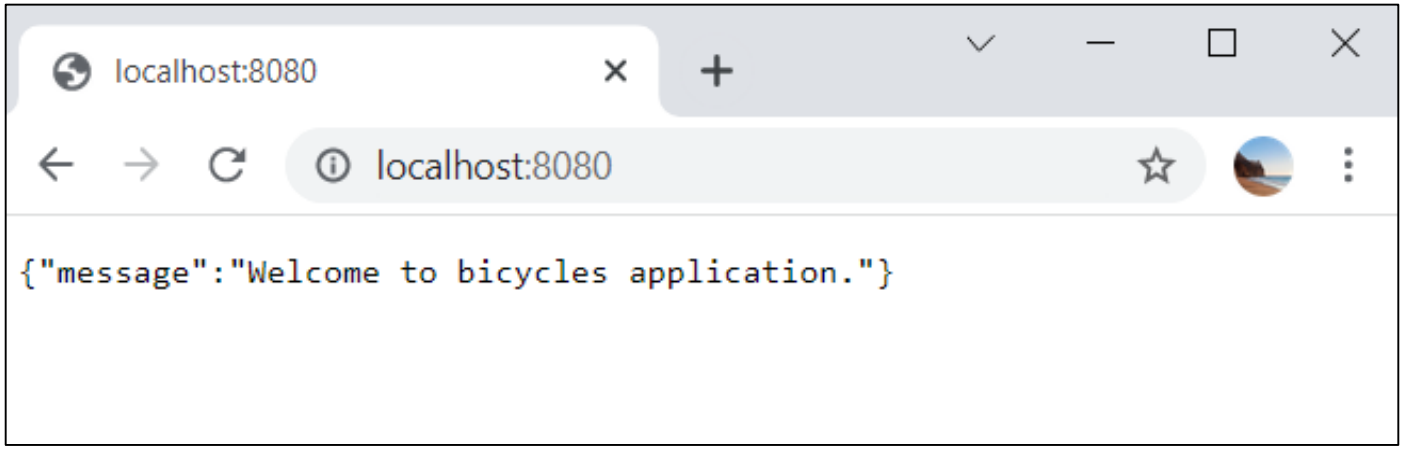
Tenemos un end-point que escucha en:
http://localhost:8080/

Y devolverá un mensajito en formato JSON…

Un simple Get con NodeJS
Arranquemos nuestra API

```
tibur@DESKTOP-O2362TM MINGW64 /c/MisCosas/Casa/Bicycles/backend
$ node index.js
Server is running on port 8080.
```

Ahora nuestra API está escuchando en http://localhost:8080

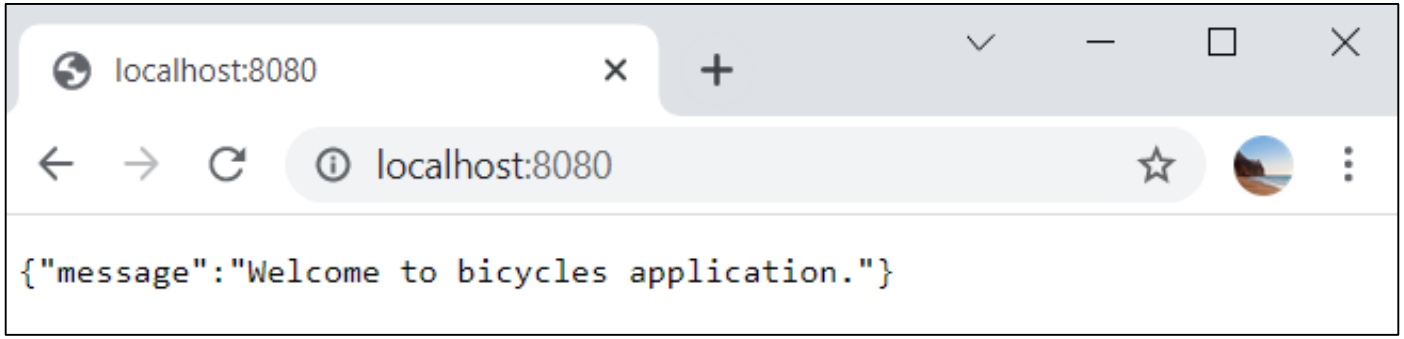


Un simple Get con NodeJS
Nuestra API tiene un end-point

```
// simple route
app.get("/", (req, res) => {
  res.json({ message: "Welcome to bicycles application." });
});
```

Hay que tener en cuenta que realmente estamos accediendo a un end-point que en este caso es:

GET http://localhost:8080/



Un simple Get con NodeJS

Prueba tus end-points con POSTMAN

Instala **POSTMAN** desde:

https://www.postman.com/downloads/

Con **POSTMAN** podemos probar nuestra API accediendo a los end-points. En el pantallazo se muestra que accedemos a:

GET http://localhost:8080/

Un simple Get con NodeJS

Ahora vamos a por el ORM...

ORM (Object Relationship Mapping) permite en la práctica crear una base de datos orientada a objetos.

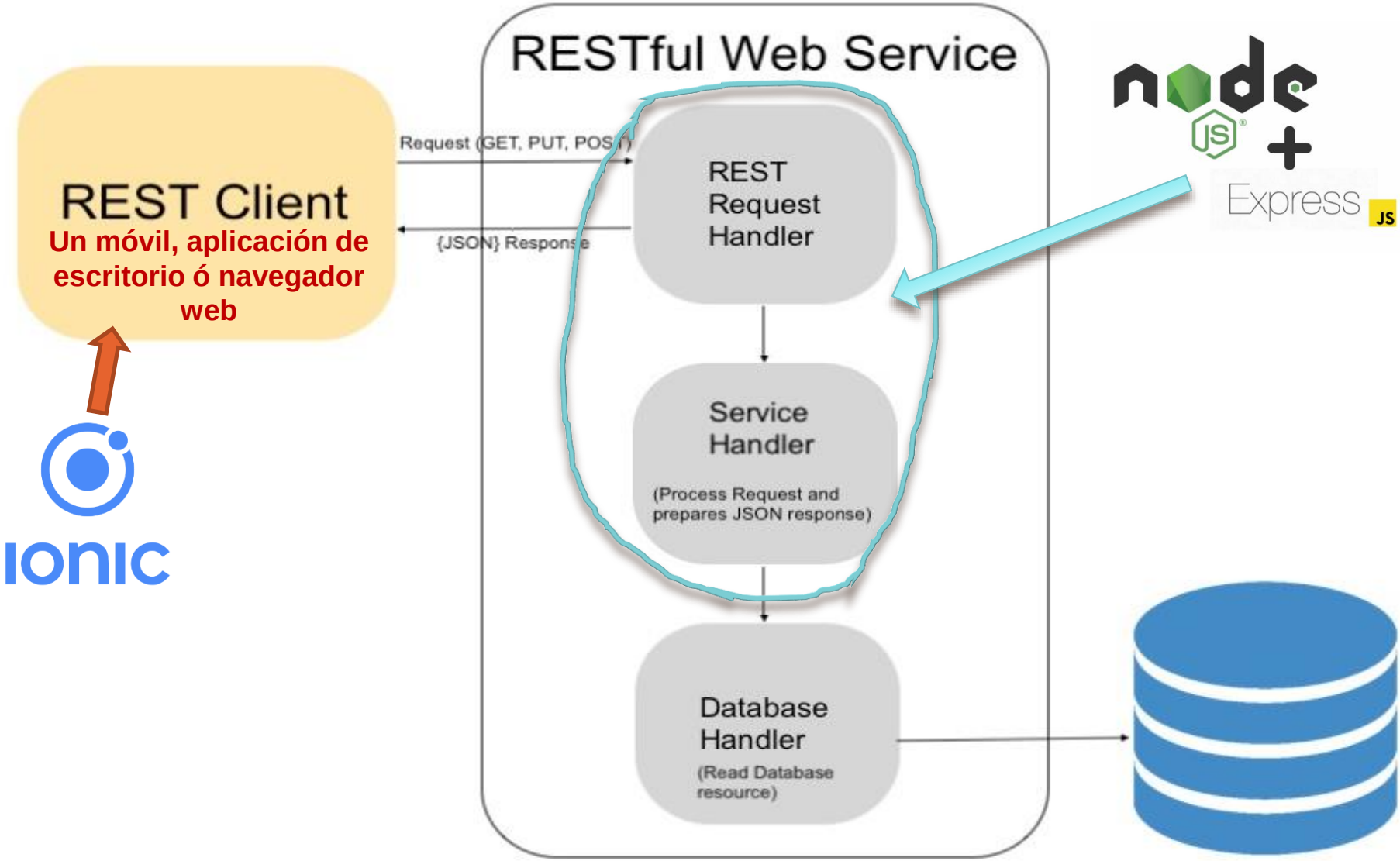
Para programar podrás usar objetos que el ORM guardará automáticamente en registros.

Una clase se corresponde con una tabla
Un objeto se corresponde con un registro

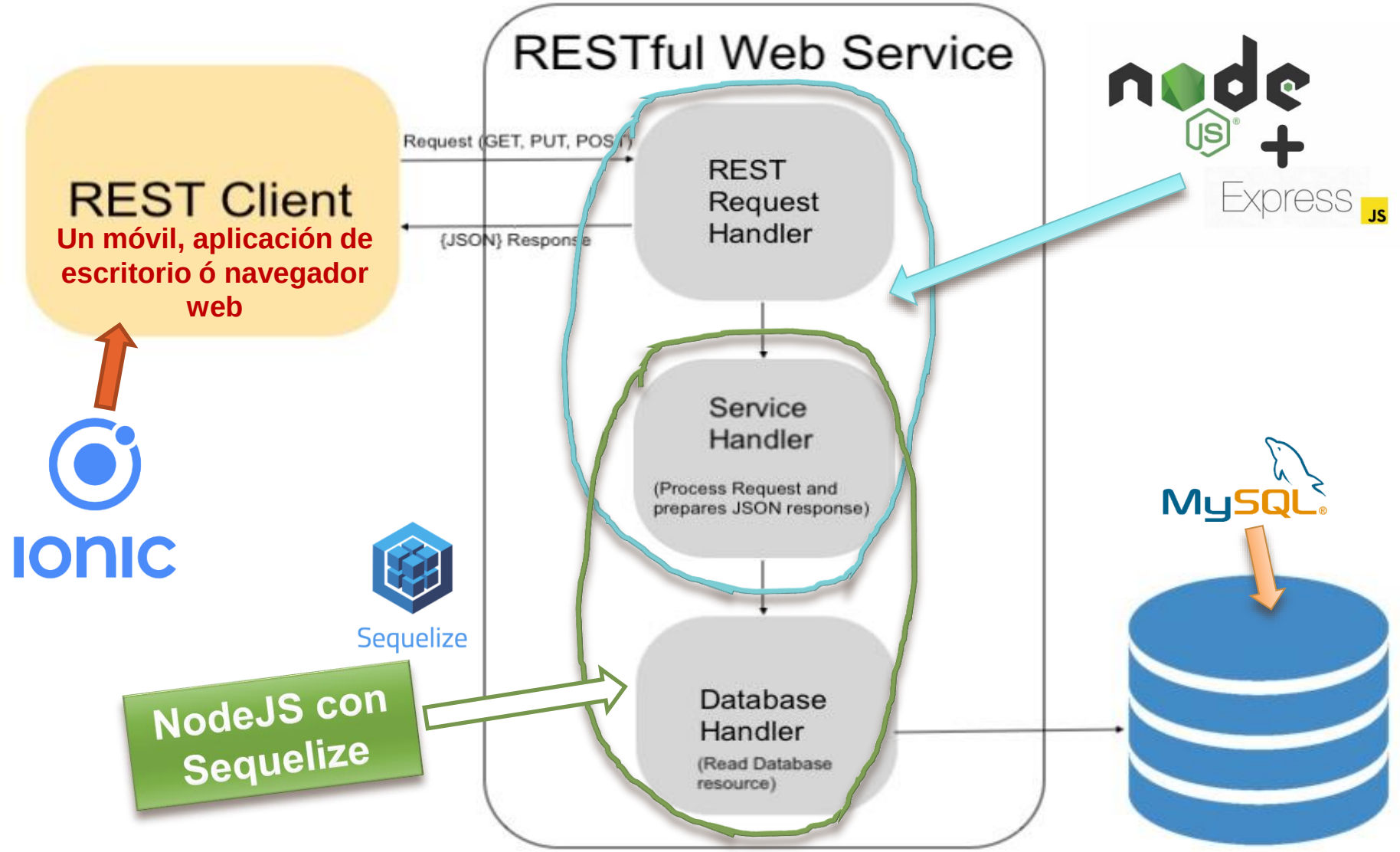
En nuestro caso usaremos:

Sequelize

No perdamos nunca la visión global que perseguimos...



No perdamos nunca la visión global que perseguimos...



Un simple Get con NodeJS
Instalar sequelize y mysql

Instala el paquete de sequelize y de mysql2

```
tibur@DESKTOP-O2362TM MINGW64 /c/MisCosas/Casa/Bicycles/backend
$ npm install sequelize mysql2
```

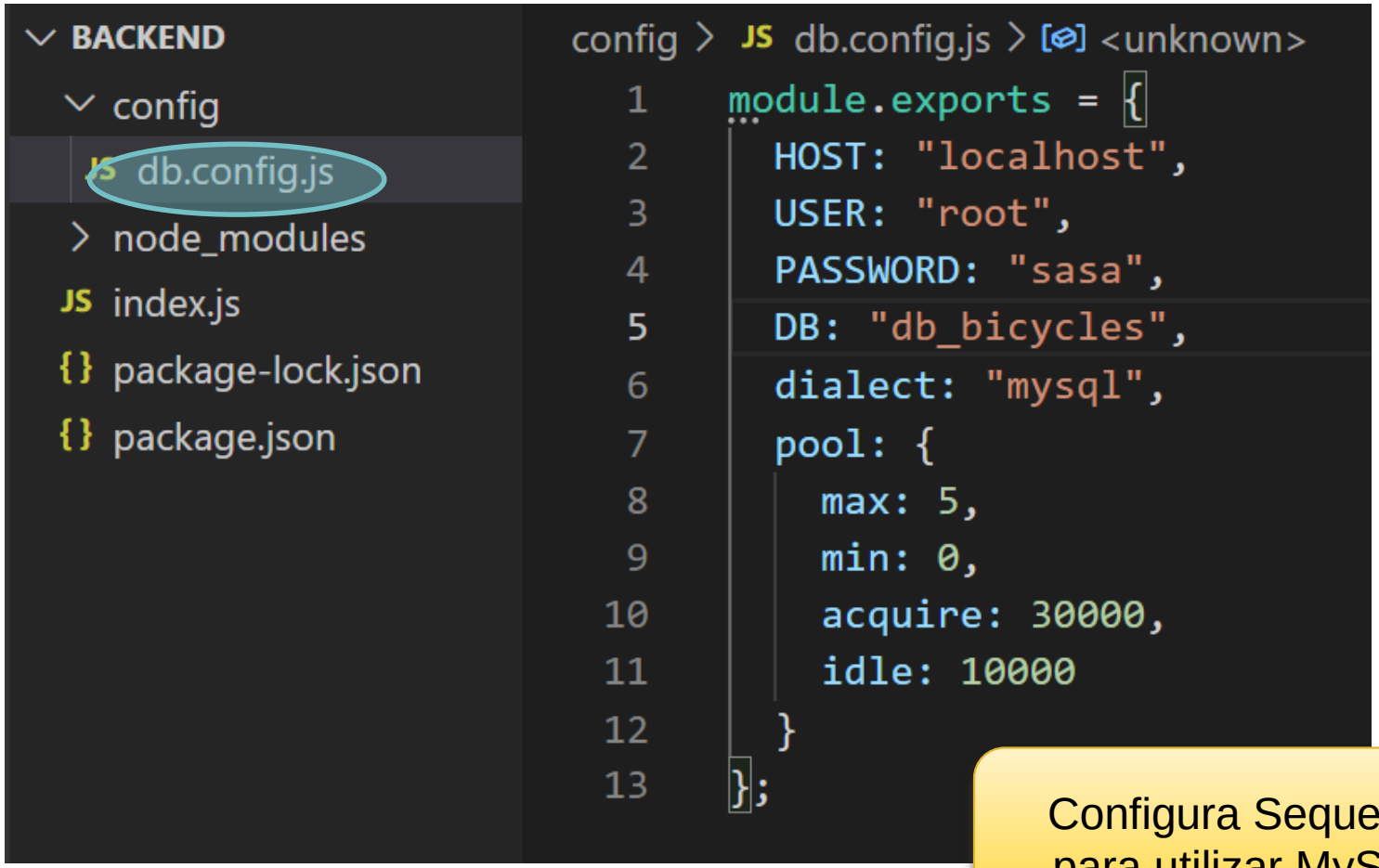
Puedes instalar varios paquetes a la vez.

Aquí hemos instalado sequelize y mysql2 a la vez.

Sequelize es el ORM.

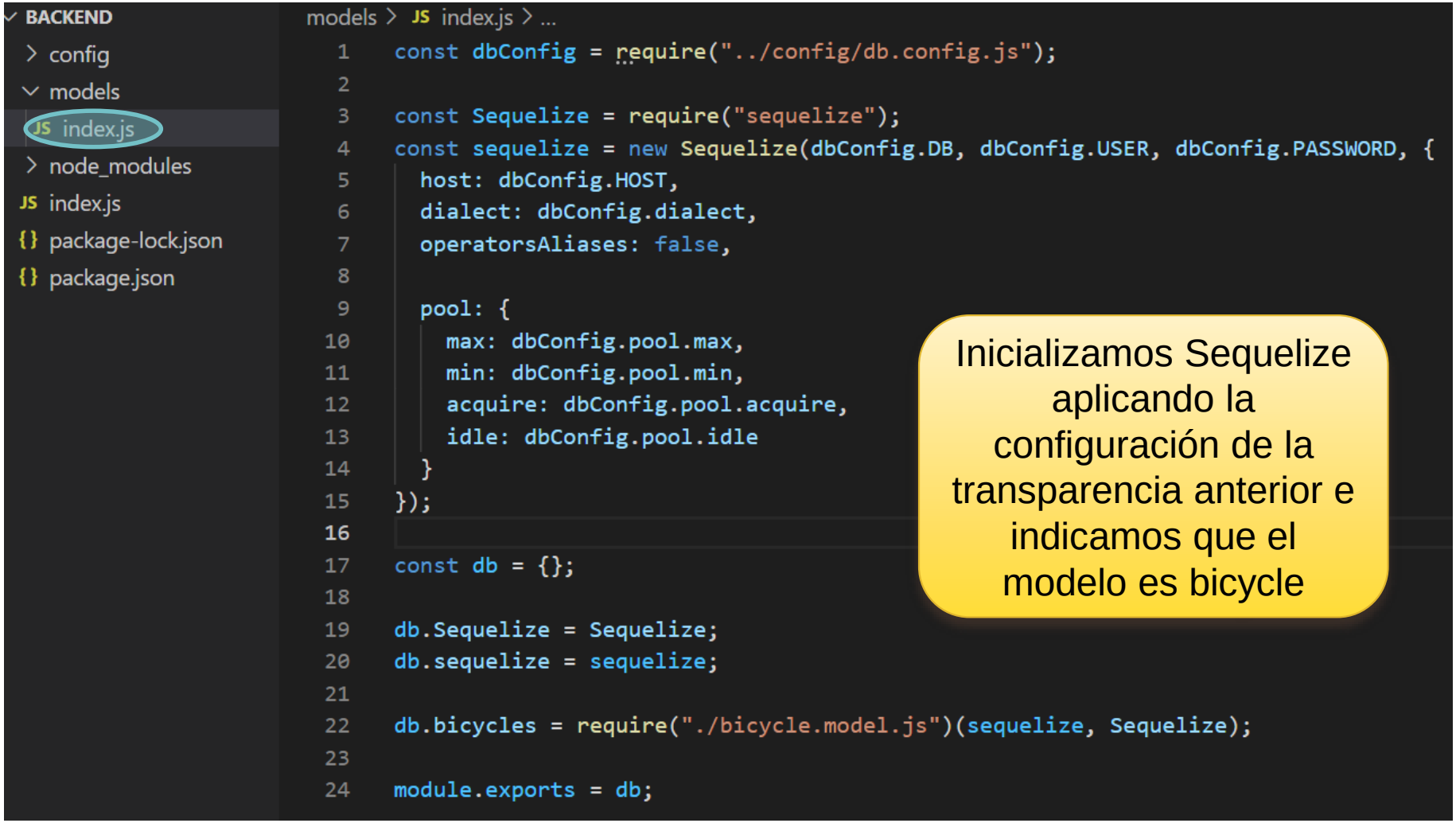
Mysql2 es para utilizar mysql.

Un simple Get con NodeJS Usando el ORM Sequelize

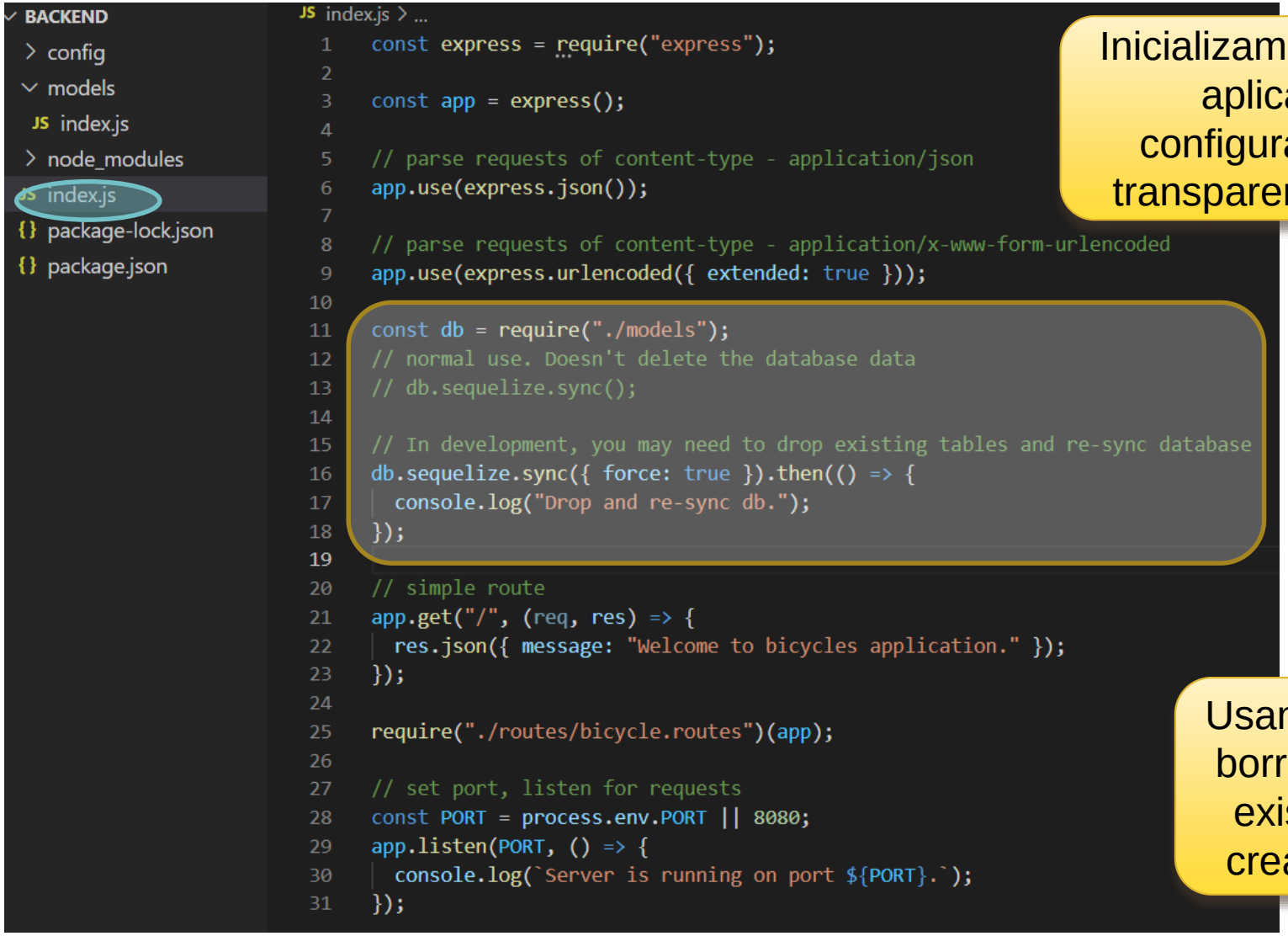


Configura Sequelize para utilizar MySQL

Un simple Get con NodeJS
Inicializar Sequelize



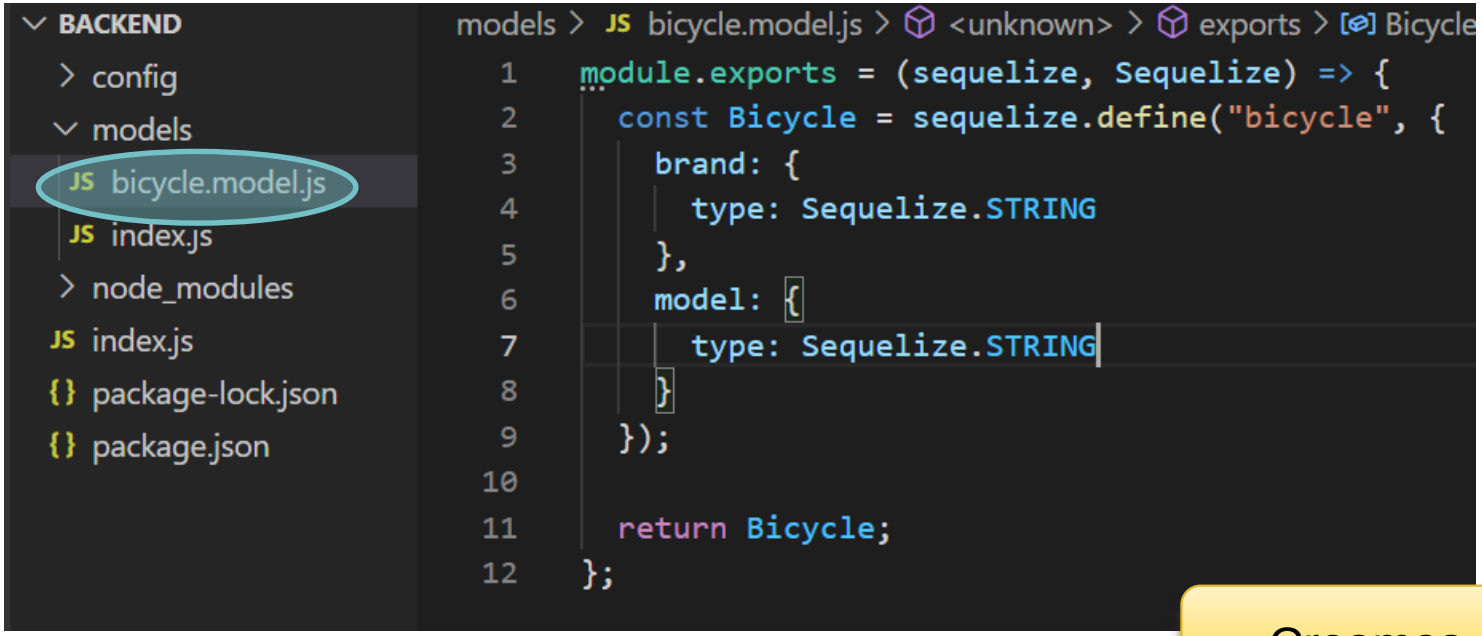
Un simple Get con NodeJS Sync (force or not force)



Inicializamos Sequelize aplicando la configuración de la transparencia anterior

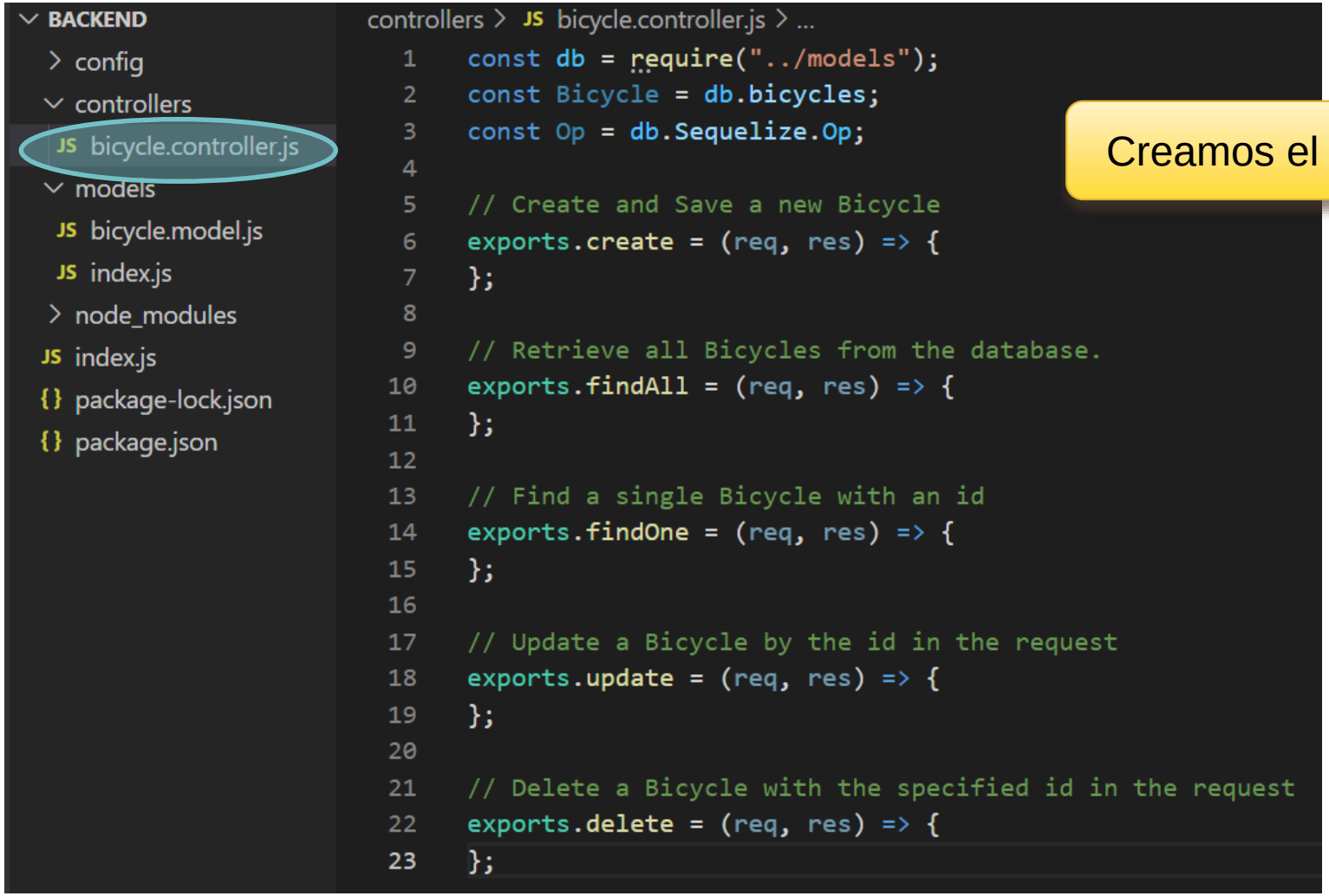
Usando force: true borrará las tablas existentes y las creará de nuevo

Un simple Get con NodeJS
Creemos el modelo



Creamos el modelo

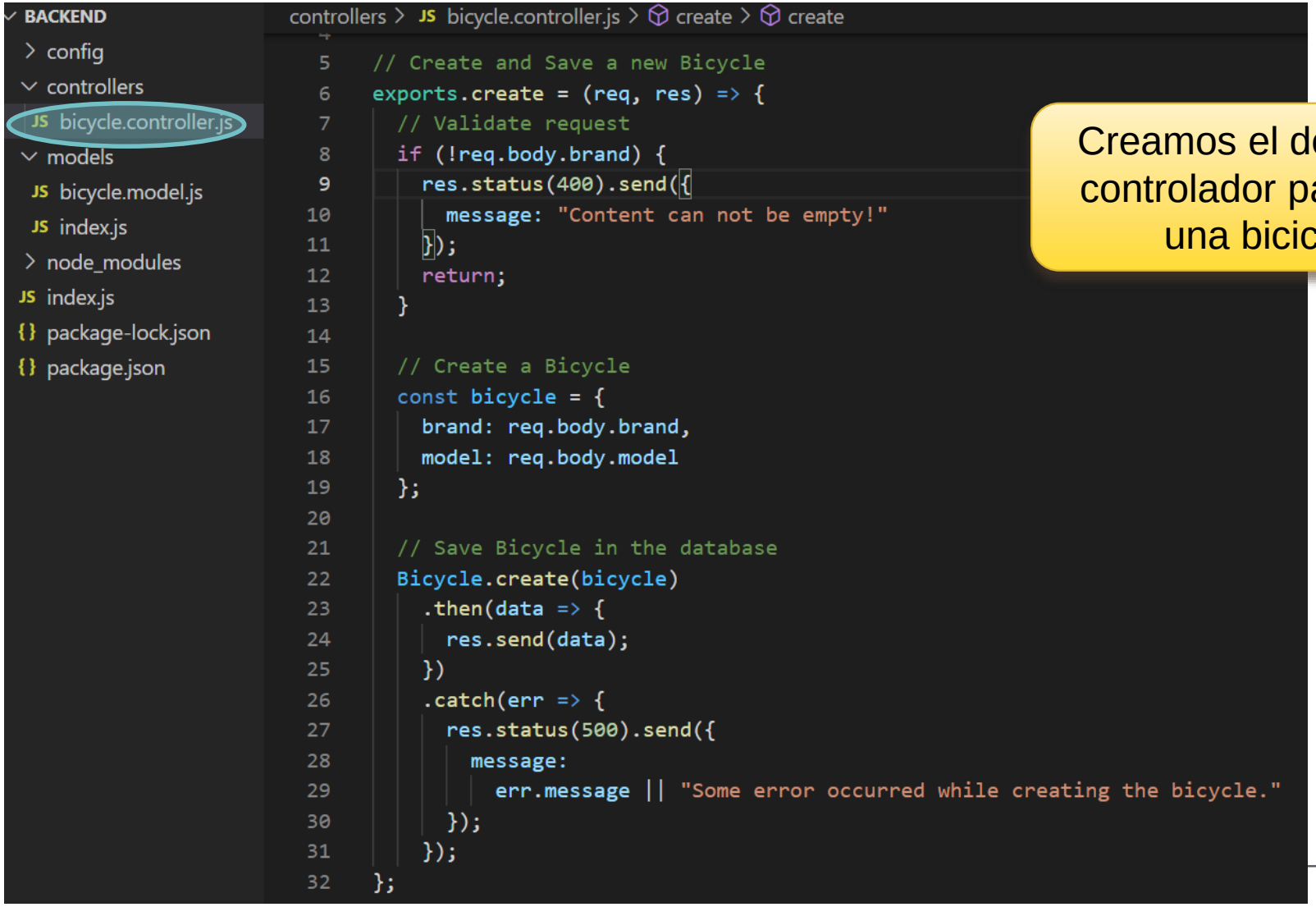
Un simple Get con NodeJS
Creemos el controlador



Creamos el controlador

Un simple Get con NodeJS

Creemos el método create en el controlador



Creamos el detalle del controlador para crear una bicicleta

Un simple Get con NodeJS

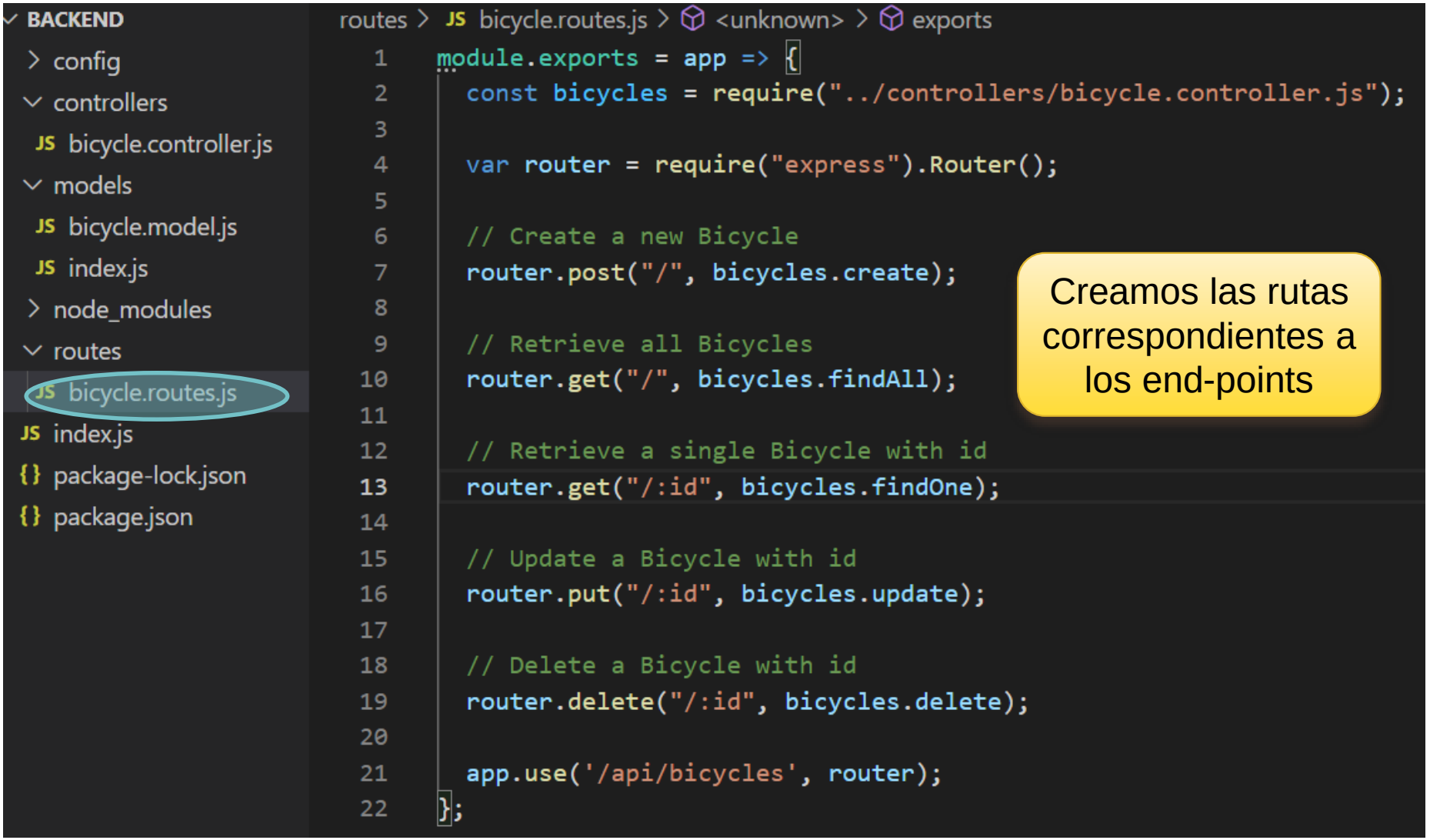
Creemos el método findAll en el controlador

Creamos el detalle del controlador para mostrar todas las bicicletas

De esta manera podrías añadir el resto de métodos del controlador:
https://www.bezkoder.com/node-js-express-sequelize-mysql/

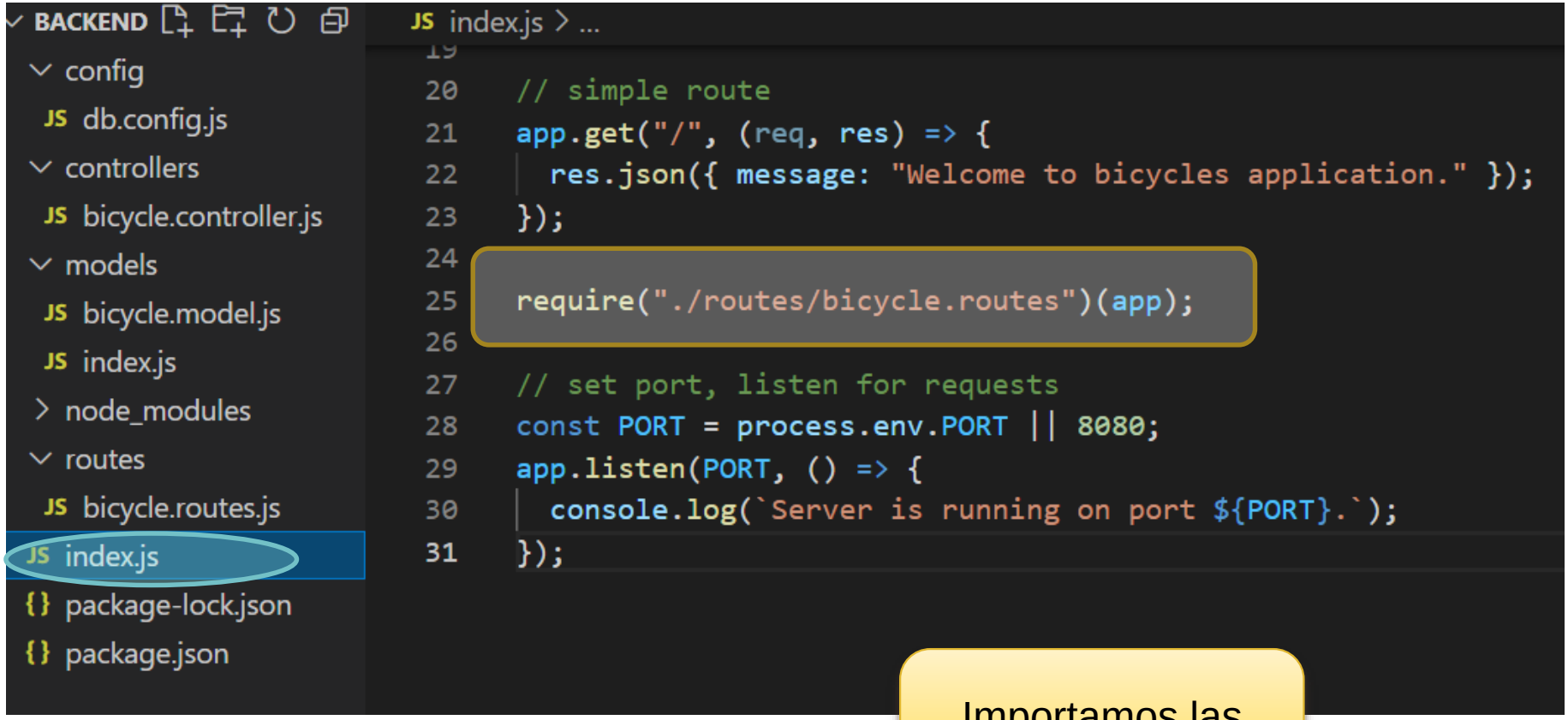
Un simple Get con NodeJS

Creemos las rutas (los end-point)

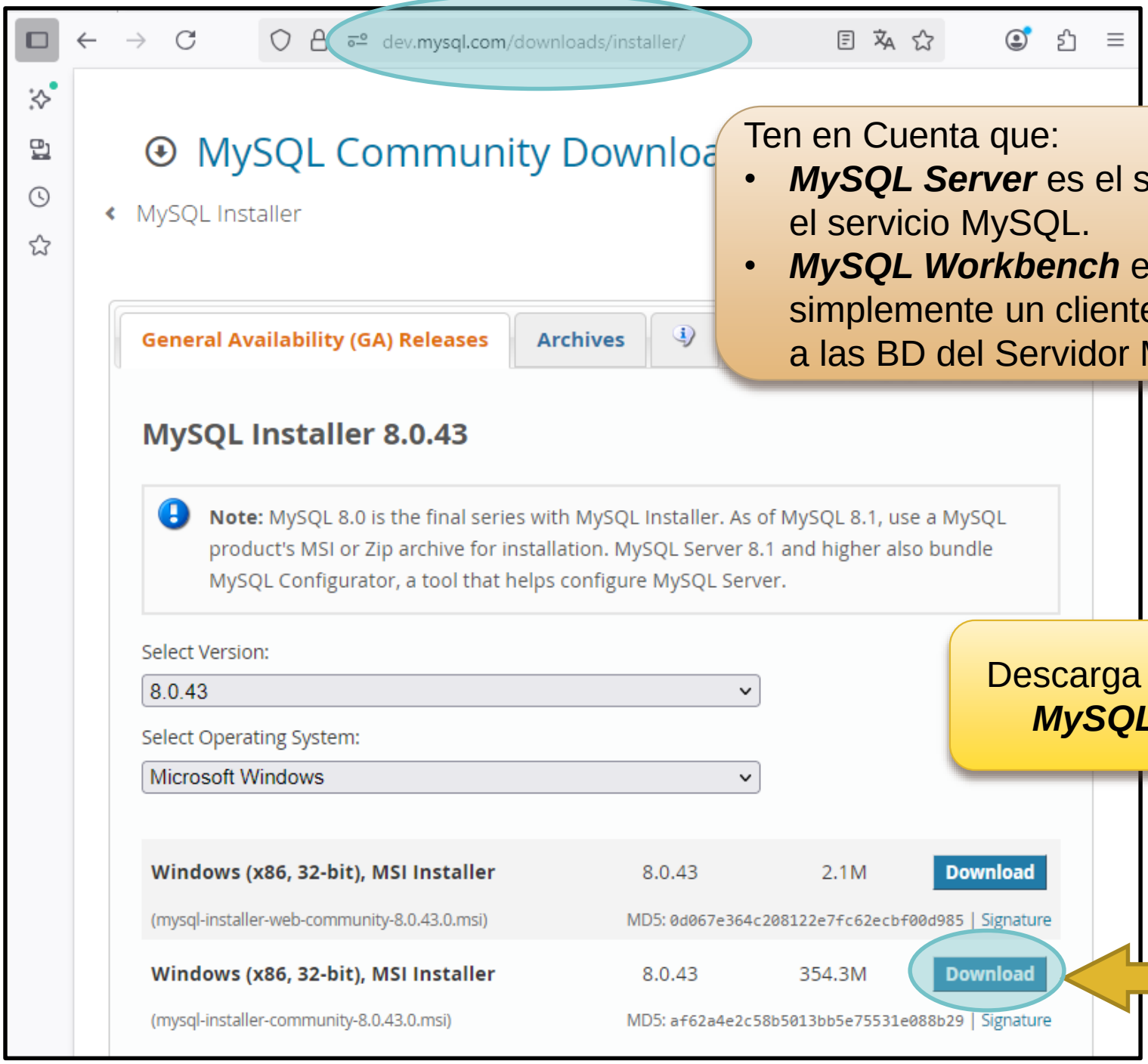


Un simple Get con NodeJS

Importamos las rutas en index.js



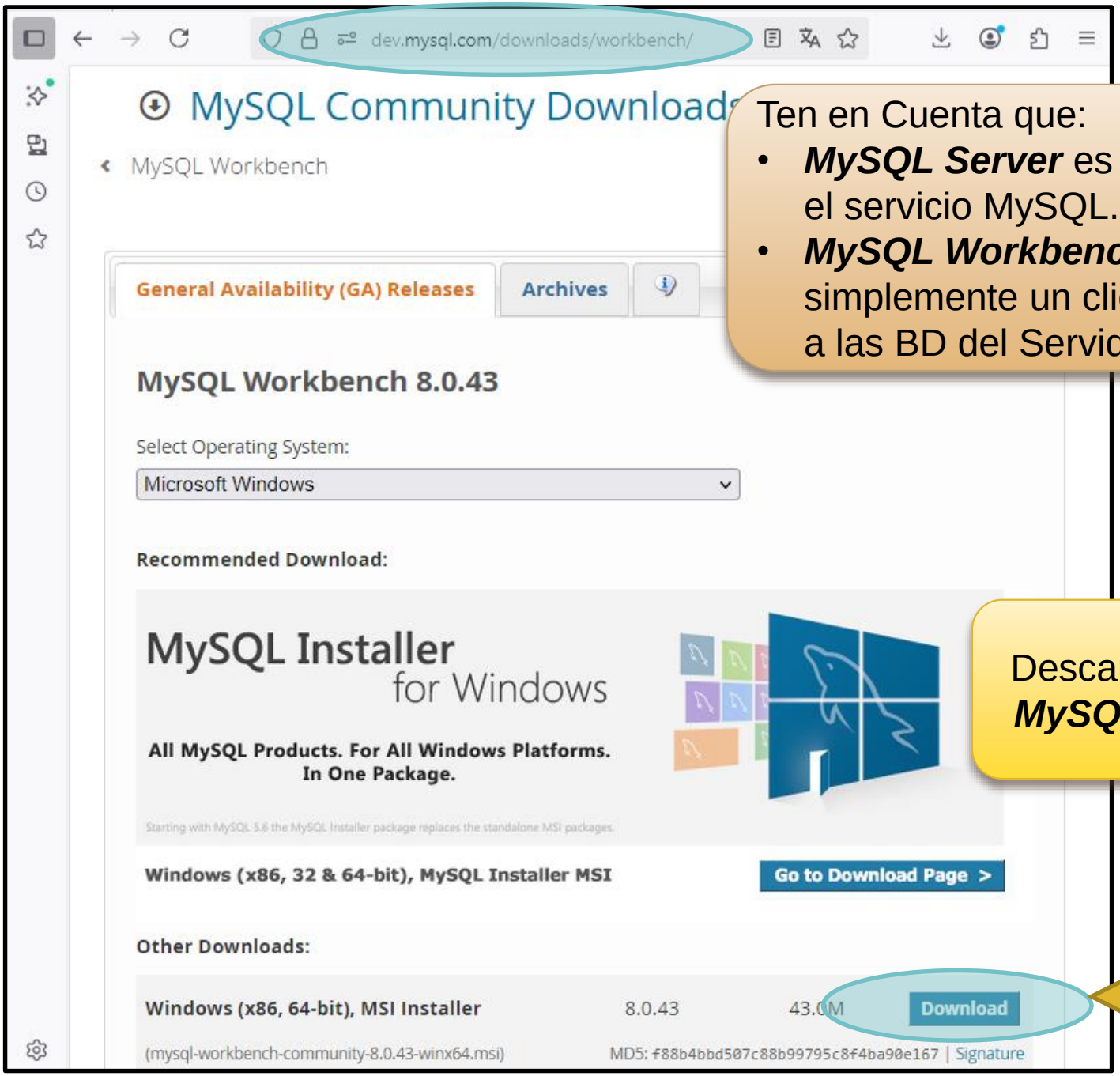
Importamos las rutas en index.js



Ten en Cuenta que:

- ***MySQL Server*** es el software para el servicio MySQL.
- ***MySQL Workbench*** es simplemente un cliente que accede a las BD del Servidor MySQL.

Descarga e instala el ***MySQL Server***

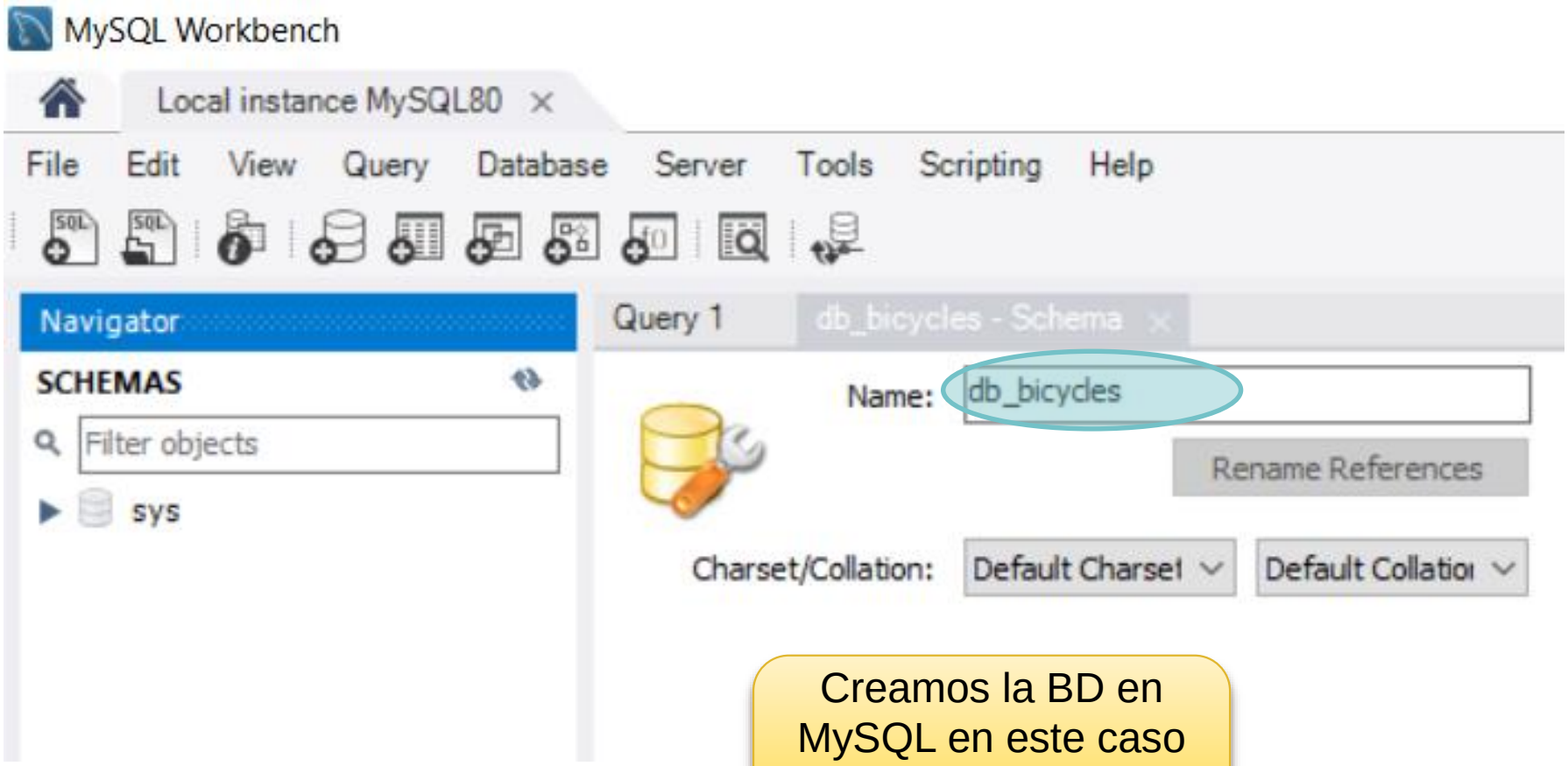


Ten en Cuenta que:

- ***MySQL Server*** es el software para el servicio MySQL.
- ***MySQL Workbench*** es simplemente un cliente que accede a las BD del Servidor MySQL.

Descarga e instala el ***MySQL Workbench***

Un simple Get con NodeJS
Creamos la BD



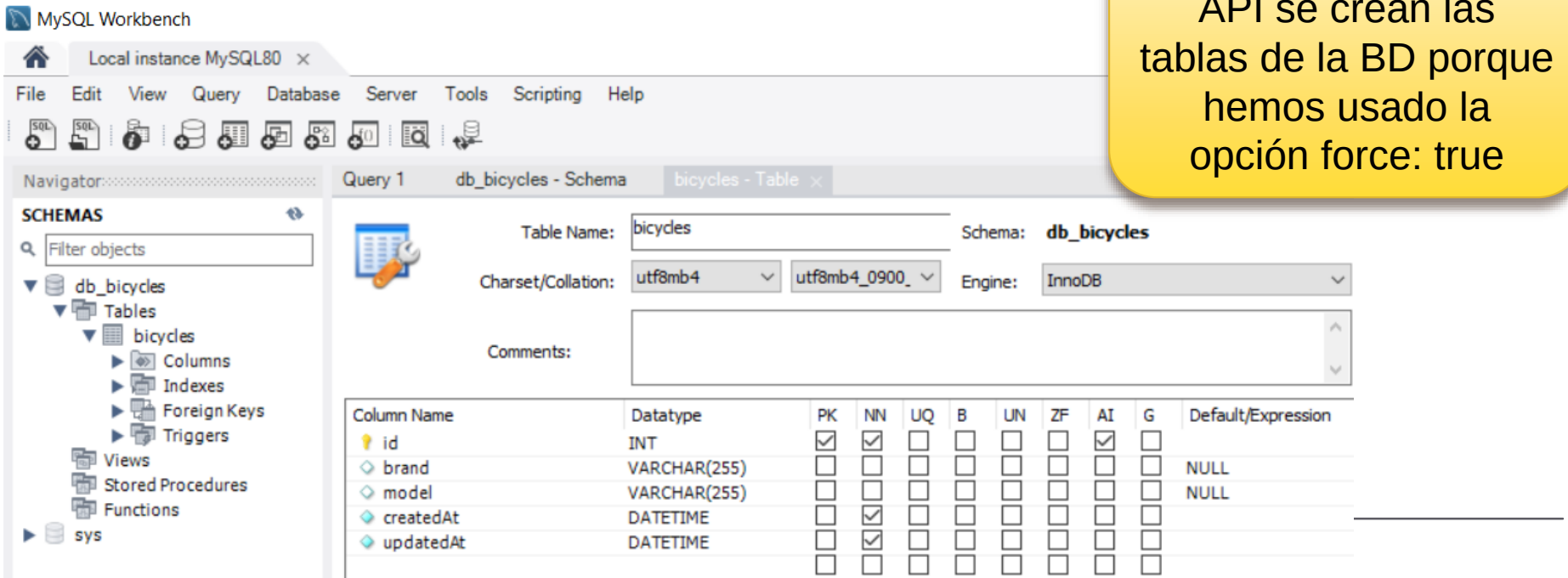
Creamos la BD en MySQL en este caso con el MySQL Workbench

Un simple Get con NodeJS

Arrancamos nuestra API

```
tibur@DESKTOP-O2362TM MINGW64 /c/MisCosas/Casa/Bicycles/backend
$ node index.js
(node:12344) [SEQUELIZE0004] DeprecationWarning: A boolean value was passed to options.operatorsAliases. This is a no-op with v5 and should
 be removed.
(Use `node --trace-deprecation ...` to show where the warning was created)
Server is running on port 8080.
Executing (default): DROP TABLE IF EXISTS `bicycles`;
Executing (default): DROP TABLE IF EXISTS `bicycles`;
Executing (default): CREATE TABLE IF NOT EXISTS `bicycles` (`id` INTEGER NOT NULL auto_increment , `brand` VARCHAR(255), `model` VARCHAR(25
5), `createdAt` DATETIME NOT NULL, `updatedAt` DATETIME NOT NULL, PRIMARY KEY (`id`)) ENGINE=InnoDB;
Executing (default): SHOW INDEX FROM `bicycles`
Drop and re-sync db.
```

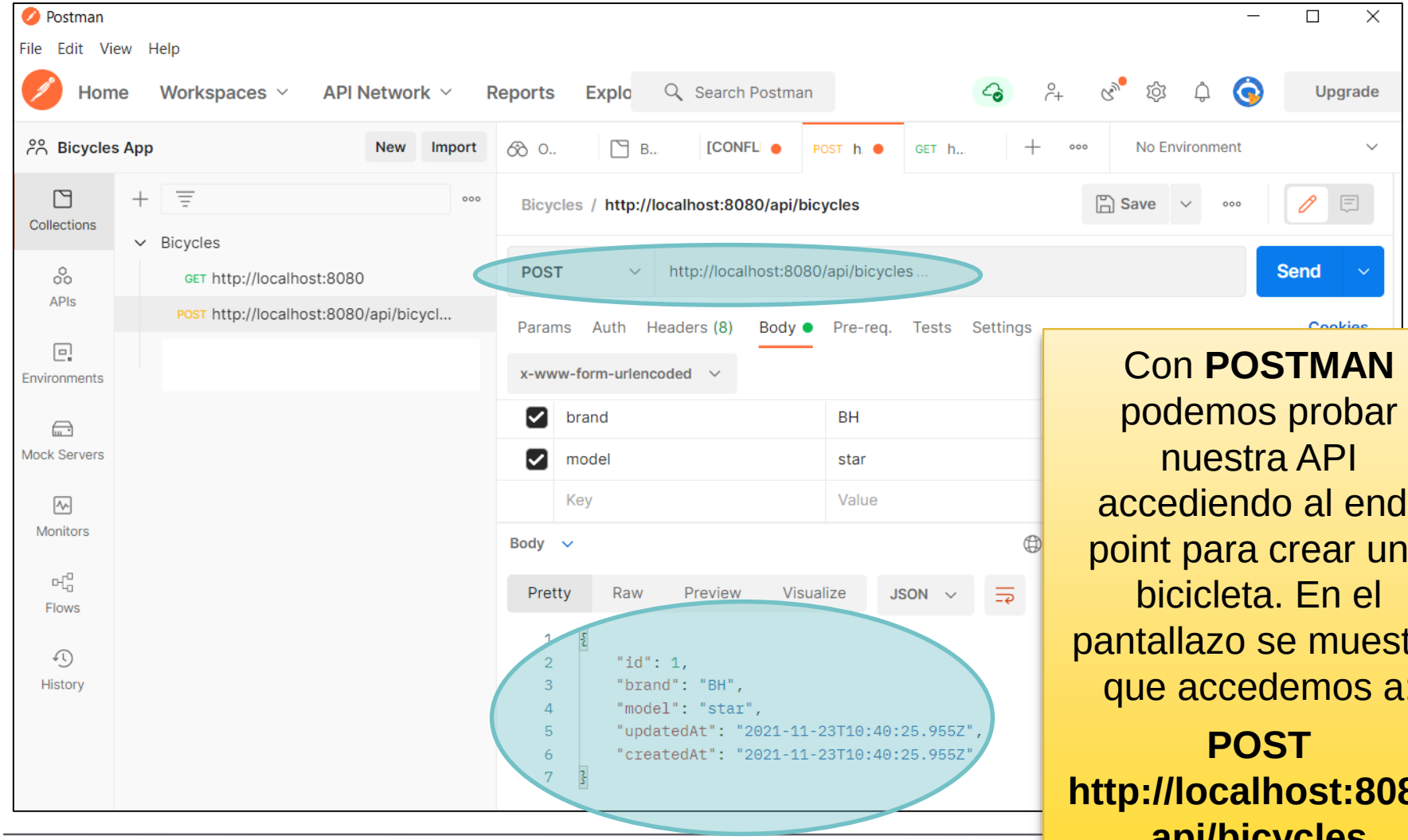
Al arrancar nuestra API se crean las tablas de la BD porque hemos usado la opción force: true



Probemos ahora nuestra API usando **POSTMAN**

Un simple Get con NodeJS

Prueba tu end-point para crear una bicicleta con POSTMAN



Con **POSTMAN** podemos probar nuestra API accediendo al end-point para crear una bicicleta. En el pantallazo se muestra que accedemos a:

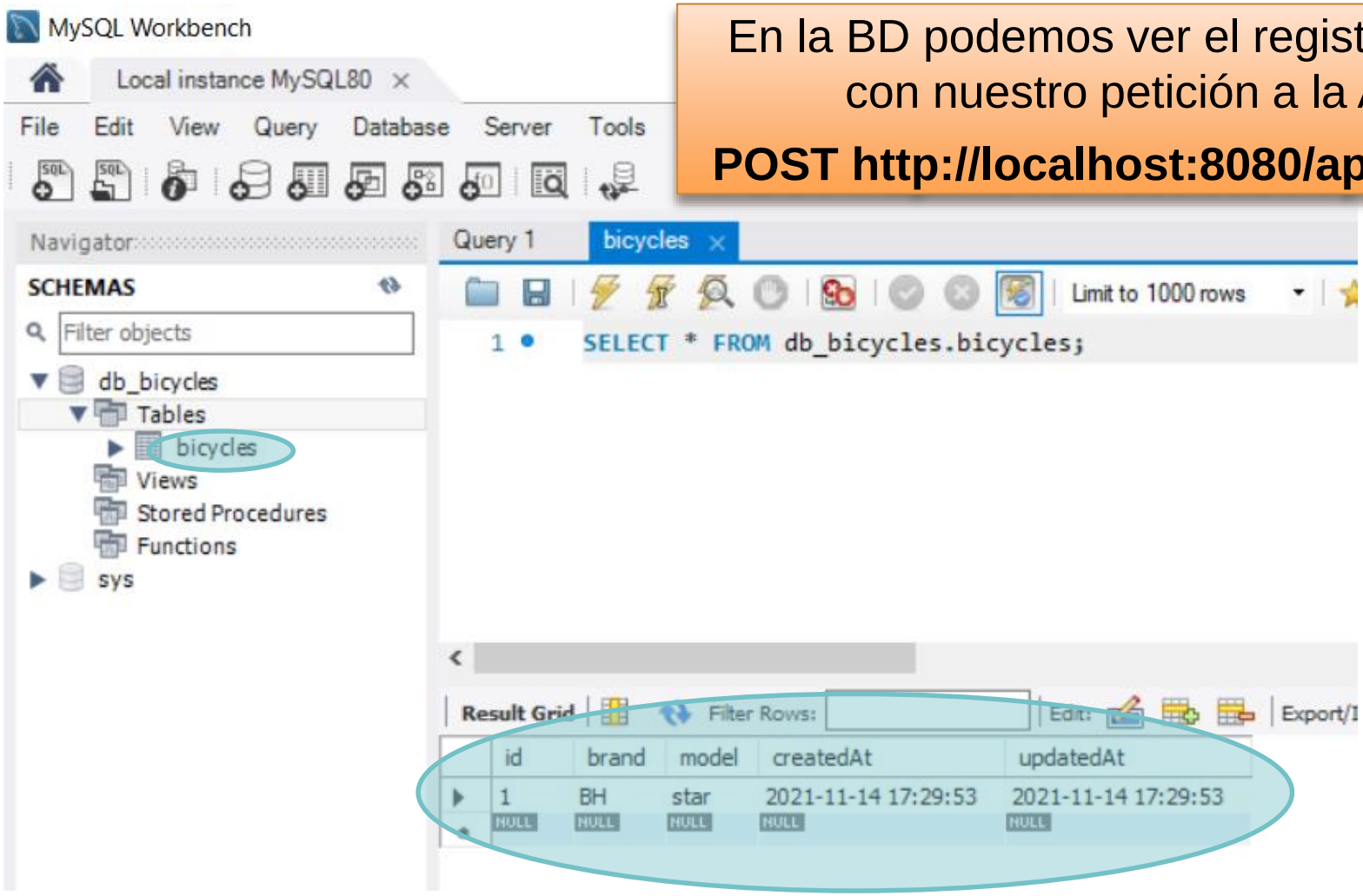
POST http://localhost:8080/api/bicycles

Un simple Get con NodeJS

Prueba tu end-point para crear una bicicleta con POSTMAN

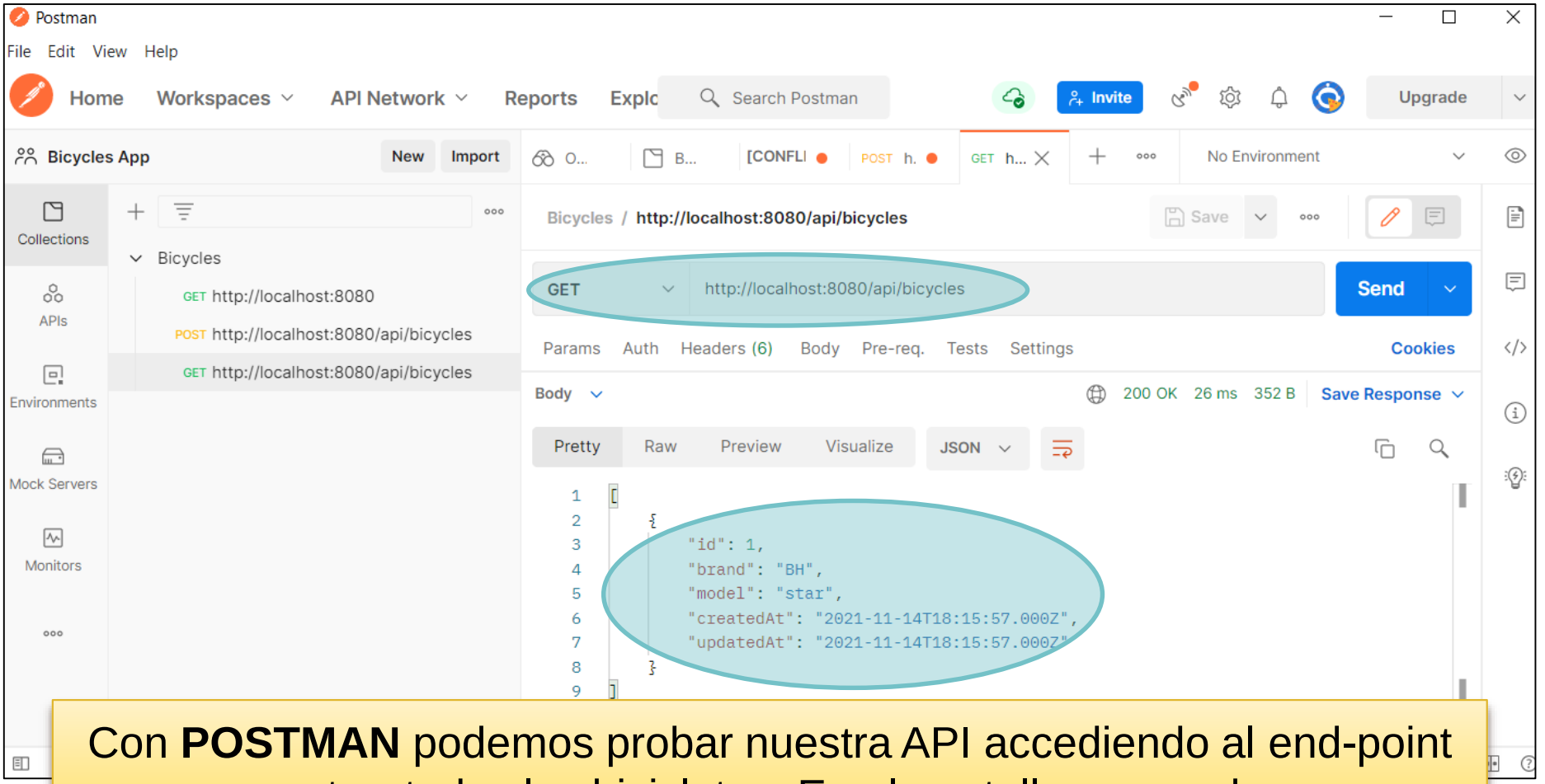
En la BD podemos ver el registro creado con nuestro petición a la API:

POST http://localhost:8080/api/bicycles



Un simple Get con NodeJS

Ahora prueba tu end-point para mostrar las bicicletas con POSTMAN



Con **POSTMAN** podemos probar nuestra API accediendo al end-point para mostrar todas las bicicletas. En el pantallazo accedemos a:

GET http://localhost:8080/api/bicycles

Sigue aprendiendo...

Sigue el siguiente ejemplo paso a paso que es realmente el que yo he seguido para añadir todo el código del controlador que falta y hacer todas las pruebas:

- https://www.bezkoder.com/node-js-express-sequelize-mysql/

Si quieres un ejemplo más sencillo que el que hemos hecho puedes ver el siguiente vídeo: (Las versiones son viejas pero los pasos siguen siendo válidos)

- https://www.youtube.com/watch?v=43D2POUWq0Y

Conclusiones

¿Qué hemos aprendido?

- Hemos instalado NodeJS.
- Hemos creado una API para hacer 3 end-points: un POST y 2 GETs.
- Hemos probado nuestra API usando POSTMAN.

Próximos pasos…

- Terminar el CRUD y probar con POSTMAN todos los end-points.
- Añadir relaciones one-to-many, many-to-many y one-to-one.
- Añadir Autenticación a nuestra API.