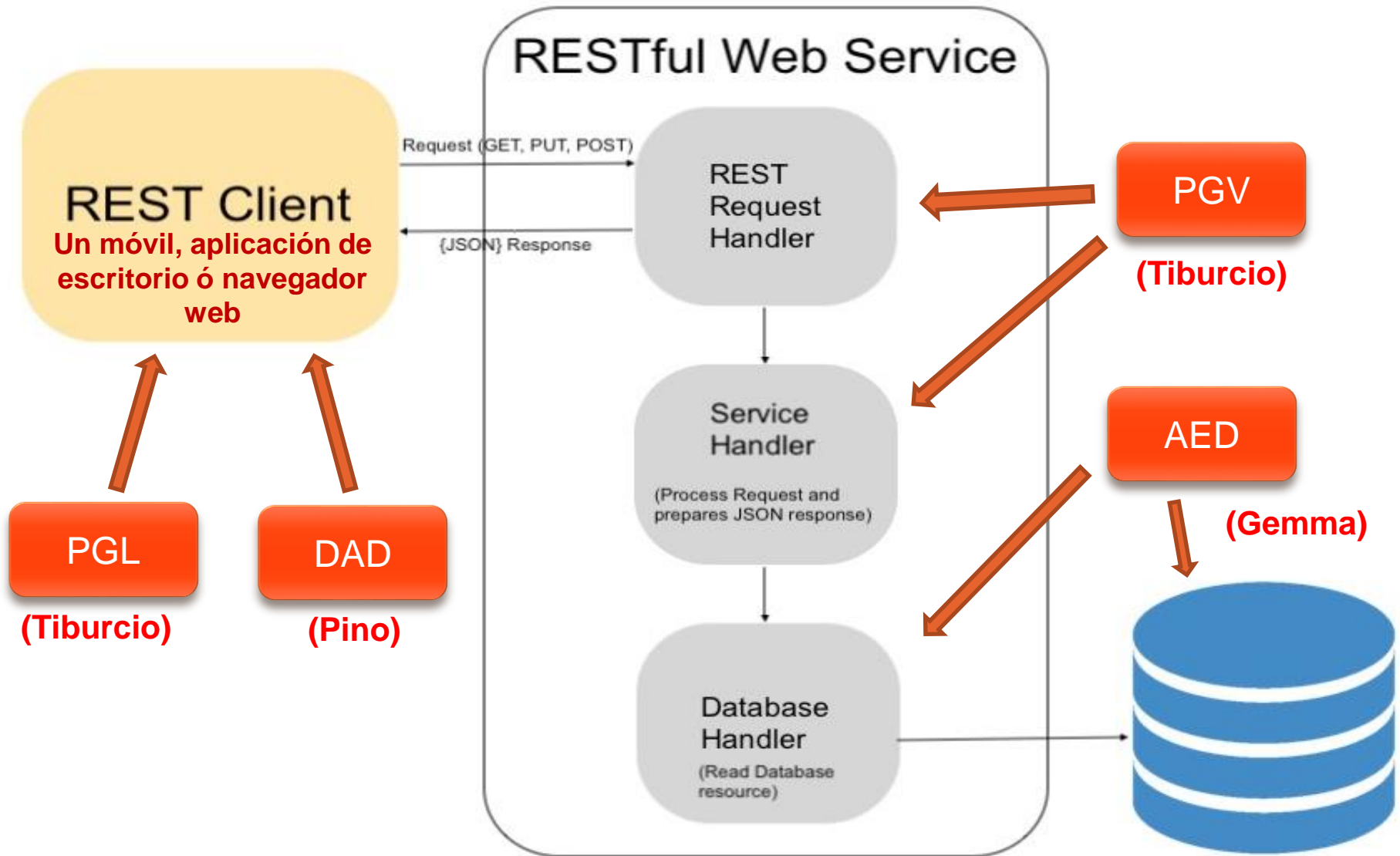


# Introducción a NodeJS

Resumen de pasos basado en la web:

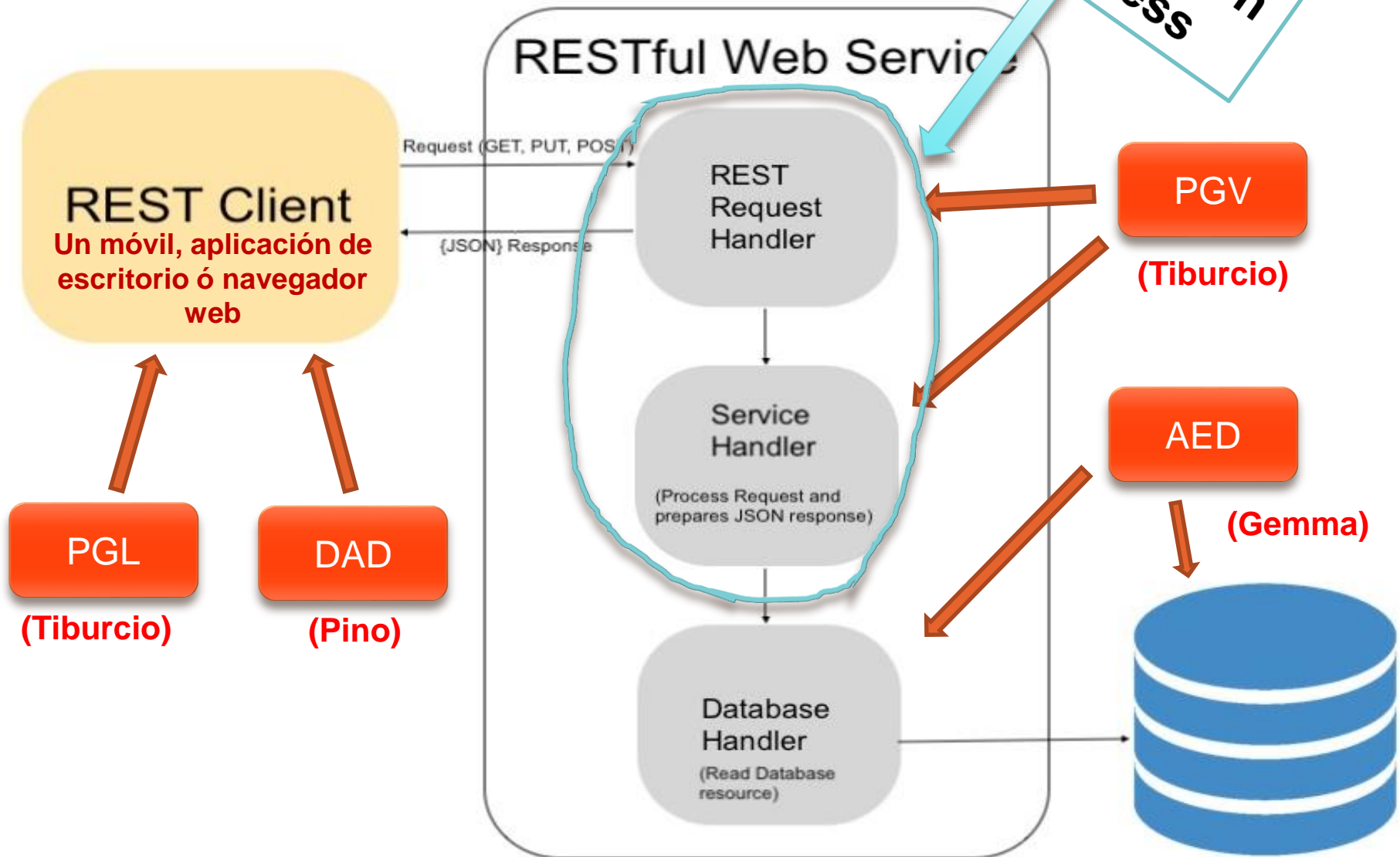
<https://www.bezkoder.com/node-js-express-sequelize-mysql/>

# No perdamos nunca la visión global que perseguimos...



# No perdamos nunca la vista global que perseguimos...

NodeJS con Express



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

Pasos previos

En la página oficial de Node:

<https://nodejs.org/es/>

Descarga e instala la versión LTS de NodeJS



**16.13.0 LTS**

Recomendado para la mayoría

**17.1.0 Actual**

Últimas características

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

¿Qué tenemos ahora?

## Tenemos instalado:

- **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
```

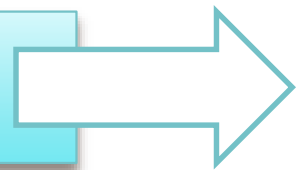
# Un simple Get con NodeJS

## npm init

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

```
tibur@DESKTOP-02362TM 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-02362TM 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-02362TM 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

```
{ } package.json > ...
4   "description": "",
5   "main": "index.js",
  Depuración de ▶
6   "scripts": {
7     "test": "echo \"Error: no test specified\" && exit
8   },
9   "author": "",
10  "license": "ISC",
11  "dependencies": {
12    "express": "^4.17.1"
13  }
14 }
15
```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL

```
tibur@DESKTOP-02362TM MINGW64 /c/MisCosas/Casa/Bicycles/backend
$ npm install express
```

```
added 50 packages, and audited 51 packages in 4s
```

```
found 0 vulnerabilities
```

```
tibur@DESKTOP-02362TM MINGW64 /c/MisCosas/Casa/Bicycles/backend
$
```

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

### ✓ BACKEND

> node\_modules

{ } package-lock.json

{ } package.json

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");
```

Importa la librería 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}.`);
```

```
});
```

# 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-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}.`);
});
```

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-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}.`);
});
```

Y lo más importante:

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

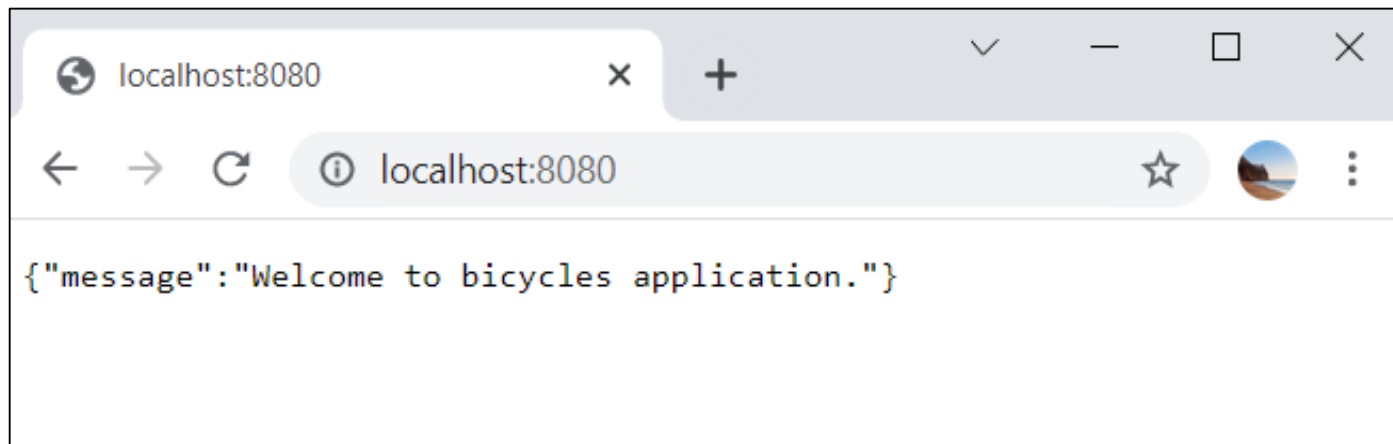
**Y devolverá un mensajito en formato JSON...**

# Un simple Get con NodeJS

## Arranquemos nuestra API

```
tibur@DESKTOP-02362TM MINGW64 /c/MisCosas/Casa/Bicycles/backend  
$ node index.js  
Server is running on port 8080.
```

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



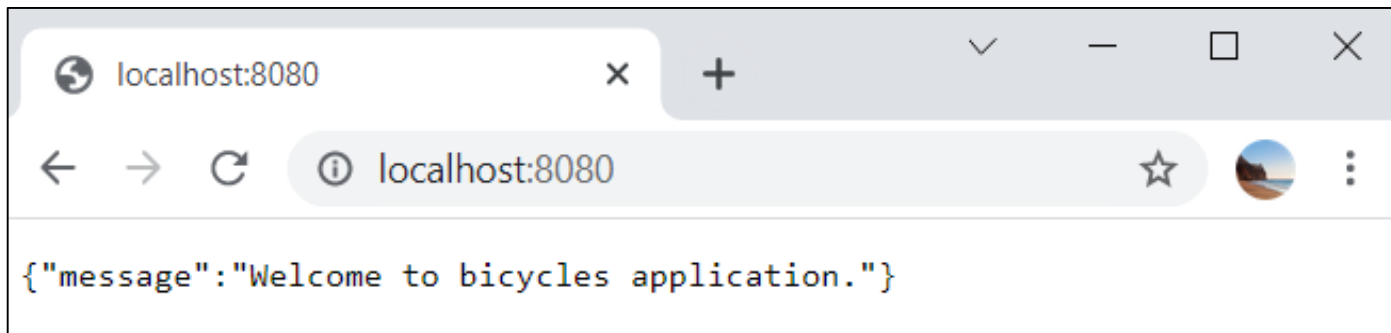
# Un simple Get con NodeJS

## 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



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

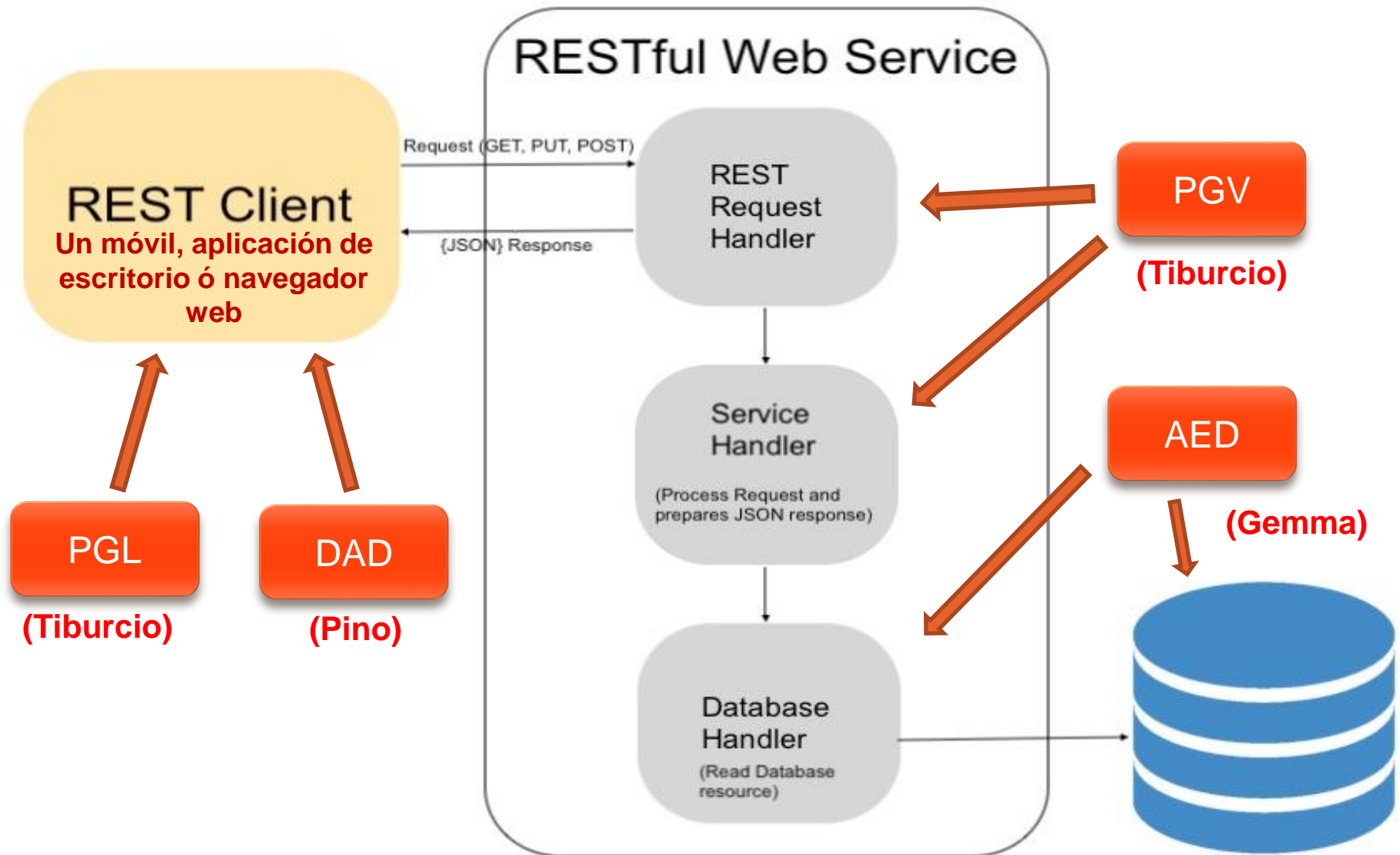
**Un objeto se corresponde con un registro**

En nuestro caso usaremos:



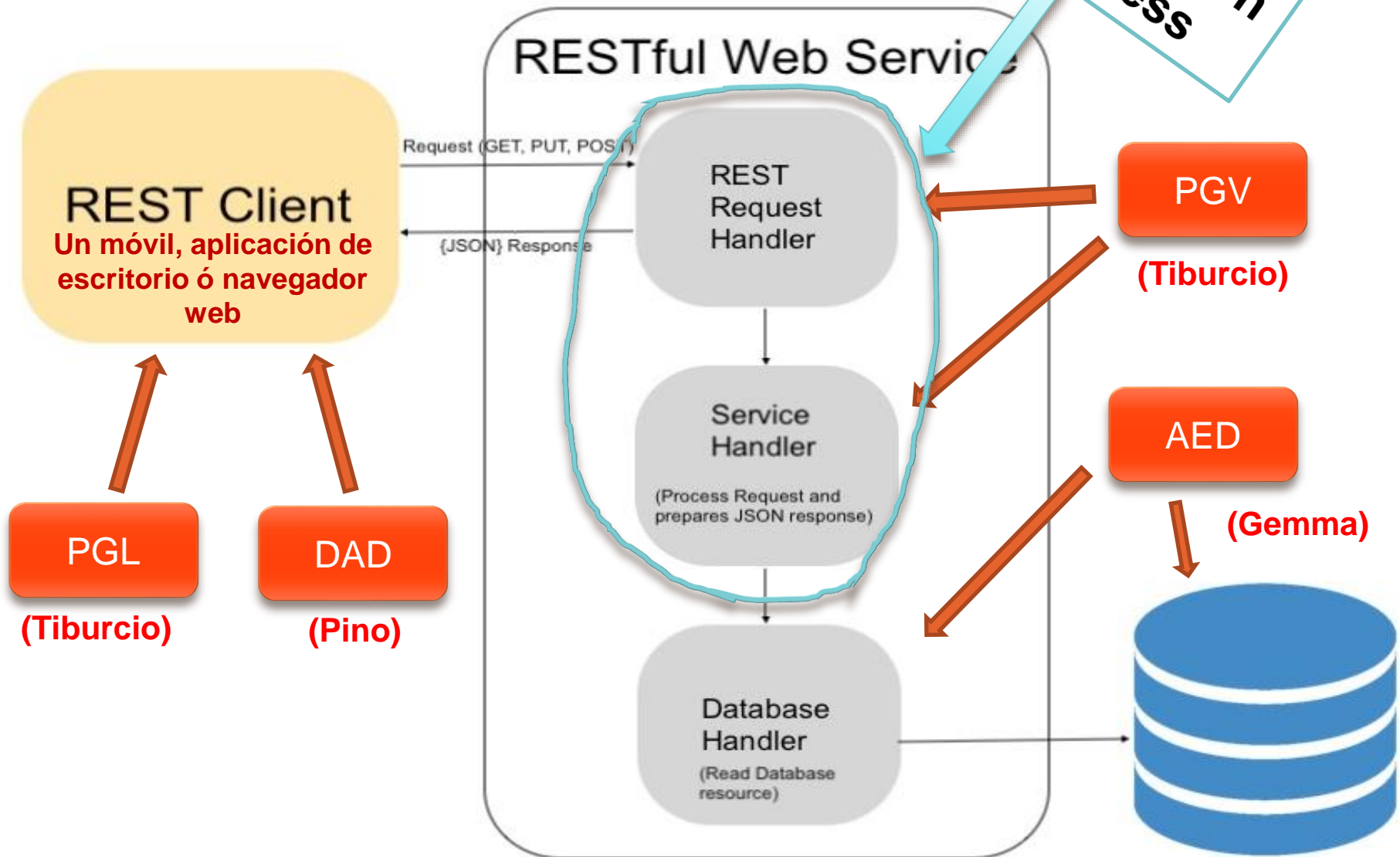
# Sequelize

# No perdamos nunca la visión global que perseguimos...

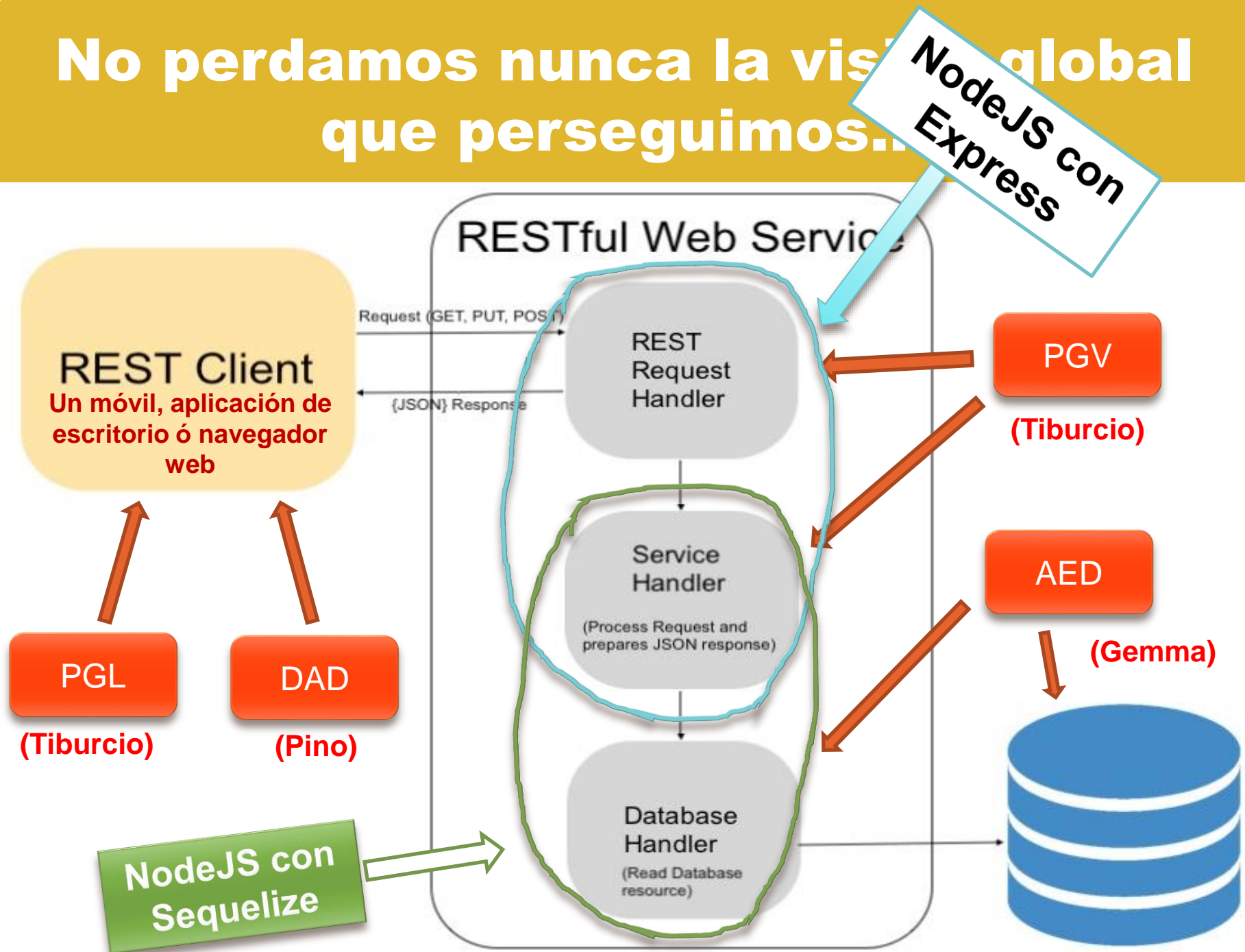


# No perdamos nunca la vista global que perseguimos...

NodeJS con Express



# No perdamos nunca la vista global que perseguimos...





# Un simple Get con NodeJS

## Instalar sequelize y mysql

Instala el paquete de sequelize y de mysql2

```
tibur@DESKTOP-02362TM MINGW64 /c/MisCosas/Casa/Bicycles/backend  
$ npm install sequelize mysql2
```

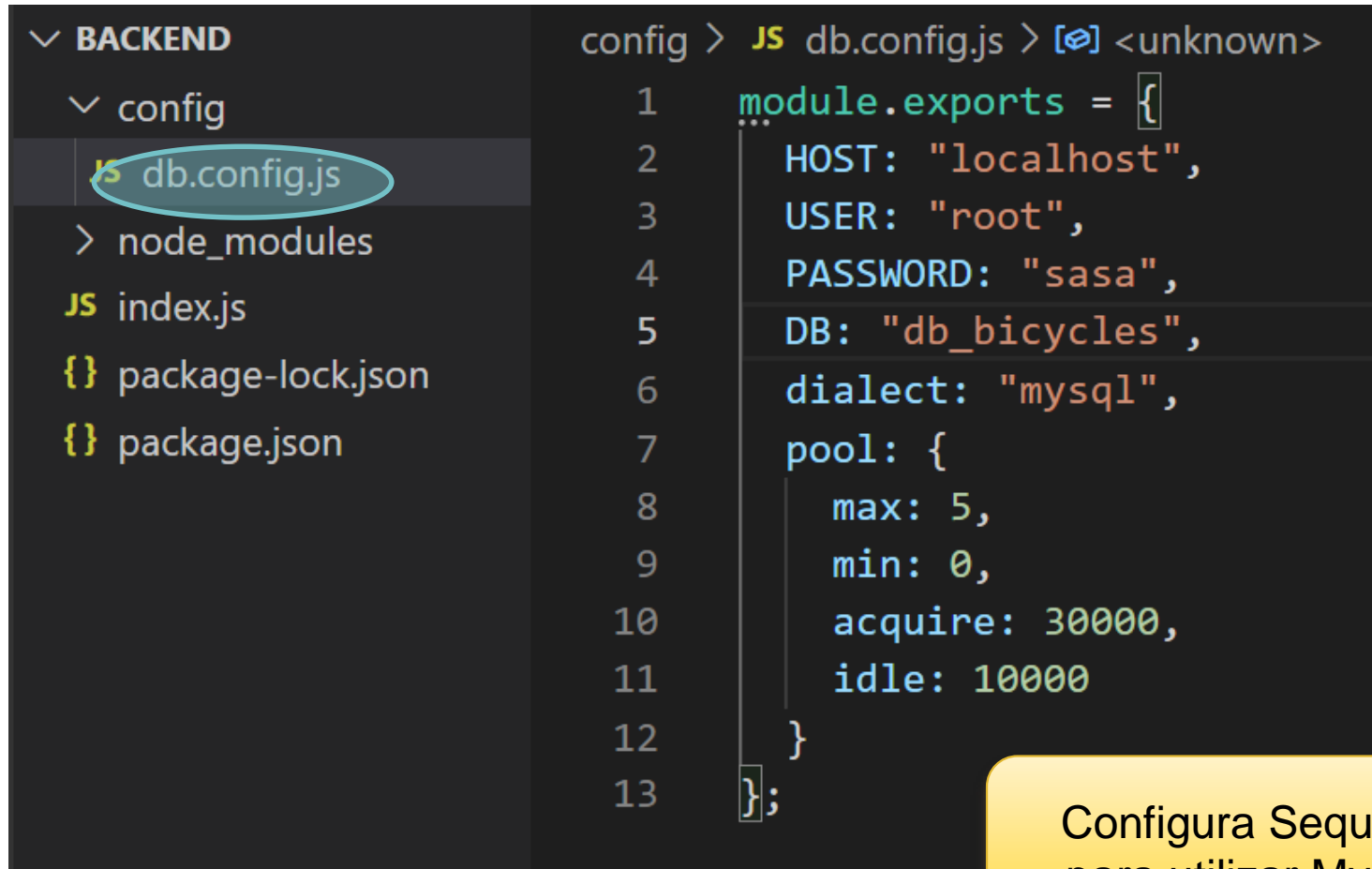
Puedes instalar varios paquetes a la vez.

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



```
config > JS db.config.js > [?] <unknown>
1  module.exports = {
2      HOST: "localhost",
3      USER: "root",
4      PASSWORD: "sasa",
5      DB: "db_bicycles",
6      dialect: "mysql",
7      pool: {
8          max: 5,
9          min: 0,
10         acquire: 30000,
11         idle: 10000
12     }
13 };
```

Configura Sequelize  
para utilizar MySQL

# Un simple Get con NodeJS

## Inicializar Sequelize

```
models > JS index.js > ...
1  const dbConfig = require("../config/db.config.js");
2
3  const Sequelize = require("sequelize");
4  const sequelize = new Sequelize(dbConfig.DB, dbConfig.USER, dbConfig.PASSWORD, {
5      host: dbConfig.HOST,
6      dialect: dbConfig.dialect,
7      operatorsAliases: false,
8
9      pool: {
10         max: dbConfig.pool.max,
11         min: dbConfig.pool.min,
12         acquire: dbConfig.pool.acquire,
13         idle: dbConfig.pool.idle
14     }
15 });
16
17 const db = {};
18
19 db.Sequelize = Sequelize;
20 db.sequelize = sequelize;
21
22 db.bicycles = require("../bicycle.model.js")(sequelize, Sequelize);
23
24 module.exports = db;
```

Inicializamos Sequelize aplicando la configuración de la transparencia anterior e indicamos que el modelo es bicycle

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

✓ BACKEND

> config

✓ models

JS index.js

> node\_modules

JS index.js

{ } package-lock.json

{ } package.json

JS index.js > ...

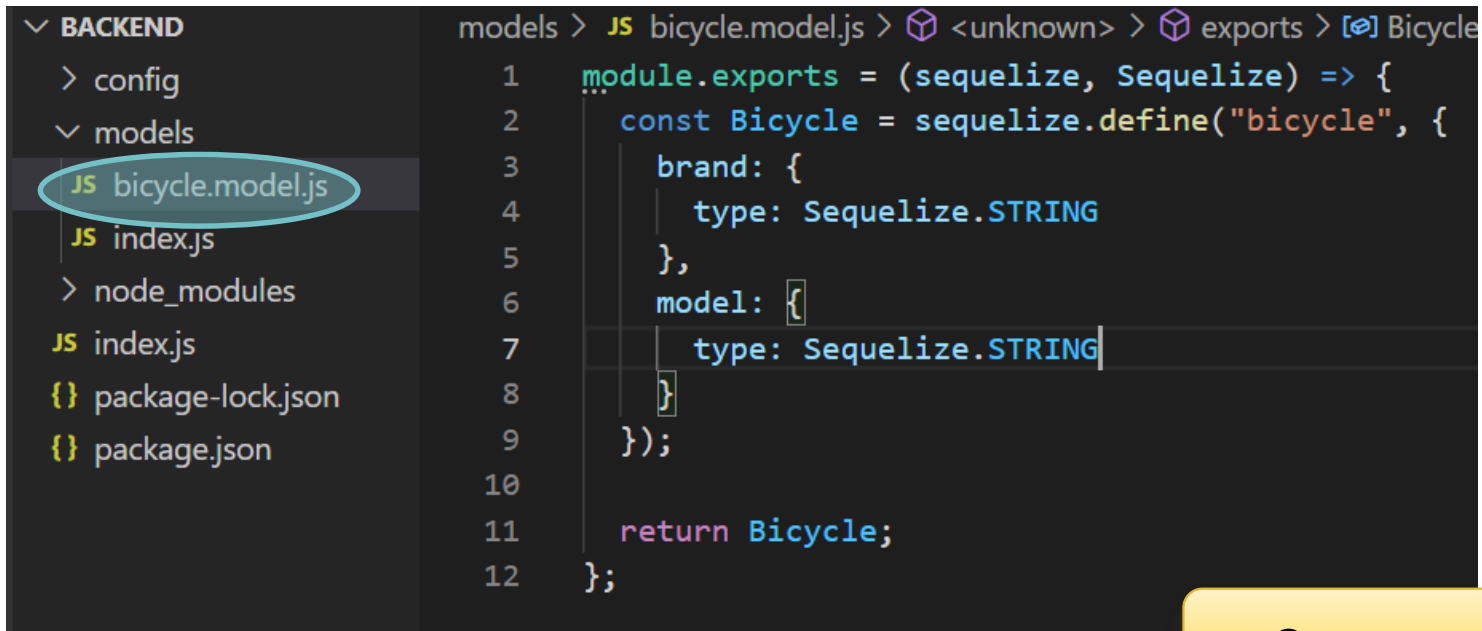
```
1  const express = require("express");
2
3  const app = express();
4
5  // parse requests of content-type - application/json
6  app.use(express.json());
7
8  // parse requests of content-type - application/x-www-form-urlencoded
9  app.use(express.urlencoded({ extended: true }));
10
11  const db = require("./models");
12  // normal use. Doesn't delete the database data
13  // db.sequelize.sync();
14
15  // In development, you may need to drop existing tables and re-sync database
16  db.sequelize.sync({ force: true }).then(() => {
17    console.log("Drop and re-sync db.");
18  });
19
20  // simple route
21  app.get("/", (req, res) => {
22    res.json({ message: "Welcome to bicycles application." });
23  });
24
25  require("./routes/bicycle.routes")(app);
26
27  // set port, listen for requests
28  const PORT = process.env.PORT || 8080;
29  app.listen(PORT, () => {
30    console.log(`Server is running on port ${PORT}.`);
31  });
```

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



```
models > JS bicycle.model.js > <unknown> > exports > Bicycle
1  module.exports = (sequelize, Sequelize) => {
2    const Bicycle = sequelize.define("bicycle", {
3      brand: {
4        type: Sequelize.STRING
5      },
6      model: {
7        type: Sequelize.STRING
8      }
9    });
10
11   return Bicycle;
12 };
```

Creamos el modelo

# Un simple Get con NodeJS

## Creemos el controlador

✓ BACKEND

- > config
- ✓ controllers
  - JS bicycle.controller.js
- ✓ models
  - JS bicycle.model.js
  - JS index.js
- > node\_modules
- JS index.js
- { } package-lock.json
- { } package.json

```
controllers > JS bicycle.controller.js > ...
1  const db = require("../models");
2  const Bicycle = db.bicycles;
3  const Op = db.Sequelize.Op;
4
5  // Create and Save a new Bicycle
6  exports.create = (req, res) => {
7  };
8
9  // Retrieve all Bicycles from the database.
10 exports.findAll = (req, res) => {
11 };
12
13 // Find a single Bicycle with an id
14 exports.findOne = (req, res) => {
15 };
16
17 // Update a Bicycle by the id in the request
18 exports.update = (req, res) => {
19 };
20
21 // Delete a Bicycle with the specified id in the request
22 exports.delete = (req, res) => {
23 };
```

Creemos el controlador

# Un simple Get con NodeJS

## Creemos el método create en el controlador

```
✓ BACKEND
  > config
  > controllers
  JS bicycle.controller.js
  > models
    JS bicycle.model.js
    JS index.js
  > node_modules
  JS index.js
  {} package-lock.json
  {} package.json

controllers > JS bicycle.controller.js > create > create
5 // Create and Save a new Bicycle
6 exports.create = (req, res) => {
7   // Validate request
8   if (!req.body.brand) {
9     res.status(400).send({
10      message: "Content can not be empty!"
11    });
12    return;
13  }
14
15  // Create a Bicycle
16  const bicycle = {
17    brand: req.body.brand,
18    model: req.body.model
19  };
20
21  // Save Bicycle in the database
22  Bicycle.create(bicycle)
23    .then(data => {
24      res.send(data);
25    })
26    .catch(err => {
27      res.status(500).send({
28        message:
29          err.message || "Some error occurred while creating the bicycle."
30      });
31    });
32  };
```

Creamos el detalle del controlador para crear una bicicleta

# Un simple Get con NodeJS

## Creemos el método findAll en el controlador



The screenshot shows a code editor with a file explorer on the left and a code editor on the right. The file explorer shows a project structure with folders for config, controllers, and models, and files for bicycle.controller.js, bicycle.model.js, index.js, package-lock.json, and package.json. The file bicycle.controller.js is selected and highlighted with a red circle. The code editor shows the implementation of the findAll method in the controller. The code is as follows:

```
controllers > JS bicycle.controller.js > findAll > findAll > then() callback
30     });
31   });
32 };
33
34 // Retrieve all Bicycles from the database.
35 exports.findAll = (req, res) => {
36   Bicycle.findAll()
37     .then(data => {
38       res.send(data);
39     })
40     .catch(err => {
41       res.status(500).send({
42         message:
43           err.message || "Some error occurred while retrieving bicycles."
44       });
45     });
46 };
47
```

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)

✓ BACKEND

> config

✓ controllers

JS bicycle.controller.js

✓ models

JS bicycle.model.js

JS index.js

> node\_modules

✓ routes

JS bicycle.routes.js

JS index.js

{ } package-lock.json

{ } package.json

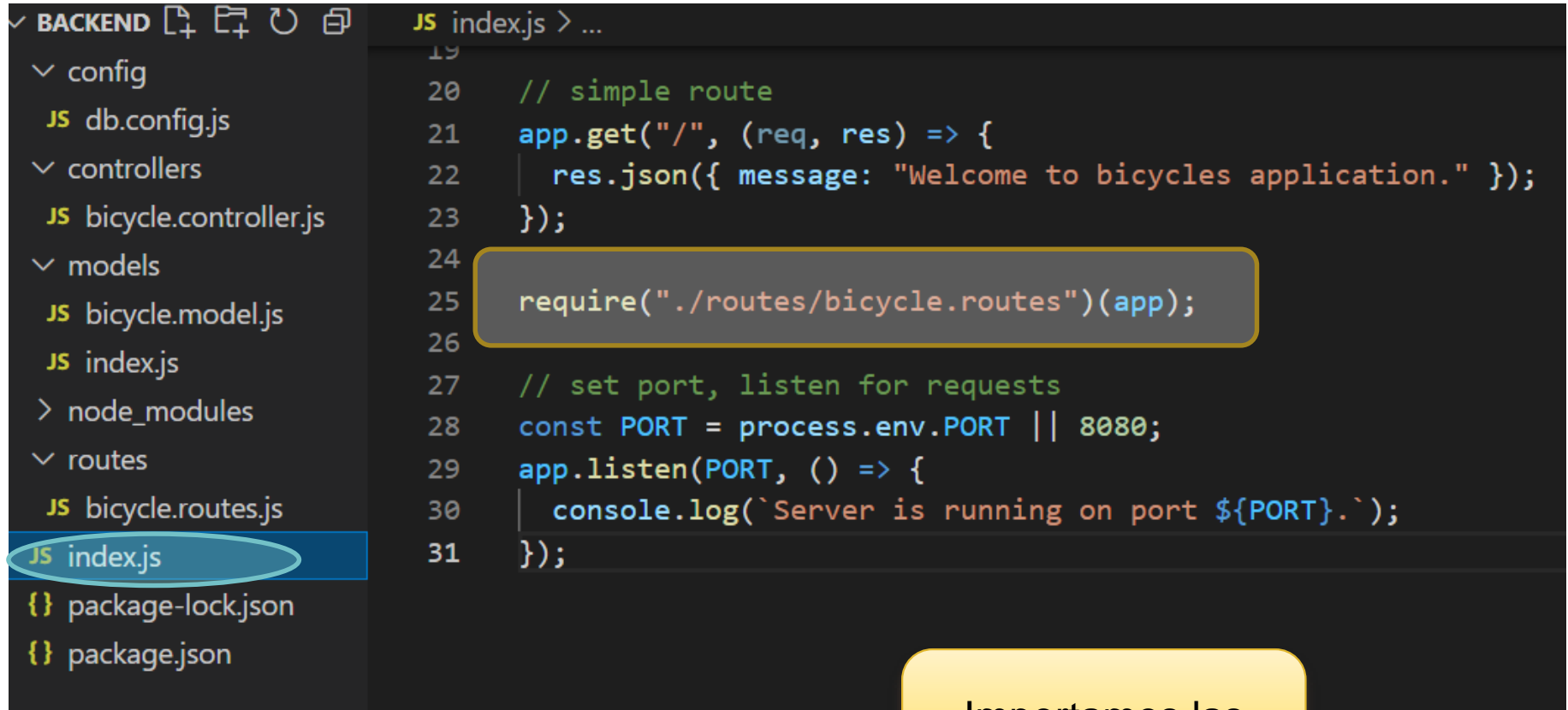
routes > JS bicycle.routes.js > <unknown> > exports

```
1 module.exports = app => {
2   const bicycles = require("../controllers/bicycle.controller.js");
3
4   var router = require("express").Router();
5
6   // Create a new Bicycle
7   router.post("/", bicycles.create);
8
9   // Retrieve all Bicycles
10  router.get("/", bicycles.findAll);
11
12  // Retrieve a single Bicycle with id
13  router.get("/:id", bicycles.findOne);
14
15  // Update a Bicycle with id
16  router.put("/:id", bicycles.update);
17
18  // Delete a Bicycle with id
19  router.delete("/:id", bicycles.delete);
20
21  app.use('/api/bicycles', router);
22 }
```

Creamos las rutas correspondientes a los end-points

# Un simple Get con NodeJS

## Importamos las rutas en index.js

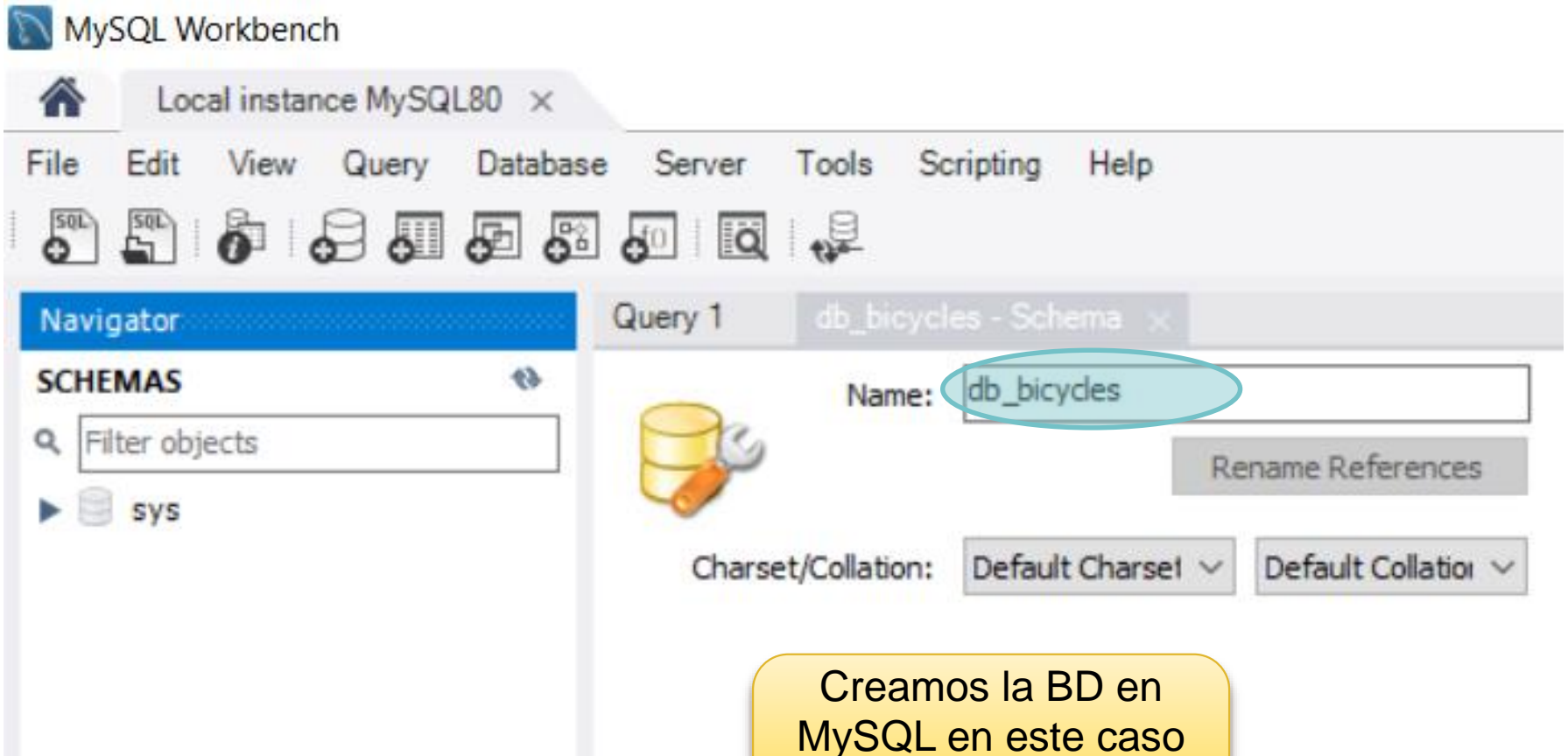


```
JS index.js > ...
19
20 // simple route
21 app.get("/", (req, res) => {
22   res.json({ message: "Welcome to bicycles application." });
23 });
24
25 require("./routes/bicycle.routes")(app);
26
27 // set port, listen for requests
28 const PORT = process.env.PORT || 8080;
29 app.listen(PORT, () => {
30   console.log(`Server is running on port ${PORT}.`);
31 });
```

Importamos las  
rutas en index.js

# Un simple Get con NodeJS

## Creamos la BD



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

## Arrancamos nuestra API

```
$ node index.js
```

(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(255), `createdAt` DATETIME NOT NULL, `updatedAt` DATETIME NOT NULL, PRIMARY KEY (`id`)) ENGINE=InnoDB;
```

```
Executing (default): SHOW INDEX FROM `bicycles`
```

Drop and re-sync db.

1

Local instance MySQL80 x

File Edit View Query Database Server Tools Scripting Help

Navigator:.....

## SCHEMAS

Filter objects

▼  db\_bicycles

▼  Tables

▼ bicycles

▶ Columns

▶  Indexes

▶ Foreign Keys

▶ **Tri**

Views

Stored Pro

## Functions

Query 1 db\_bicycles - Schema

Table Name: bicydes

Charset/Collation: utf8mb4

© 2007 The Authors

Comments:

Schema: **db bicycles**

Engine: InnoDB

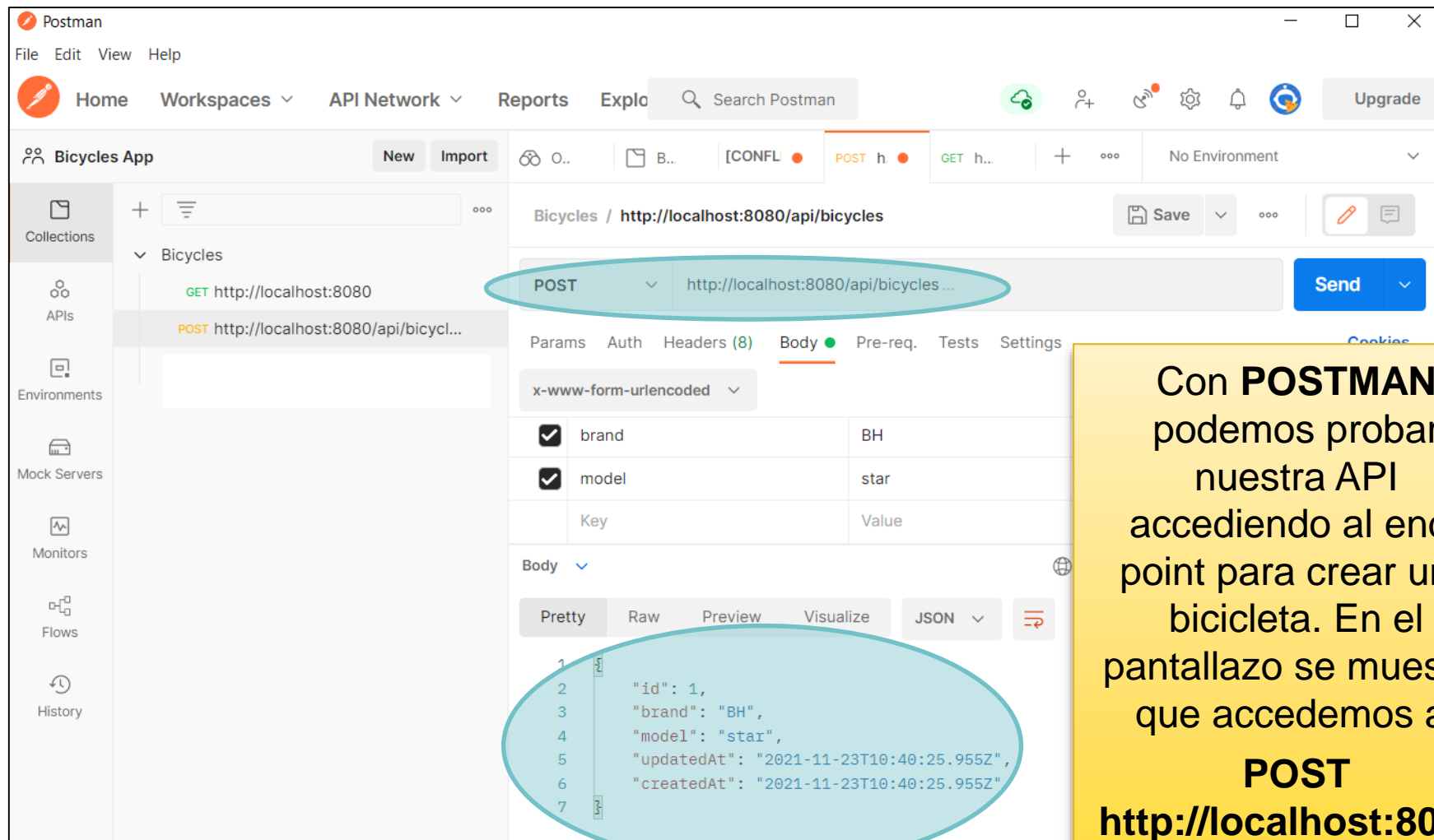
100%

[illegible]

Probemos ahora  
nuestra API usando  
**POSTMAN**

# Un simple Get con NodeJS

## Prueba tu end-point para crear una bicicleta con POSTMAN



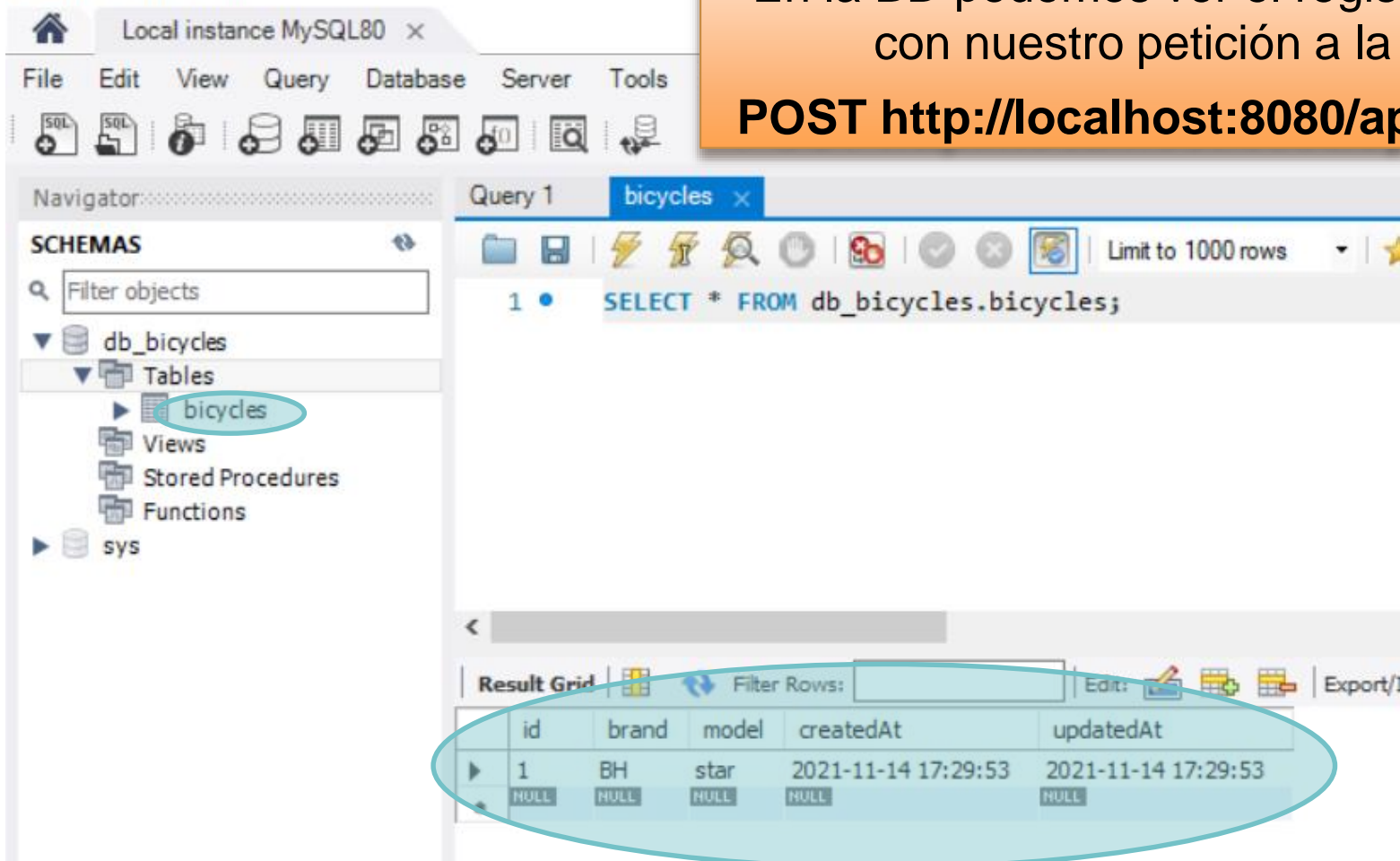
Con **POSTMAN** podemos probar nuestra API accediendo al end-point para crear una bicicleta. En el pantallazo se muestra que accedemos a:

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

# Un simple Get con NodeJS

## Prueba tu end-point para crear una bicicleta con POSTMAN

MySQL Workbench



The screenshot shows the MySQL Workbench interface. On the left, the 'SCHEMAS' pane shows the 'db\_bicycles' database with a table named 'bicycles' highlighted. The main query editor shows a query: `SELECT * FROM db_bicycles.bicycles;`. Below the query editor, the 'Result Grid' shows the following data:

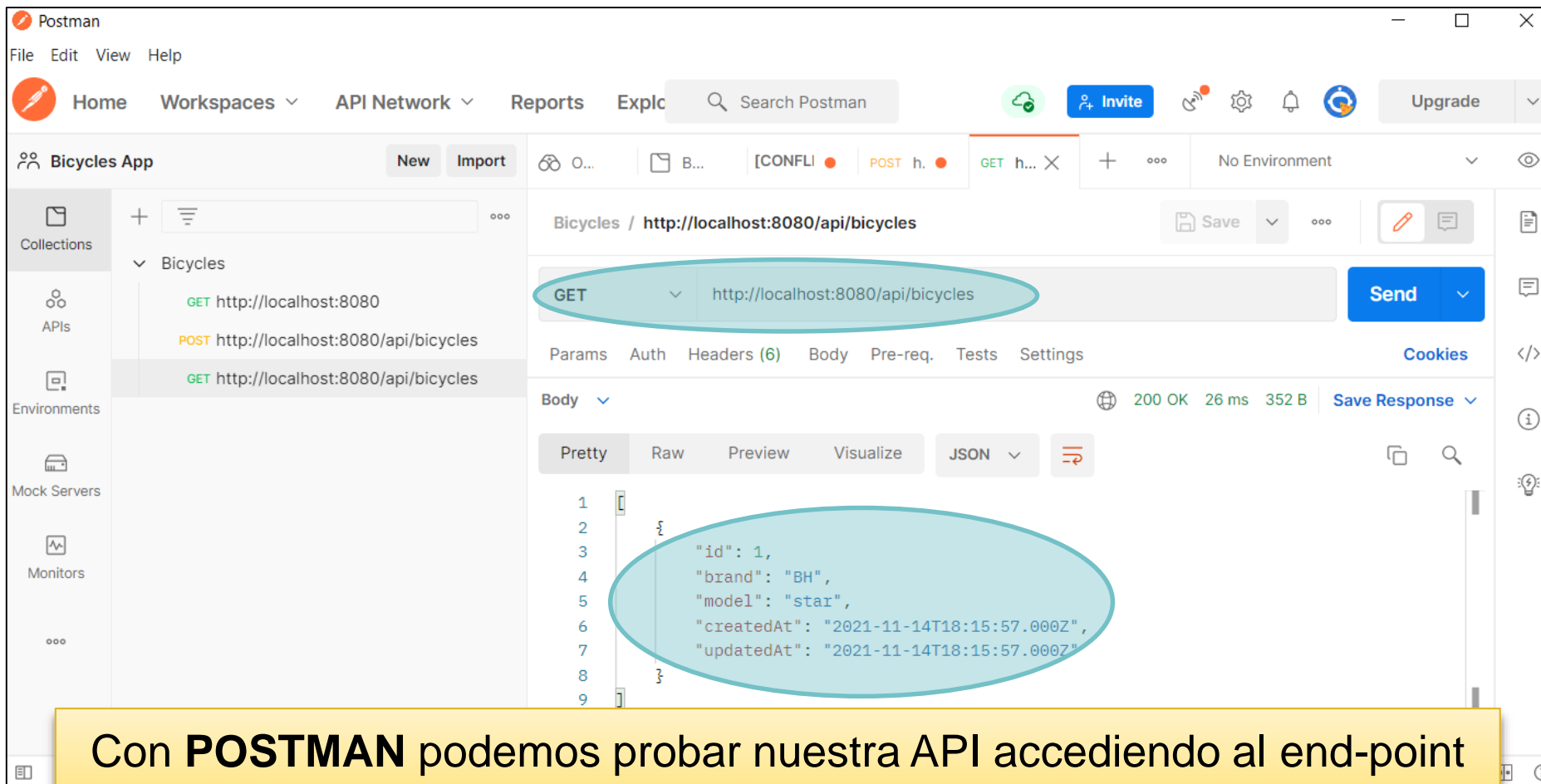
id	brand	model	createdAt	updatedAt
1	BH	star	2021-11-14 17:29:53	2021-11-14 17:29:53
	NULL	NULL	NULL	NULL

En la BD podemos ver el registro creado con nuestra 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:

- <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.
-