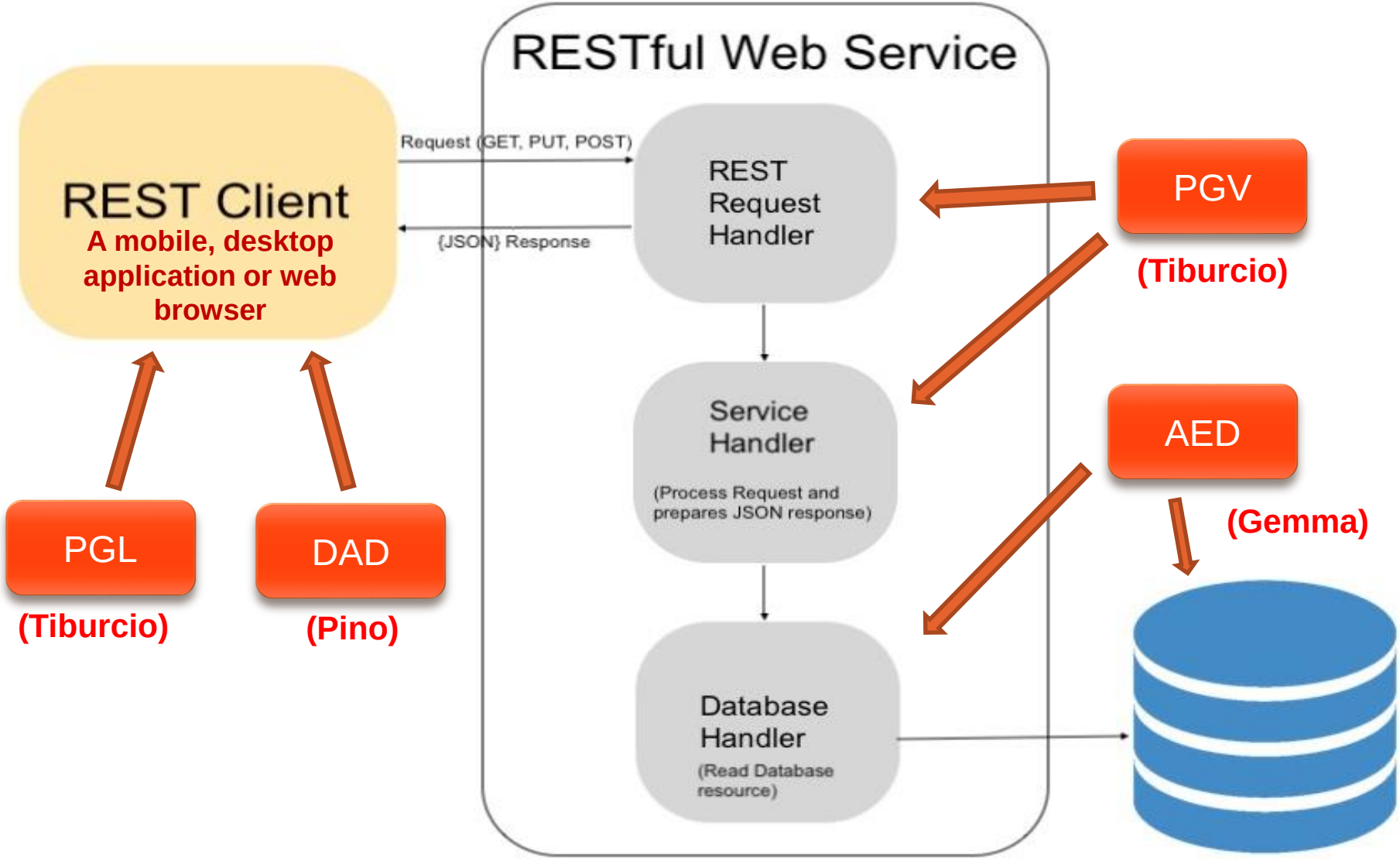
Introduction to NodeJS

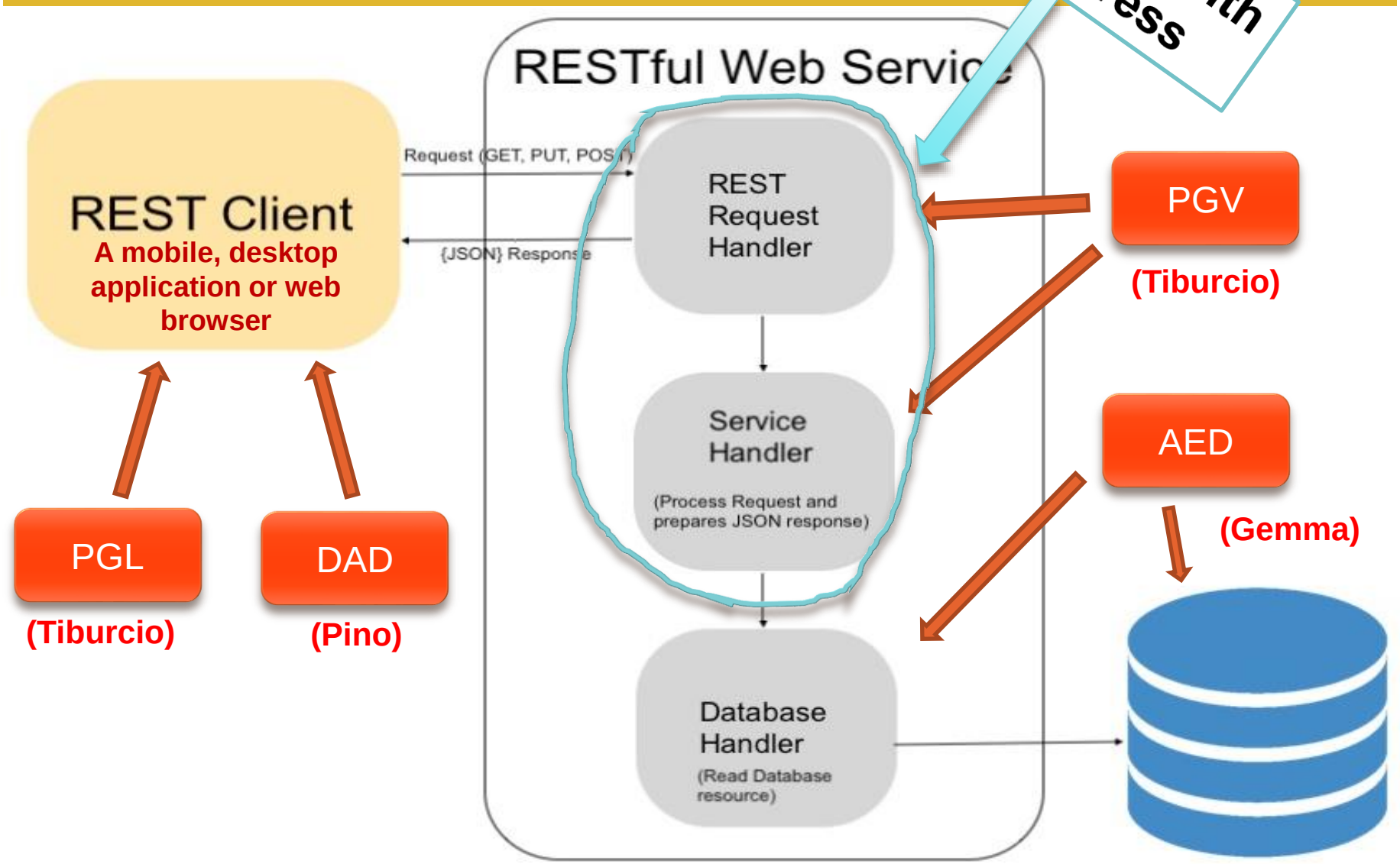
Summary of steps based on this web:

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

Let us never lose the global vision that we pursue…



Let us never lose the global vision that we pursue...



To the mess... we are going to make our first API with NodeJS...

Previous steps

On the official Node page: https://nodejs.org/es/

Download and install the LTS version of NodeJS

To the mess... we are going to make our first API with NodeJS...

What have we made so far?

We have installed:

- **NodeJS**, which is what has made possible JavaScript to run outside of the web browser.
- **npm**, which is the package manager for node. (Similar to apt for Linux)
- The commands below allow you to see the installed version.

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

A simple Get with NodeJS
npm init

Create a directory for your backend and start a project with node

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

It will ask you several things... let's see ...

A simple Get with 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)
```

Enter the name of your API. By default it is the name of the directory in which you create your project... In this case I have pressed ENTER for the default option…

A simple Get with 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:
```

It will ask you a series of questions that are self-understood... I have answered in this example with the default option simply by pressing ENTER….

A simple Get with 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
$ 
```

In the last question, answer ENTER again ... to answer that everything is OK with which the **package.json** file will have been created for you, which collects all the information entered in JSON format..

A simple Get with NodeJS
And now we install 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
```

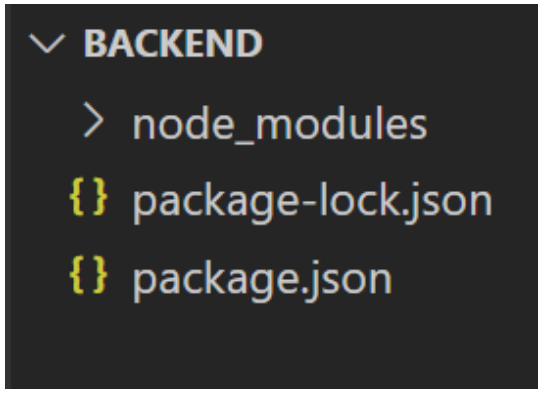
PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL

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

added 50 packages, and audited 51 packages in 4s

found 0 vulnerabilities

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

And now we install the package **Express** which allows us to code the **end-points** of my **API**.

After installing the package **Express** the dependecy will show up in **package.json**

A simple Get with NodeJS
Our directory should look like this now



See what you have in your project right now

package.json

When initializing your node project with “npm init” the package.json file was created

node_modules

When installing the express package the node_modules directory has been created. From now on node_modules will house all the packages that you install

A simple Get with NodeJS
Let's create 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}.`);
});
```

Create the file **index.js** and copy this code in it

A simple Get with NodeJS
Let's create 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}.`);
});
```

Import the package **express**

A simple Get with NodeJS
Let's create 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}.`);
});
```

We start using express using the constant **app**

A simple Get with NodeJS
Let's create 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}.`);
});
```

We can use the express library for content application/json y application/x-www-form-urlencoded

We will see it a little later

A simple Get with NodeJS
Let's create 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}.`);
});
```

With these lines we start our API that will listen on port 8080

A simple Get with NodeJS
Let's create 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}.`);
});
```

And the most important:

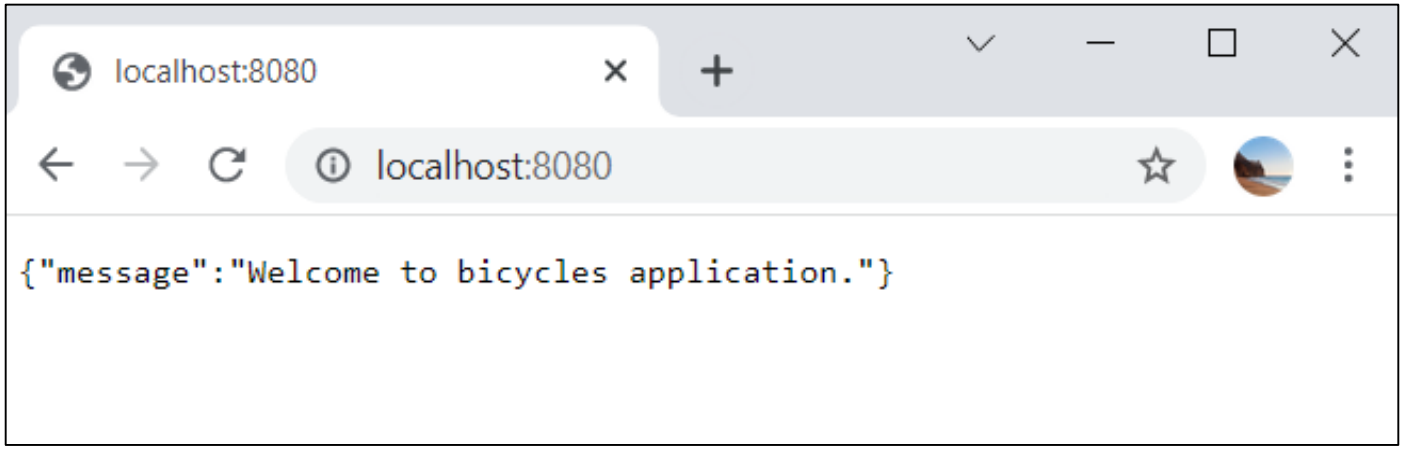
We have an end-point which listens on:
http://localhost:8080/

And it will return a little message in JSON format…

A simple Get with NodeJS
Let's boot our API

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

Our API is now listening on
http://localhost:8080

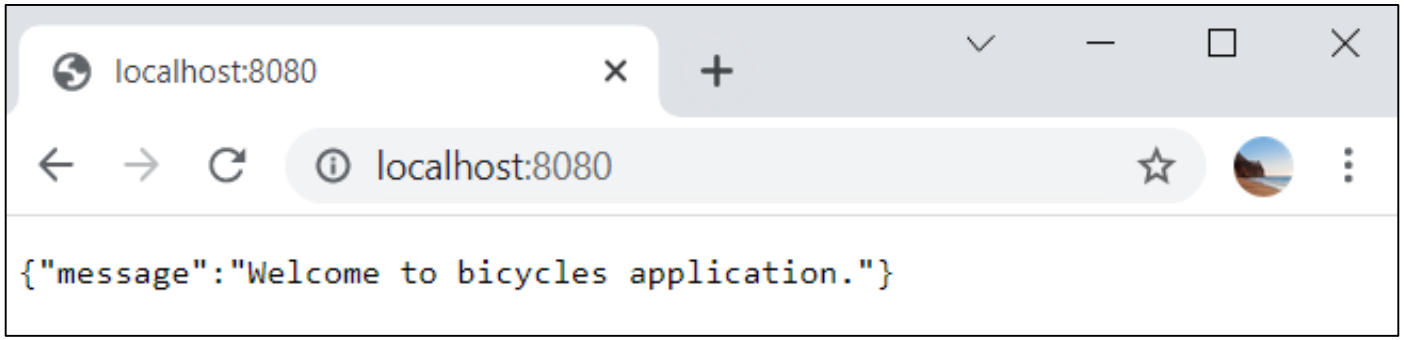


A simple Get with NodeJS
Our API has an end-point

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

Keep in mind that we are really accessing an end-point, which in this case is:

GET http://localhost:8080/

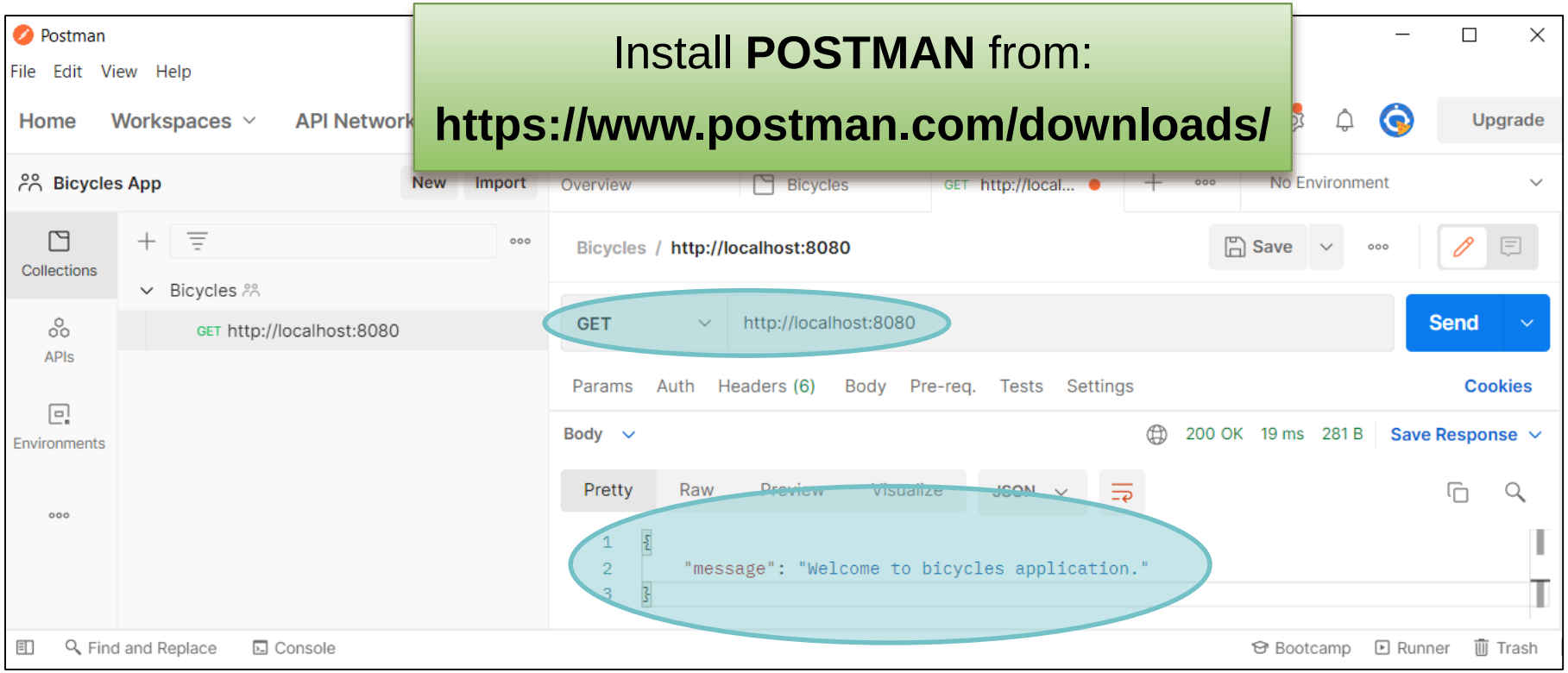


A simple Get with NodeJS

Test your end-points with POSTMAN

Install **POSTMAN** from:

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



With **POSTMAN** we can test our API accessing to its end-points. The screenshot shows that we access:

GET http://localhost:8080/

A simple Get with NodeJS

Now let's go for the ORM...

ORM (Object Relationship Mapping) allows in practice to create an object-oriented database.

To program you can use objects that the ORM will automatically save in DB records

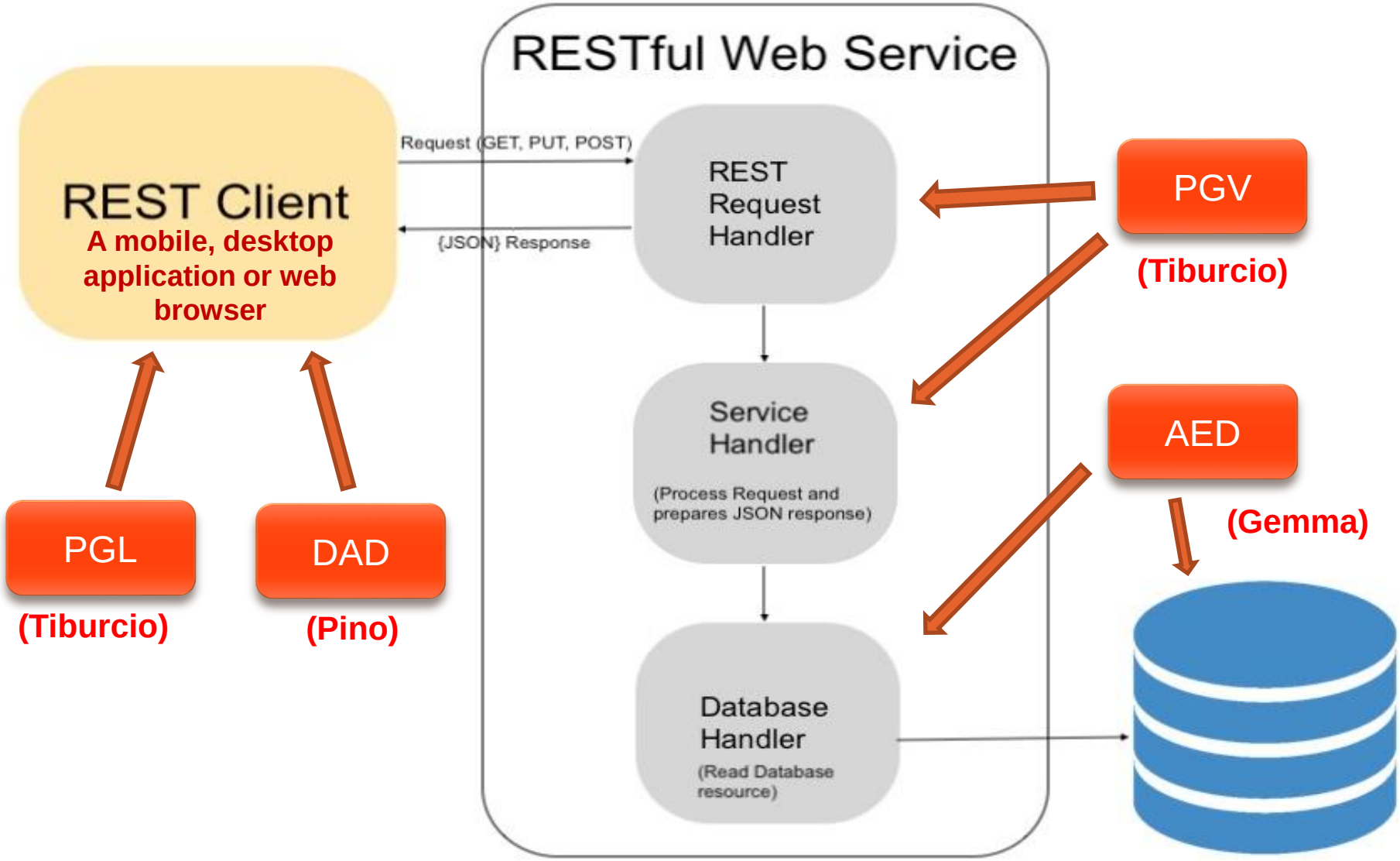
A class corresponds to a table

An object corresponds to a record

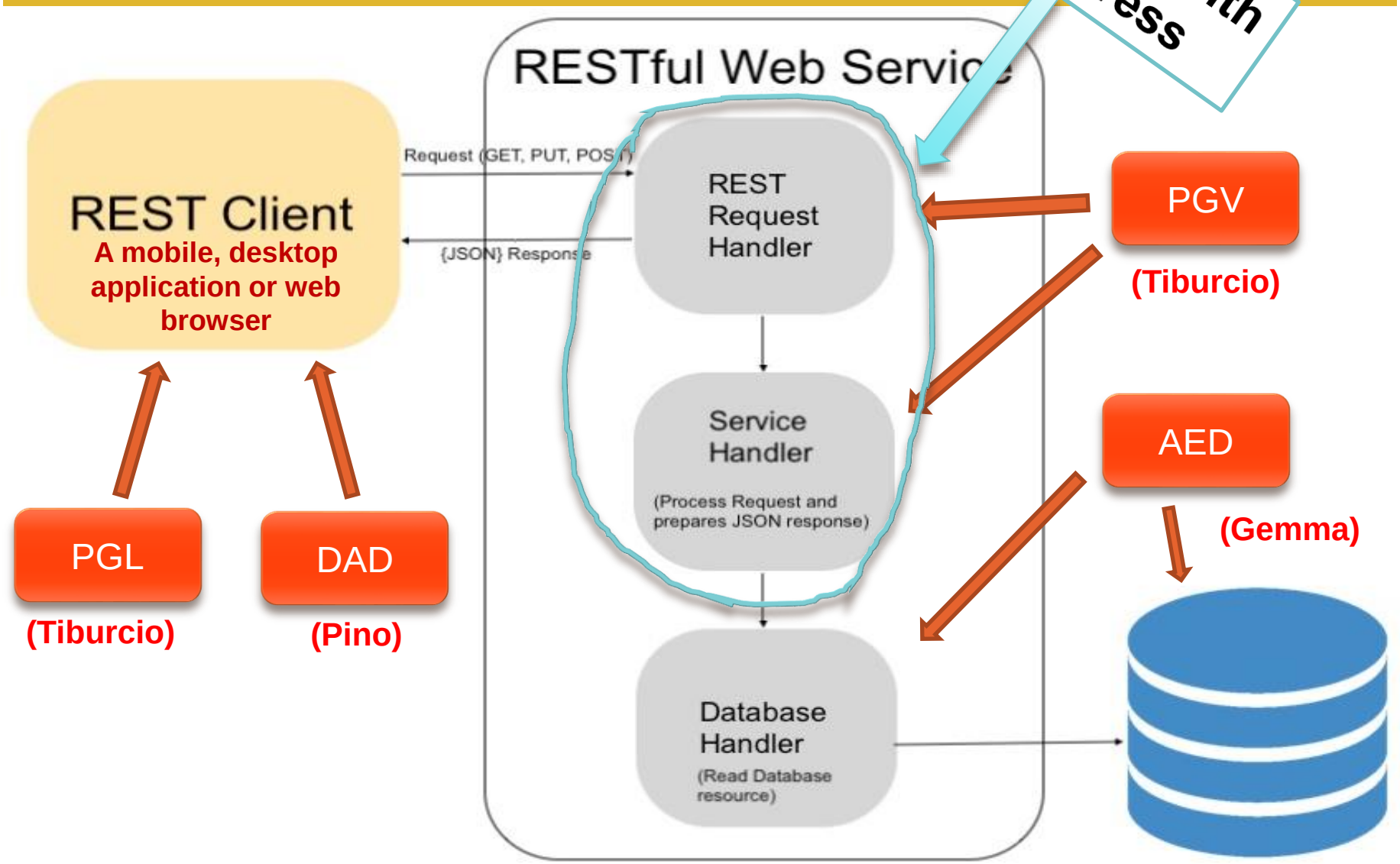
In our case we will use:

Sequelize

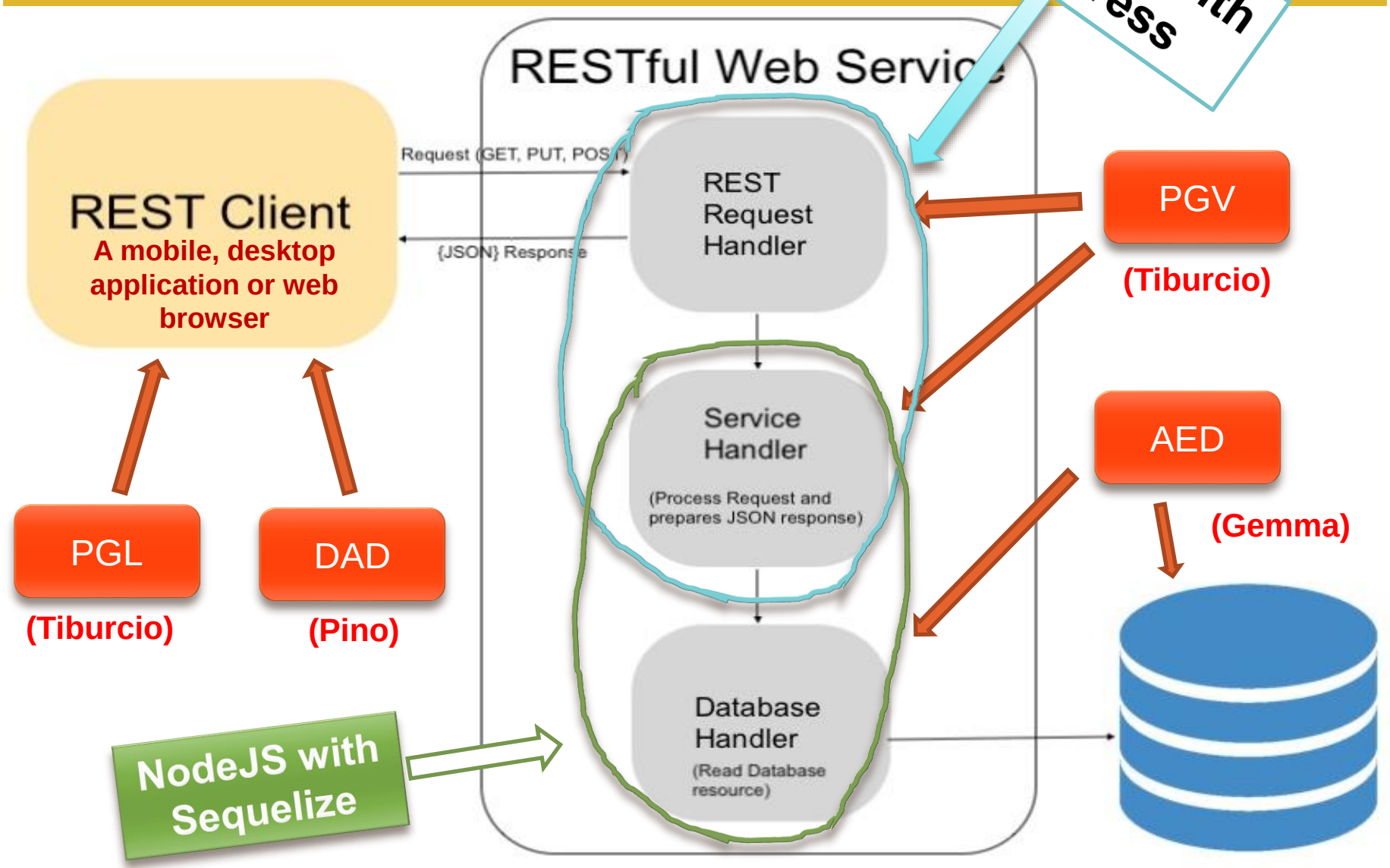
Let us never lose the global vision that we pursue…



Let us never lose the global vision that we pursue...



Let us never lose the global vision that we pursue...



A simple Get with NodeJS
Install sequelize and mysql

Install the packages sequelize and mysql2

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

You can install several packages at once.

Here we have installed sequelize and mysql2 at once.

Sequelize is the ORM.

Mysql2 is the package to use mysql with sequelize.

A simple Get with NodeJS Using the Sequelize ORM

BACKEND
- config
  - db.config.js
- node_modules
- index.js
- package-lock.json
- package.json

config > db.config.js > <unknown>

```js
module.exports = {
  HOST: "localhost",
  USER: "root",
  PASSWORD: "sasa",
  DB: "db_bicycles",
  dialect: "mysql",
  pool: {
    max: 5,
    min: 0,
    acquire: 30000,
    idle: 10000
  }
};
```

Configure Sequelize to use it with MySQL

A simple Get with NodeJS
Initialize Sequelize

BACKEND
- config
- models
  - index.js
- node_modules
- index.js
- package-lock.json
- package.json

models > 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;
```

We initialize Sequelize applying the previous transparency settings and indicate that the model is bicycle

A simple Get with NodeJS Sync (force or not force)

BACKEND
- config
- models
  - index.js
- node_modules
- index.js
- package-lock.json
- package.json

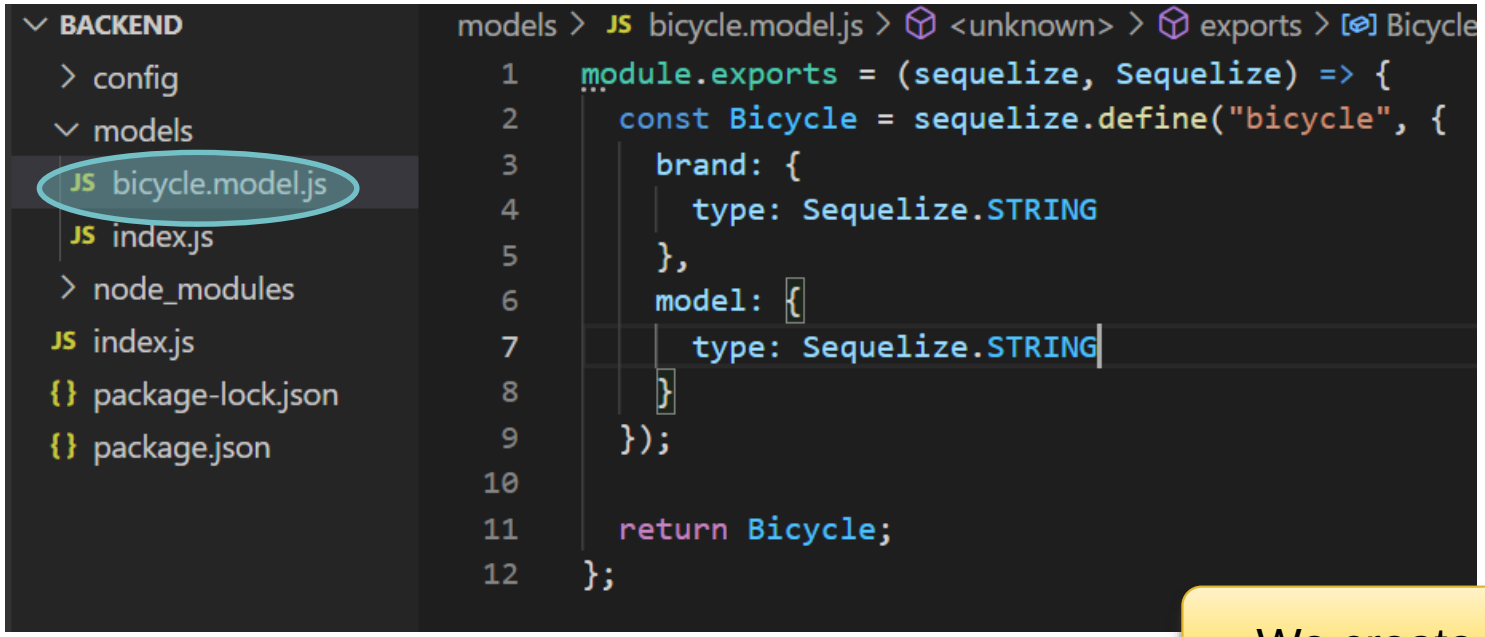
index.js > ...

```
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 });
```

We initialize Sequelize applying the previous transparency settings

Using force: true will delete the existing tables and create them again

A simple Get with NodeJS
Let's create the model



We create the model

A simple Get with NodeJS
Let's create the controller

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

controllers > JS bicycle.controller.js > ...

```js
const db = require("../models");
const Bicycle = db.bicycles;
const Op = db.Sequelize.Op;

// Create and Save a new Bicycle
exports.create = (req, res) => {
};

// Retrieve all Bicycles from the database.
exports.findAll = (req, res) => {
};

// Find a single Bicycle with an id
exports.findOne = (req, res) => {
};

// Update a Bicycle by the id in the request
exports.update = (req, res) => {
};

// Delete a Bicycle with the specified id in the request
exports.delete = (req, res) => {
};
```

We create the controller

A simple Get with NodeJS

Let's create the method create in the controller

BACKEND
- config
- controllers
  - bicycle.controller.js
- models
  - bicycle.model.js
  - index.js
- node_modules
- index.js
- package-lock.json
- package.json

controllers > bicycle.controller.js > create > create

```
// Create and Save a new Bicycle
exports.create = (req, res) => {
  // Validate request
  if (!req.body.brand) {
    res.status(400).send({
      message: "Content can not be empty!"
    });
    return;
  }

  // Create a Bicycle
  const bicycle = {
    brand: req.body.brand,
    model: req.body.model
  };

  // Save Bicycle in the database
  Bicycle.create(bicycle)
    .then(data => {
      res.send(data);
    })
    .catch(err => {
      res.status(500).send({
        message:
          err.message || "Some error occurred while creating the bicycle."
      });
    });
};
```

We create the controller details to create a bicycle

A simple Get with NodeJS

Let's create the method findAll in the controller

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

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
```

We create the controller details to show all the bikes

This way you could add the rest of the controller methods:
https://www.bezkoder.com/node-js-express-sequelize-mysql/

A simple Get with NodeJS

Now we create the routes (the end-points)

We create the routes corresponding to the end-points

A simple Get with NodeJS

Let’s import the routes into index.js

BACKEND

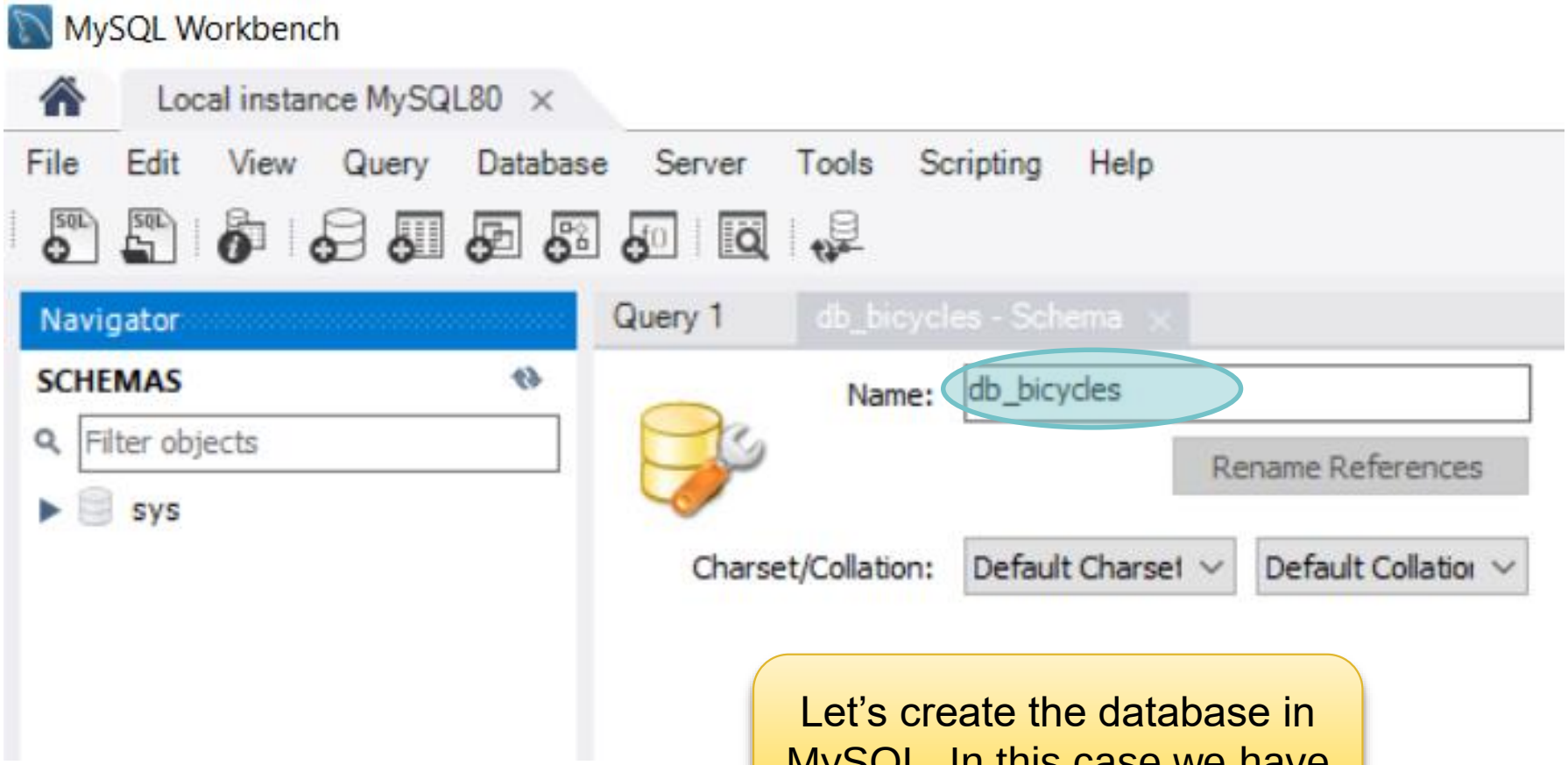
- config
  - db.config.js
- controllers
  - bicycle.controller.js
- models
  - bicycle.model.js
  - index.js
- node_modules
- routes
  - bicycle.routes.js
- index.js
- package-lock.json
- package.json

index.js > ...

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

require("./routes/bicycle.routes")(app);

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

Let’s import the routes into index.js

A simple Get with NodeJS
Let's create the Database



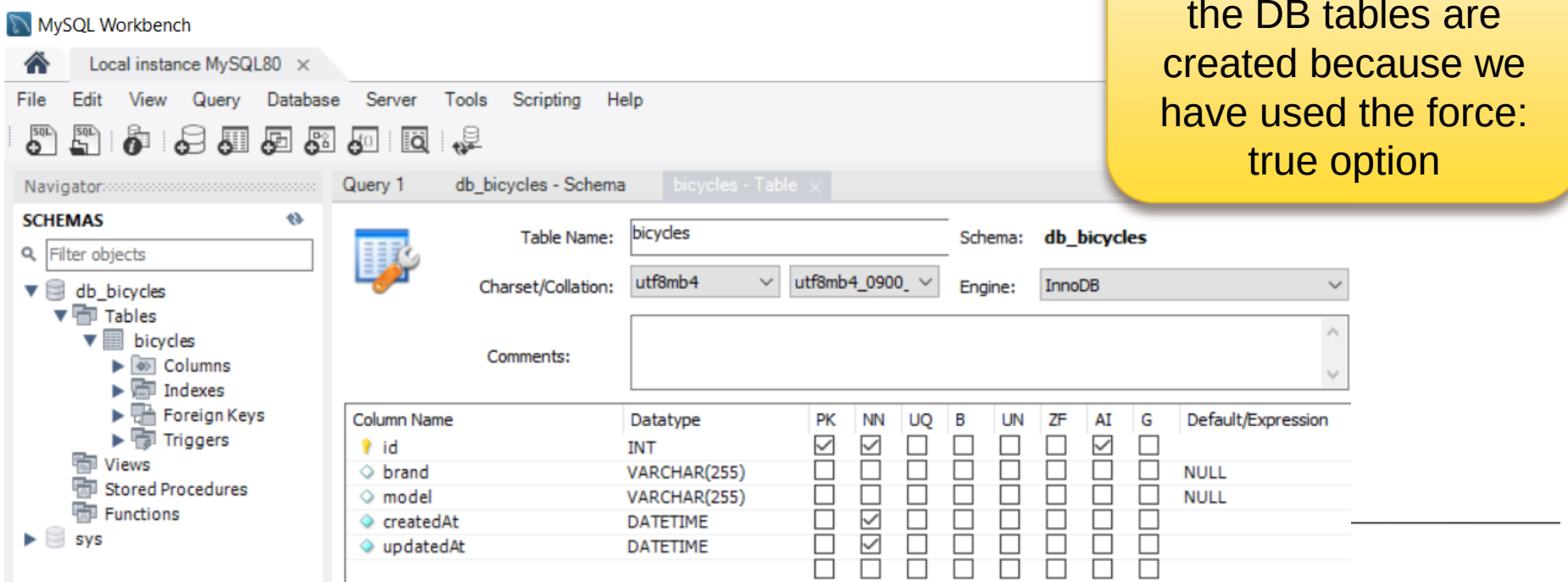
Let's create the database in MySQL. In this case we have used MySQL Workbench

A simple Get with NodeJS

Let's boot our 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.
```

When starting our API, the DB tables are created because we have used the force: true option



Let’s try our API now using

POSTMAN

A simple Get with NodeJS

Test your end-point to create a bike with POSTMAN

Postman

File Edit View Help

Home Workspaces API Network Reports Explo Search Postman Upgrade

Bicycles App New Import

O.. B.. [CONFL POST h GET h.. No Environment

Collections

APIs

Environments

Mock Servers

Monitors

Flows

History

Bicycles

GET http://localhost:8080

POST http://localhost:8080/api/bicycl...

Bicycles / http://localhost:8080/api/bicycles Save

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

Params Auth Headers (8) Body Pre-req. Tests Settings Cookies

x-www-form-urlencoded

☑	brand	BH
☑	model	star
	Key	Value

Body

Pretty Raw Preview Visualize JSON

```
{
    "id": 1,
    "brand": "BH",
    "model": "star",
    "updatedAt": "2021-11-23T10:40:25.955Z",
    "createdAt": "2021-11-23T10:40:25.955Z"
}
```

With **POSTMAN** we can test our API accessing to the end-point to create a bike. In the screenshot we are accessing the end-point:

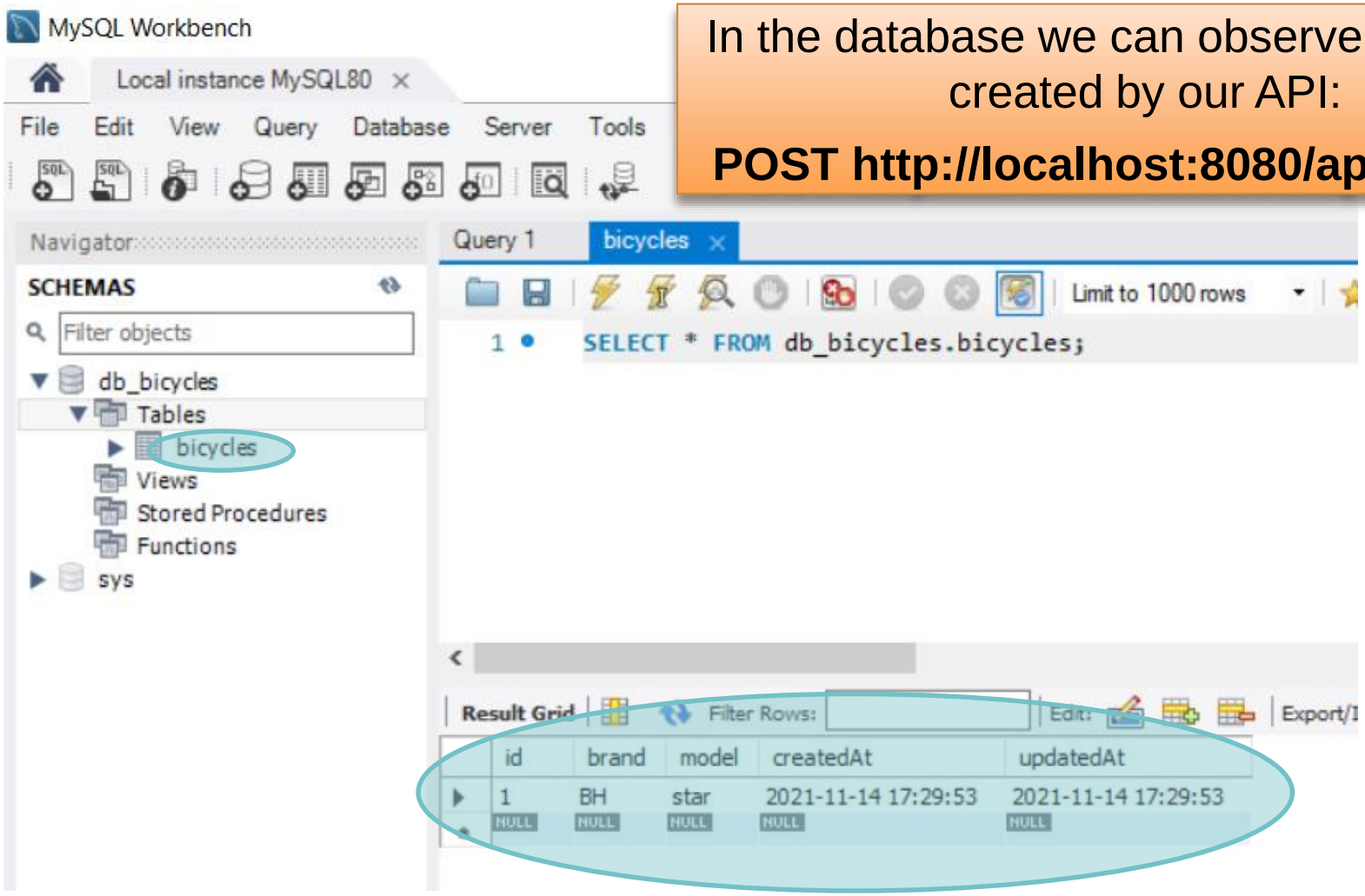
POST http://localhost:8080/api/bicycles

A simple Get with NodeJS

Test your end-point to create a bike with POSTMAN

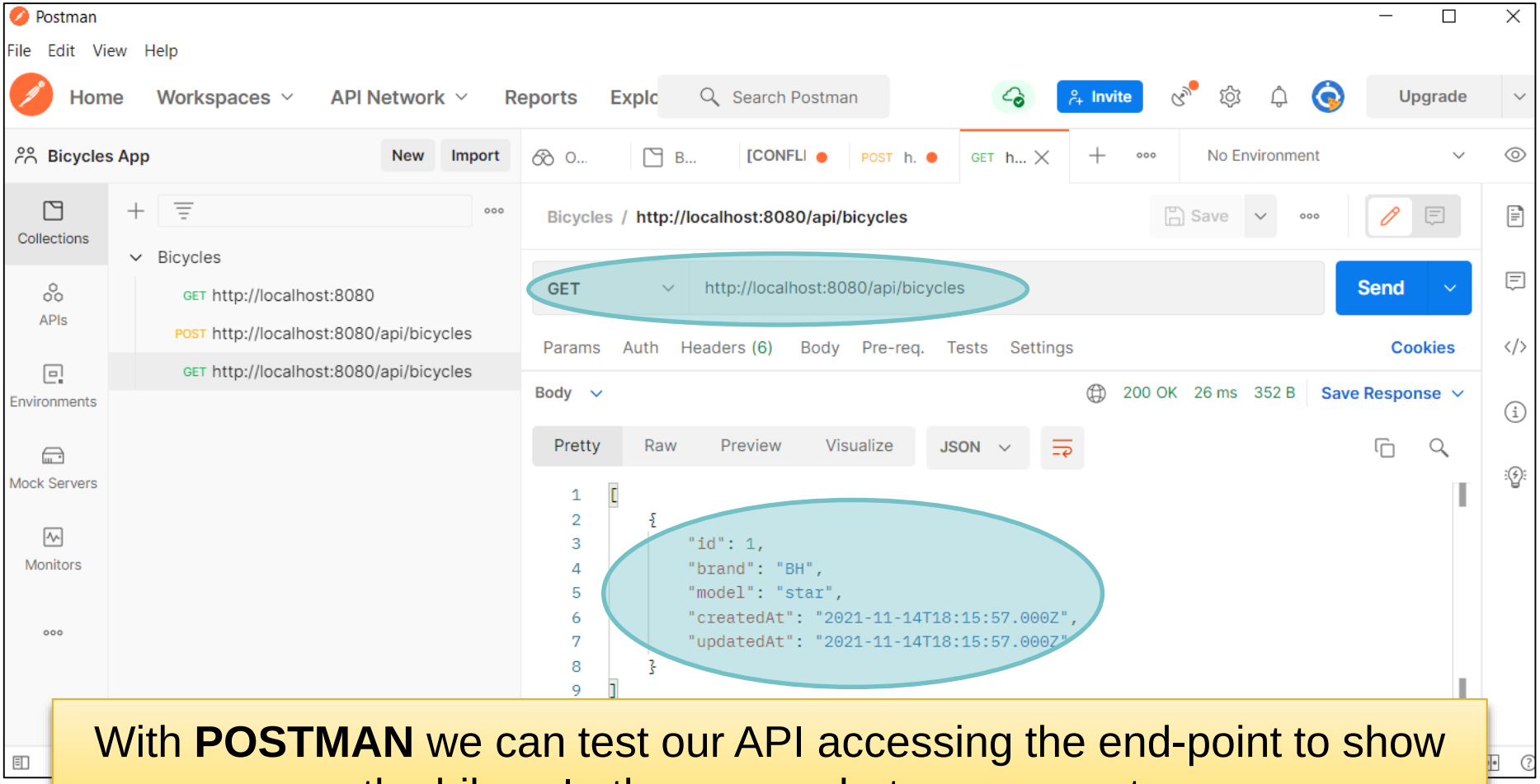
In the database we can observe the record created by our API:

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



A simple Get with NodeJS

Now try your end-point to show the bikes with POSTMAN



With **POSTMAN** we can test our API accessing the end-point to show the bikes. In the screenshot we access to:

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

Keep on learning...

Follow the following example step by step which is actually the one I have followed to add all the missing controller code and do all the tests:

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

If you want a simpler example than the one we have done, you can see the following video:

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

Conclussions

What have we learned so far?

- We have installed NodeJS.
- We have created an API to create 3 end-points: 1 POST and 2 GETs.
- We have test our API using POSTMAN.

Next steps…

- Finish the CRUD and test with POSTMAN all the end-points.
- Add relationships one-to-many, many-to-many and one-to-one.
- Add Authentication to our API.