

[Open in app](#)[Get started](#)

Tomás Malio

[Follow](#)

May 22, 2020 · 13 min read

[Save](#)

# Node JS, Express y MySQL con Sequelize

## Requerimientos

- **Node.js:** es un entorno de código abierto, multi-plataforma, entorno de ejecución de JavaScript que ejecuta código JavaScript fuera de un navegador web.
- **Express o Express.js:** es un marco de aplicación web para Node.js, lanzado como software gratuito y de código abierto bajo la Licencia MIT. Está diseñado para crear aplicaciones web y API. Se le ha llamado el marco de servidor estándar para Node.js.
- **NPM:** es un administrador de paquetes para el lenguaje de programación JavaScript. Es el administrador de paquetes predeterminado para el entorno de tiempo de ejecución de JavaScript Node.js.  
Consiste en un cliente (línea de comando), también llamado npm, y una base de datos en línea de paquetes públicos y privados pagos, llamado registro npm.



[Open in app](#)[Get started](#)

- **Postman:** es un entorno de desarrollo de APIs que nos permite diseñar, probar y monitorizar servicios REST.

## Comenzando

Para comenzar a desarrollar nuestro proyecto con Node.js, primero vamos abrir nuestro IDE de trabajo, en mi caso, voy a utilizar Visual Studio Code.

Una vez que hemos ingresado en nuestro IDE y creado un espacio de trabajo, lo que vamos a realizar algunos comandos con NPM.

**Importante:** si no están seguros de tener Node y NPM. Les recomiendo realizar las siguientes verificaciones desde la terminal.

```
node -v
npm -v
```

```
MacBookTomas:ejemplo-sequelize tomasmalio$ node -v
v10.15.3
MacBookTomas:ejemplo-sequelize tomasmalio$ npm -v
6.14.5
MacBookTomas:ejemplo-sequelize tomasmalio$ █
```

Validando si tenemos en nuestra computadora Node y NPM

El segundo punto de control, es controlar que estemos donde tenemos que estar 😊. Vamos a validar que estamos dentro de la carpeta que hemos creado del proyecto:

```
pwd
/Users/tomasmalio/development/ejemplo-sequelize
```

Es muy importante que siempre verifiquemos **TODO** para no tener errores que nos vuelven locos 🤪

## Creando el paquete

Ya estamos listos para comenzar con nuestro proyecto, y lo primero que vamos hacer es



[Open in app](#)[Get started](#)

El **comando init** nos va a desplegar un montón de campos para completar o dejar por defecto:

- name
- description
- version
- **entry point: (index.js)** ingresen app.js

*Ejemplo de cómo quedaría el package.json:*

```
{  
  "name": "ejemplo-sequelize",  
  "version": "1.0.0",  
  "description": "",  
  "main": "app.js",  
  "scripts": {  
    "test": "echo \\\"Error: no test specified\\\" && exit 1"  
  },  
  "author": "Tomas Malio",  
  "license": "ISC",  
}
```

## Instalando Express

Es el momento de instalar Express y algunas dependencias necesarias para que todo funcione.

```
npm install --save express body-parser morgan
```

El indicador “**save**”, guardará estos paquetes en la sección de dependencias de su archivo **package.json** que han creado anteriormente.

Ahora es momento de **crear** nuestro archivo **app.js** en la carpeta raíz. En este archivo, vamos a crear nuestra aplicación Express.



[Open in app](#)[Get started](#)

```
const bodyParser = require('body-parser');

// This will be our application entry. We'll setup our server here.
const http = require('http');

// Set up the express app
const app = express();

// Log requests to the console.
app.use(logger('dev'));

// Parse incoming requests data (https://github.com/expressjs/body-parser)
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended: false }));

// Setup a default catch-all route that sends back a welcome message
// in JSON format.
app.get('*', (req, res) => res.status(200).send({
  message: 'Welcome to the beginning of nothingness.',
}));

const port = parseInt(process.env.PORT, 10) || 8000;
app.set('port', port);

const server = http.createServer(app);
server.listen(port);

module.exports = app;
```

La aplicación se ejecutará correctamente con el puerto 8000. Ahora necesitamos una forma de reiniciar el servidor cada vez que cambiemos algo en nuestro código.

Para eso, utilizaremos el excelente paquete **nodemon** npm. Así que a continuación ejecutaremos el siguiente comando en nuestra terminal:

```
npm i -D nodemon
```

Luego de ejecutar el comando, abra su archivo **package.json** y cree un comando para ejecutar el servidor. Ese comando se creará en la sección de secuencias de comandos. Edite su **package.json** en la sección de scripts de la siguiente manera:



[Open in app](#)[Get started](#)

Si llegamos a este punto... estamos muy bien 🎸



Los Rolling Stones

## El primer paso

*Es un pequeño paso para el hombre, pero un gran paso para la humanidad.*

```
npm start
```

Como resultado tendremos un mensaje como el siguiente:



[Open in app](#)[Get started](#)

```
[nodemon] 2.0.4
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): ***!
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node app.js`
```

y si visitamos <http://localhost:8000> van poder observar un mensaje así:

```
{
  message: "Welcome to the beginning of nothingness."
}
```

En este momento su proyecto debería tener una **estructura** como la siguiente:

```
mi-proyecto/
├── node_modules/
├── app.js
├── package-lock.json
└── package.json
```

## Sequelize

Llego el momento tan esperado  , vamos a pasar a instalar Sequelize, que como mencionamos al comienzo, es un ORM para Postgres, MySQL, MariaDB, SQLite y SQL Server.

Vamos a hacer uso del paquete Sequelize CLI para iniciar el proyecto por nosotros. También nos ayudará a generar migraciones de bases de datos.

Comencemos instalando el paquete Sequelize CLI.

```
npm install -g sequelize-cli
```

Pueden instalar el paquete **sequelize-cli** en su proyecto localmente usando -D



[Open in app](#)[Get started](#)

dependencias.

Recordemos que en este tutorial, vamos a estar interactuando con MySQL únicamente.

```
npm install --save sequelize  
npm install --save mysql2
```

## Inicializando Sequelize

Después de la instalación, utilice CLI para generar migraciones, seeders, configuración, modelos y el archivo de configuración.

*En la terminal vamos a ejecutar el siguiente comando:*

```
sequelize init
```

Si inspecciona tu directorio en este momento, se darán cuenta de que el comando anterior acaba de crear los directorios y generó el código repetitivo. Su estructura de directorio ahora debería verse así:

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

Consideremos, por ejemplo, el archivo **server / models / index.js** que se generó automáticamente.



[Open in app](#)[Get started](#)

```
4  const path = require('path');
5  const Sequelize = require('sequelize');
6  const basename = path.basename(__filename);
7  const env = process.env.NODE_ENV || 'development';
8  const config = require(__dirname + '/../config/config.json')[env];
9  const db = {};
10
11 let sequelize;
12 if (config.use_env_variable) {
13   sequelize = new Sequelize(process.env[config.use_env_variable], config);
14 } else {
15   sequelize = new Sequelize(config.database, config.username, config.password, config);
16 }
17
18 fs
19 .readdirSync(__dirname)
20 .filter(file => {
21   return (file.indexOf('.') !== 0) && (file !== basename) && (file.slice(-3) === '.js');
22 })
23 .forEach(file => {
24   const model = sequelize['import'](path.join(__dirname, file));
25   db[model.name] = model;
26 });
27
28 Object.keys(db).forEach(modelName => {
29   if (db[modelName].associate) {
30     db[modelName].associate(db);
31   }
32 });
33
34 db.sequelize = sequelize;
35 db.Sequelize = Sequelize;
36
37 module.exports = db;
```

## Configuración de Base de Datos

Ahora vamos a configurar la base de datos nuestra en el siguiente archivo **config.json** que se encuentra en **config / config.json**.





Open in app

Get started

```
4      "password": "123456",
5      "database": "ejemplo-sequelize",
6      "host": "127.0.0.1",
7      "dialect": "mysql",
8      "operatorsAliases": false
9    },
10   "test": {
11     "username": "root",
12     "password": "123456",
13     "database": "ejemplo-sequelize",
14     "host": "127.0.0.1",
15     "dialect": "mysql",
16     "operatorsAliases": false
17   },
18   "production": {
19     "username": "root",
20     "password": "123456",
21     "database": "ejemplo-sequelize",
22     "host": "127.0.0.1",
23     "dialect": "mysql",
24     "operatorsAliases": false
25   }
26 }
```

Contenido ejemplificativo del config.json

```
{
  "username": "root",
  "password": "123456",
  "database": "ejemplosequelize",
  "host": "127.0.0.1",
  "dialect": "mysql"
}
```

**Importante validen que tengan la base de datos creada en su servidor o computadora.**

Si no tienen una aplicación para conectarse a una base de datos relacional, les recomiendo descargar el [MySQL Workbench](#).

Para crear una base de datos, ejecuten el siguiente comando:



[Open in app](#)[Get started](#)

## Generando Modelos y Migraciones

Ahora, de nuevo, usaremos el comando **sequelize cli** para generar archivos de modelo y migraciones.

Vamos a tener tres modelos: Usuario, Juego, Participación.

La relación entre una Usuario y Juego estará vista en Participación, que será el momento que un jugador juega un juego.

```
sequelize model:create --name usuario --attributes  
username:string,status:char
```

Como podrán observar, estamos creando el **modelo Usuario** el cual tiene atributos definidos: id, username, status, date.

Como resultado deberíamos tener en **models** el archivo **usuario.js**

mi-proyecto/models/usuario.js

```
'use strict';

module.exports = (sequelize, DataTypes) => {
  const Usuario = sequelize.define('usuario', {
    id: DataTypes.INTEGER,
    username: DataTypes.STRING,
    status: DataTypes.CHAR
  }, {});
  Usuario.associate = function(models) {
    // associations can be defined here
  };

  return Usuario;
};
```



[Open in app](#)[Get started](#)

- allowNull
- autoIncrement
- primaryKey
- type

A su vez, las **tablas** pueden incluir:

- timestamp: ingresa la fecha de registro
- tableName: nombre de la tabla

Por último, viene la parte interesante de las base de datos relacionadas, donde se pueden asociar los campos de sus tablas para que todo esté como uno quiere 😊.

*Vamos a ver como queda **usuario.js** una vez que le aplicamos los cambios:*

```
'use strict';

module.exports = (sequelize, DataTypes) => {
  const Usuario = sequelize.define('usuario', {
    id: {
      allowNull: false,
      autoIncrement: true,
      primaryKey: true,
      type: DataTypes.INTEGER
    },
    username: {
      allowNull: false,
      type: DataTypes.STRING
    },
    status: {
      allowNull: true,
      defaultValue: 1,
      type: DataTypes.CHAR
    }
  }, {
    timestamps: false,
    freezeTableName: true,
    tableName: 'usuario',
    classMethods: {}
  })
}
```



[Open in app](#)[Get started](#)

```

};

return Usuario;

};

```

Luego podrán observar también que se ha generado un nuevo archivo en **migrations**:

mi-proyecto/migrations/<date>-create-usuario.js

```

'use strict';

module.exports = {

  up: (queryInterface, Sequelize) => {

    return queryInterface.createTable('usuario', {
      id: {
        allowNull: false,
        autoIncrement: true,
        primaryKey: true,
        type: Sequelize.INTEGER
      },
      username: {
        type: Sequelize.STRING
      },
      status: {
        type: Sequelize.CHAR
      }
    });
  },

  down: (queryInterface, Sequelize) => {
    return queryInterface.dropTable('usuario');
  }
};

```

A continuación vamos a crear el modelo y migraciones de Juego:

```
sequelize model:create --name juego --attributes
  name:string description:string status:char
```



[Open in app](#)[Get started](#)

```
sequelize model:create --name participacion --attributes
jugador_id:integer,juego_id:integer,status:char
```

Cómo nuestra relación entre Usuario y Juego esta representada en Participación, vamos a tener que asociar la tabla a cada uno de los campos respectivamente.

Vamos a ver cómo quedaría esta asociación (**associate**):

```
participacion.associate = function(models) {
  // associations can be defined here
  participacion.belongsTo(models.usuario,
  {
    as: 'usuario',
    foreignKey: 'usuario_id',
  })
  participacion.belongsTo(models.juego,
  {
    as: 'juego',
    foreignKey: 'juego_id',
  })
};
```

Como pueden observar, estamos vinculando Participación al modelo usuario y juego.

*Ustedes ya tienen generado los modelos y migraciones para este ejemplo, si tienen otro tipo de proyecto recuerden que van a tener que hacer un comando para cada una de las “tablas” que ustedes posean.*

Si quieren más información sobre cómo crear las migraciones, tipo de dato, etc les recomiendo que ingresen a la página de [Sequelize / Migrations](#).

Si quieren referenciar **foreingKey** en las **migrations**, lo que tienen que hacer es incluir **references{}**:



[Open in app](#)[Get started](#)

```

    allowNull: true,
    references: {
      model: 'juego',
      key: 'id'
    },
}

```

Cuando ejecutamos estas migraciones, se ejecutará la función `up`. La misma, se encargará de **crear la tabla y sus columnas asociadas para nosotros** (magia).

*Si, por alguna razón, tuviéramos que revertir (deshacer) la migración, la función `down` se ejecuta y deshace lo que hiciera la función `up`, devolviendo así nuestra base de datos al mismo estado en el que estaba antes de realizar la migración.*

Estas migraciones son una representación de cómo queremos que se vean nuestros modelos en la base de datos.

Sequelize genera automáticamente los campos **id**, **createdAt** y **updatedAt**. Además de eso, cada vez que se guarda un modelo, el campo `updatedAt` se actualiza automáticamente para reflejar el nuevo tiempo de actualización.

Con los modelos y las migraciones en su lugar, ahora estamos listos para conservar los modelos en la base de datos ejecutando las migraciones. Para hacer esto, ejecutamos el siguiente comando:

```
sequelize db:migrate
```

```

MacBookTomas:ejemplo-sequelize tomasmalio$ sequelize db:migrate
Sequelize CLI [Node: 10.15.3, CLI: 5.5.1, ORM: 5.21.10]
Loaded configuration file "config/config.json".
Using environment "development".
(node:28170) [SEQUELIZE0004] DeprecationWarning: A boolean value was passed to options.operatorsAliases. This is a no-op with v5 and should be removed.
== 20200522161338-create-usuario: migrating =====
== 20200522161338-create-usuario: migrated (0.073s)

== 20200522172351-create-participacion: migrating =====
== 20200522172351-create-participacion: migrated (0.026s)

== 20200522172407-create-juego: migrating =====
== 20200522172407-create-juego: migrated (0.025s)

```



[Open in app](#)[Get started](#)

migraciones actuales 😊.

## Creando Controladores y Enrutamiento (Routing)

Vamos a crear los controladores para cada uno de los modelos que generamos anteriormente.

### Creando Usuario Controller

Creamos un archivo que se llame `usuario.js` dentro de la carpeta `controllers/`.

Dentro de ese archivo `usuario.js` será responsable de crear, enumerar, actualizar y eliminar usuarios.

Vamos a crear el código del controlador.

```
const Sequelize      = require('sequelize');
const usuario       = require('../models').usuario;

module.exports = {

  create(req, res) {
    return usuario
      .create ({
        username: req.params.username,
        status: req.params.status
      })
      .then(usuario => res.status(200).send(usuario))
      .catch(error => res.status(400).send(error))
  },

  list(_, res) {
    return usuario.findAll({})
      .then(usuario => res.status(200).send(usuario))
      .catch(error => res.status(400).send(error))
  },

  find (req, res) {
    return usuario.findAll({
      where: {
        username: req.params.username,
      }
    })
    .then(usuario => res.status(200).send(usuario))
    .catch(error => res.status(400).send(error))
  }
},
```



[Open in app](#)[Get started](#)

Cómo podemos observar, estamos utilizando **3 acciones** dentro de lo que es **usuario.js**. En el controlador, hemos creado:

- **create(req, res)**
- **list(\_ , res)**
- **find(req, res)**

Vamos a entender qué significa ese paréntesis. Cuando uno utiliza **req** (requerimientos) y **res** (resultados). Existe la posibilidad que no quieran recibir nada, entonces simplemente pueden utilizar el guión bajo (\_).

Ahora que tenemos listo el controlador, vamos a generar el enrutamiento para que podamos acceder a nuestro **servicio**.

Vamos a generar un archivo en **routes/index.js** y dentro debemos incluir la siguiente información:

```
/* Controllers */
const usuarioController = require('../controllers/usuario');

module.exports = (app) => {
    app.get('/api', (req, res) => res.status(200).send ({
        message: 'Example project did not give you access to the api
web services',
    });

    app.post('/api/usuario/create/username/:username/status/:status',
    usuarioController.create);
    app.get('/api/usuario/list', usuarioController.list);
    app.get('/api/usuario/find/username/:username',
    usuarioController.find);

};
```

Ahora que tienen generado el **routes/index.js**, tiene que incluir una línea justo antes de



[Open in app](#)[Get started](#)

Quedando así:

```
const express      = require('express');
const logger       = require('morgan');
const bodyParser   = require('body-parser');

// This will be our application entry. We'll setup our server here.
const http = require('http');

// Set up the express app
const app = express();

// Log requests to the console.
app.use(logger('dev'));

// Parse incoming requests data (https://github.com/expressjs/body-
parser)
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended: false }));

// Setup a default catch-all route that sends back a welcome message
in JSON format.
require('./routes')(app);
app.get('*', (req, res) => res.status(200).send({
  message: 'Welcome to the beginning of nothingness.',
}));

const port = parseInt(process.env.PORT, 10) || 8000;
app.set('port', port);

const server = http.createServer(app);
server.listen(port);

module.exports = app;
```

Hemos generado **3 nuevos servicios para Usuario:**

**Create:**

**POST:** `http://localhost:8000/api/usuario/create/username/paula/status/1`

**Resultado:**



[Open in app](#)[Get started](#)

## List:

**GET:** <http://localhost:8000/api/usuario/list>

## Resultado:

```
[
  {
    "id": 1,
    "username": "paula",
    "status": "1"
  },
  {
    "id": 2,
    "username": "tomas",
    "status": "1"
  }
]
```

## Find:

**GET:** <http://localhost:8000/api/usuario/find/username/paula>

## Resultado:

```
[
  {
    "id": 1,
    "username": "paula",
    "status": "1"
  }
]
```

## Creando Juego Controller

Es similar al del usuario, simplemente cambia que estamos utilizando como modelo y ciertos parámetros dentro de las consultas de SQL.

```
const Sequelize = require('sequelize');
const juego = require('../models').juego;
```



[Open in app](#)[Get started](#)

```

        .create ({
            name: req.body.name,
            description: req.body.description,
            status: req.body.status
        })
        .then(juego => res.status(200).send(juego))
        .catch(error => res.status(400).send(error))
    },
    list(_, res) {
        return juego.findAll({})
        .then(juego => res.status(200).send(juego))
        .catch(error => res.status(400).send(error))
    },
    find (req, res) {
        return juego.findAll({
            where: {
                name: req.body.name,
            }
        })
        .then(juego => res.status(200).send(juego))
        .catch(error => res.status(400).send(error))
    },
};


```

Como podemos observar, estamos recibiendo los parámetros en formato body.

Una vez que tenemos creado el Controlador, debemos declararlo dentro de un archivo **index.js** dentro de la carpeta **Controller**

```

const juegoController = require('./juego');

module.export = {
    juegoController
}

```

Vamos a incluir en nuestro archivo **routes/index.js** la siguiente información:



[Open in app](#)[Get started](#)

- Después de los últimos servicios que hemos incluido:

```
app.post('/api/juego/create', juegoController.create);
app.get('/api/juego/list', juegoController.list);
app.get('/api/juego/find', juegoController.find);
```

Hemos generado **3 nuevos servicios** para Juego:

### Create:

**POST:** <http://localhost:8000/api/juego/create>

### Body:

```
{
  "name": "Lengua Avanzada",
  "description": "Aprendé sobre la lengua española",
  "status": 1
}
```

### Resultado:

```
{
  "createdAt": 1590175915006,
  "updatedAt": 1590175915006,
  "id": 2,
  "name": "Lengua Avanzada",
  "description": "Aprendé sobre la lengua española",
  "status": 1
}
```

### List:

**GET:** <http://localhost:8000/api/juego/list>

### Resultado:



[Open in app](#)[Get started](#)

```

    "createdAt": "2020-05-22T19:05:06.000Z",
    "updatedAt": "2020-05-22T19:05:06.000Z"
}
]
```

**Find:****GET:** <http://localhost:8000/api/juego/find>**Body:**

```
{
  "name": "Lengua Avanzada"
}
```

**Resultado:**

```

[
  {
    "id": 2,
    "name": "Lengua Avanzada",
    "description": "Aprendé sobre la lengua española",
    "status": "1",
    "createdAt": "2020-05-22T19:31:55.000Z",
    "updatedAt": "2020-05-22T19:31:55.000Z"
  }
]
```

**Creando Participación Controller**

Llego el momento de crear el controlador que esta asociado a Usuario y Juego.

```

const Sequelize      = require('sequelize');
const Op            = Sequelize.Op;
const participacion = require('../models').participacion;
const usuario       = require('../models').usuario;
const juego         = require('../models').juego;

module.exports = {

```



[Open in app](#)[Get started](#)

```
username: req.body.usuario
}, {
    id: req.body.usuario
}]
}
});

// Juego
const responseJuego = juego.findOne({
    where: {
        [Op.or]: [
            name: req.body.juego
        , {
            id: req.body.juego
        }
    ]
}
});

Promise
.all ([responseUsuario, responseJuego])
.then(responses => {
    return participacion
        .create ({
            usuario_id: responses[0].id,
            juego_id: responses[1].id,
            status: req.body.status,
        })
        .then(participacion =>
res.status(200).send(participacion))
})
.catch(error => res.status(400).send(error));
}

list(_, res) {
    return participacion.findAll({
        include: [
            {
                model: usuario,
                as: 'usuario'
            , {
                model: juego,
                as: 'juego'
            }
        ]
})
.then(participacion => res.status(200).send(participacion))
.catch(error => res.status(400).send(error))
}

find (req, res) {
    return participacion.findAll(
```



[Open in app](#)[Get started](#)

```
        },
        model: juego,
        as: 'juego'
    ]}
})
.then(participacion => res.status(200).send(participacion))
.catch(error => res.status(400).send(error))
},
};

};
```

Vamos a incluir en nuestro archivo **routes/index.js** la siguiente información:

- Despues del controlador de juego:

```
const participacionController = require('../controllers/participacion');
```

- Despues de los últimos servicios que hemos incluido:

```
app.post('/api/participacion/create',
participacionController.create);
app.get('/api/participacion/list', participacionController.list);
app.get('/api/participacion/find', participacionController.find);
```

Hemos generado **3 nuevos servicios para Participación:**

```
{
    "usuario": "tomas",
    "juego": "1",
    "status": 1
}
```

**Resultado:**



[Open in app](#)[Get started](#)

```
        "status": 1  
    }
```

## List:

**GET:** <http://localhost:8000/api/juego/list>

### Resultado:

```
[  
  {  
    "id": 1,  
    "usuario_id": 1,  
    "juego_id": 1,  
    "status": "1",  
    "createdAt": "2020-05-22T19:14:21.000Z",  
    "updatedAt": "2020-05-22T19:14:21.000Z",  
    "usuario": {  
      "id": 1,  
      "username": "paula",  
      "status": "1"  
    },  
    "juego": {  
      "id": 1,  
      "name": "Lengua Avanzada",  
      "description": "Aprendé sobre la lengua española",  
      "status": "1",  
      "createdAt": "2020-05-22T19:05:06.000Z",  
      "updatedAt": "2020-05-22T19:05:06.000Z"  
    }  
  },  
  {  
    "id": 2,  
    "usuario_id": 2,  
    "juego_id": 1,  
    "status": "1",  
    "createdAt": "2020-05-22T19:23:29.000Z",  
    "updatedAt": "2020-05-22T19:23:29.000Z",  
    "usuario": {  
      "id": 1,  
      "username": "tomas",  
      "status": "1"  
    },  
    "juego": {  
      "id": 1,  
      "name": "Matemática",  
      "description": "Aprendé Matemática",  
      "status": "1",  
      "createdAt": "2020-05-22T19:23:29.000Z",  
      "updatedAt": "2020-05-22T19:23:29.000Z"  
    }  
  }]
```



[Open in app](#)[Get started](#)**Find:****GET: <http://localhost:8000/api/participacion/find/id/:id>****Resultado:**

```
[
  {
    "id": 1,
    "usuario_id": 2,
    "juego_id": 1,
    "status": "1",
    "createdAt": "2020-05-22T19:23:29.000Z",
    "updatedAt": "2020-05-22T19:23:29.000Z",
    "usuario": {
      "id": 1,
      "username": "paula",
      "status": "1"
    },
    "juego": {
      "id": 1,
      "name": "Matemática",
      "description": "Aprendé Matemática",
      "status": "1",
      "createdAt": "2020-05-22T19:05:06.000Z",
      "updatedAt": "2020-05-22T19:05:06.000Z"
    }
  }
]
```

**¡Listo hemos terminado de construir nuestros servicios y ya podemos consultarlos!**

Para que tengan como referencia, nuestro directorio debería quedar así:

```
mi-proyecto/
├── node_modules/
│   └── config.json
└── controllers/
```

52 2



[Open in app](#)[Get started](#)

```
└── index.js
    ├── juego.js
    ├── participacion.js
    └── usuario.js
    └── node_modules/
        ├── Helpster/
        │   ├── index.js
        ├── seeders/
        ├── app.js
        └── package-lock.json
    └── package.json
```



Espero que les sirva este tutorial rápido para implementar Sequelize junto con Node.js.

Si desean visualizar un proyecto funcional, pueden acceder a un repositorio en [GitLab aquí](#).

