



Sequelize

THE BRIDGE
DIGITAL TALENT **ACCELERATOR**

Índice

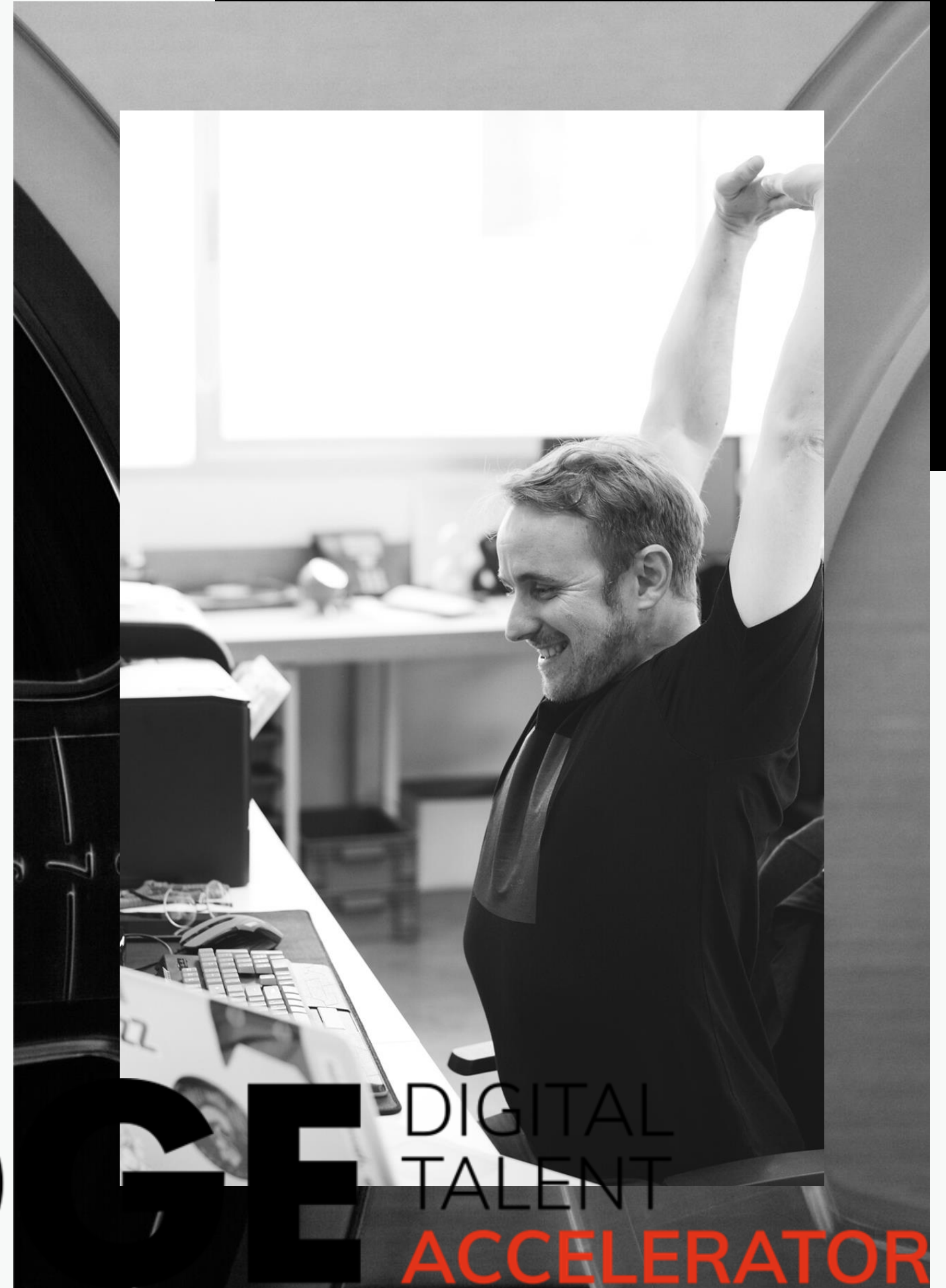
ORM

Sequelize

Queries con Sequelize

Seeders

THE  **BRIDGE** **DIGITAL
TALENT
ACCELERATOR**



ORM

Un **ORM** es un **modelo de programación** que permite **mapear** las estructuras de una base de datos relacional (SQL Server, Oracle, MySQL, etc.), en adelante **RDBMS** (Relational Database Management System), sobre una **estructura lógica de entidades** con el objeto de **simplificar** y **acelerar** el desarrollo de nuestras aplicaciones.

Cuando programamos una **aplicación** con **conexión** a una **base de datos**, habrás podido comprobar lo laborioso que es **transformar** toda la **información** que recibes de la **base datos**, principalmente en tablas, en los objetos de tu aplicación y viceversa. A ésto se le denomina **mapeo**. Utilizando un ORM este mapeo será **automático**.

Además de **mapear**, los **ORMs** nos **libera** de la escritura o generación manual de **código SQL** necesario para realizar las queries o consultas y gestionar la persistencia de datos en el **RDBMS**.

Sequelize

Sequelize es un **ORM O (Object) R (Relational) M (Mapping)** que **permite** a los usuarios llamar a **funciones javascript** para **interactuar con SQL DB** sin escribir consultas reales. Es bastante útil para acelerar el tiempo de desarrollo.



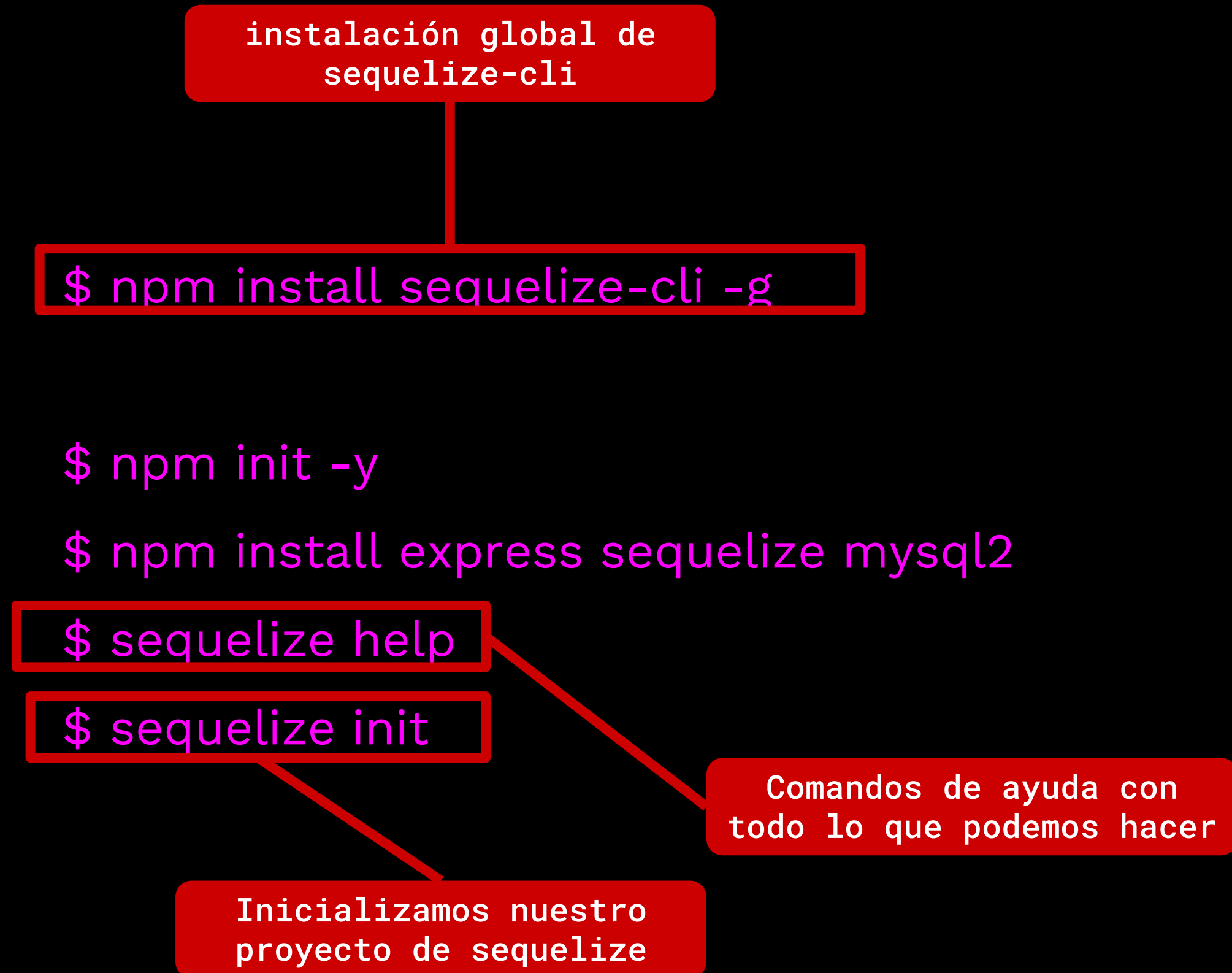
“

Creando un proyecto en Sequelize

Proyecto en Sequelize

Primero **instalamos** el CLI de Sequelize de forma global (solo se hace **una vez** en tu PC)

A continuación nos situaremos sobre el directorio en el que queremos crear el nuevo proyecto y escribiremos los comandos (por cada proyecto que creemos):



Estructura carpetas

Tendremos la siguiente estructura de carpetas:

- Carpeta **config** con un archivo **config.json** con la configuración de la **conexión a la db** en los diferentes entornos (development, test y production, aunque podemos añadir más)
- Carpeta **models** donde tendremos un **index.js** que creará la conexión a partir de los datos de config y el entorno en el que nos encontremos. Además mapeará los modelos que se encuentren en la carpeta models y los añadirá como propiedades del objeto db que finalmente exporta.
- Las carpetas **migrations** y **seeders** se encuentran **vacías**, y se llenarán cuando creemos modelos/migraciones y seeders.

```
✓ config
  {} config.json
  > migrations
✓ models
  JS index.js
  > node_modules
  > seeders
  {} package-lock.json
  {} package.json
```


Levantando la base de datos

Para crear una **base de datos**, escribiremos el **nombre** en el **config.json** en la propiedad **database**. Una vez hecho, ejecutamos el siguiente comando:

```
{  
  "development": {  
    "username": "root",  
    "password": null,  
    "database": "database_development",  
    "host": "127.0.0.1",  
    "dialect": "mysql"  
  },  
  ...  
}
```

...

```
$ sequelize db:create
```

creamos la
base de datos

Creando un Modelo (y migración)

Para crear un modelo (y migración) utilizaremos también el CLI. Definiremos el **nombre**, los **atributos** y sus **tipos de datos**. Los **atributos deben ir pegados sin espacios** después de las comas, los tipos de datos disponibles se encuentran en la [documentación](#).

Al darle enter nos creará tanto el **modelo** (que usaremos para trabajar con la BD) como la **migración** (que al ejecutarla creará la tabla con estos campos).

```
$ sequelize model:generate --name User --attributes  
name:string,email:string,password:string,role:string
```

Modelo

En el archivo generado en **models** veremos el modelo de User creado.

Por convención el modelo ha de **empezar por letra mayúscula y en singular.**

El método estático **associate** contendrá las relaciones que tendrá el modelo con otros modelos.

```
'use strict';
const { Model } = require('sequelize');
module.exports = (sequelize, DataTypes) => {
  class User extends Model {
    static associate(models) {
    }
  }
  User.init({
    name: DataTypes.STRING,
    email: DataTypes.STRING,
    password: DataTypes.STRING,
    role: DataTypes.STRING
  }, {
    sequelize,
    modelName: 'User',
  });
  return User;
};
```

Migraciones

El CLI de sequelize genera también un archivo js en la carpeta **migrations**. Este archivo contiene los **campos** del modelo más el **id** (con auto-increment y no nullable) y los **timestamps** (createdAt & updatedAt)

```
'use strict';
module.exports = {
  async up(queryInterface, Sequelize) {
    await queryInterface.createTable('Users', {
      id: {
        allowNull: false,
        autoIncrement: true,
        primaryKey: true,
        type: Sequelize.INTEGER
      },
      name: { type: Sequelize.STRING },
      email: { type: Sequelize.STRING },
      password: { type: Sequelize.STRING },
      role: { type: Sequelize.STRING },
      createdAt: { allowNull: false, type: Sequelize.DATE },
      updatedAt: { allowNull: false, type: Sequelize.DATE }
    });
  },
  async down(queryInterface, Sequelize) {
    await queryInterface.dropTable('Users');
  }
};
```

Levantando la tabla users

Para crear la tabla users en la base de datos, debemos ejecutar el siguiente comando:

```
$ sequelize db:migrate
```

Deshacer migraciones

Para deshacer la última migración o todas las migraciones ejecutamos los siguientes comandos:

```
$ sequelize db:migrate:undo
```

deshacemos la última migración

```
$ sequelize db:migrate:undo:all
```

deshacemos todas las migraciones

```
$ sequelize db:migrate:undo --name nombre_migración
```

deshacemos la migración que en concreto que queramos

Controlador User

Creamos nuestro **UserController.js** en la carpeta **controllers**. Nos importamos el **modelo User** y creamos un objeto **UserController** que contendrá los diferentes métodos que vamos a implementar, en este caso la creación de un usuario:

```
const { User } = require('../models/index.js');
```

importamos nuestro
modelo User

```
const UserController = {  
  create(req, res) {  
    req.body.role = "user";
```

```
    User.create(req.body)
```

usamos el método create
de sequelize que nos hace
un insert del usuario

```
    .then(user => res.status(201).send({ message: 'Usuario creado con éxito', user }))  
    .catch(console.error)
```

```
  },
```

```
}
```

```
module.exports = UserController
```

Ruta user

Creamos nuestro archivo **users.js** en la carpeta **routes** con el siguiente código y nos importamos el archivo **UserController.js** que estará en la carpeta **controllers** que vamos a crear:

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

```
const router = express.Router();
```

```
const UserController = require('../controllers/UserController')
```

importamos el
UserController

```
router.post('/', UserController.create)
```

```
module.exports = router;
```


Creación de index.js

Creamos como de costumbre nuestro archivo **index.js** con el siguiente código y nos importamos la carpeta **routes** que contendrá el archivo **users.js** que vamos a crear:

```
const express = require('express');  
const app = express();  
const PORT = 3000
```

```
app.use(express.json())
```

```
app.use('/users', require('./routes/users'));
```

importamos nuestras rutas
de users

```
app.listen(PORT, () => console.log('Servidor levantado en el puerto ' + PORT))
```



Relaciones en sequelize

Creando un Modelo (y migración) Posts

Ahora vamos a crear nuestro modelo y migración de nuestra nueva tabla posts:

```
$ sequelize model:generate --name Post --attributes  
title:string,content:string,UserId:integer
```

Modelo Post

Hemos generado un nuevo **modelo Post**. En este caso vamos a definir en el método estático **associate** la **relación** de los posts con los usuarios:

El modo en que se relacionan las tablas es con el **nombre del modelo** (primera en mayúscula), y **Id**. En este caso, la foreign key de User es **UserId**.

[*Documentación relaciones en sequelize](#)

```
'use strict'

const { Model } = require('sequelize')

module.exports = (sequelize, DataTypes) => {

  class Post extends Model {

    static associate(models) {

      Post.belongsTo(models.User)

    }

  }

  Post.init(

    {

      title: DataTypes.STRING,

      content: DataTypes.STRING,

      UserId: DataTypes.INTEGER,

    },

    {

      sequelize,

      modelName: 'Post',

    }

  )

  return Post

}
```

definimos
nuestra
relación

Modelo User

Un **usuario** tiene muchos **posts** por lo que en este caso en vez de poner **belongsTo**, usaremos el **hasMany**:

```
"use strict";
const { Model } = require("sequelize");
module.exports = (sequelize, DataTypes) => {
  class User extends Model {
    static associate(models) {
      User.hasMany(models.Post);
    }
  }
  ...
}
```

definimos la relación que tiene un usuario con sus pedidos

Levantando la tabla posts

Para crear la tabla posts en la base de datos que acabamos de crear debemos ejecutar el siguiente comando:

```
$ sequelize db:migrate
```

Controlador Post

Creamos nuestro **PostController.js** donde nos importamos el **modelo Post**. Creamos un objeto **PostController** que contendrá los diferentes métodos que vamos a implementar, en este caso la **creación** de un post:

```
const { Post } = require('../models/index.js');

const PostController = {
  create(req, res) {
    Post.create(req.body)
      .then(post => res.status(201).send({ message: 'Publicación creada con éxito', post }))
      .catch(console.error)
  },
}

module.exports = PostController
```


Ruta Post

Creamos nuestro archivo **posts.js** en la carpeta **routes** con el siguiente código y nos importamos el archivo **PostController.js** que estará en la carpeta **controllers**:

```
const express = require('express');  
const router = express.Router();  
const PostController = require('../controllers/PostController')  
  
router.post('/', PostController.create)  
  
module.exports = router;
```

Index.js

Importamos el nuevo archivo de **rutas Posts** en nuestro **index.js**

```
const express = require('express');  
const app = express();  
const PORT = 3000
```

```
app.use(express.json())
```

```
app.use('/users', require('./routes/users'));  
app.use('/posts', require('./routes/posts'));
```

importamos nuestras rutas
de nuestro archivo posts

```
app.listen(PORT, () => console.log('servidor levantado en el puerto' + PORT))
```

Controlador Post

Ahora crearemos un nuevo método en **PostController** que nos traerá **todos los posts** juntos al **usuario** propietario dicho post:

```
const { Post, User } = require('../models/index.js');
```

importamos además del
modelo Post el modelo
User

```
const PostController = {
```

```
...
```

```
  getAll(req, res) {
```

```
    Post.findAll({
```

```
      include: [User]
```

```
    })
```

```
    .then(posts => res.send(posts))
```

```
    .catch(err => {
```

```
      console.log(err)
```

```
      res.status(500).send({ message: 'Ha habido un problema al cargar las publicaciones' })
```

```
    })
```

```
  },
```

```
}
```

con él include le decimos
que nos traiga el User
que hizo el post

```
module.exports = PostController
```

Rutas Posts

Añadimos la **nueva ruta** en nuestro archivo **posts.js** que nos traerá **todos los posts** junto al usuario propietario del post:

```
const express = require('express');
const router = express.Router();
const PostController = require('../controllers/PostController')

router.post('/', PostController.create)
router.get('/', PostController.getAll)

module.exports = router;
```

Controlador User

Podemos hacer lo mismo con el **User**. Crearemos un nuevo método que nos traerá **todos los users** junto a los **posts** que tiene:

```
const { User, Post } = require('../models/index.js');
```

importamos además del
modelo Post el modelo
User

```
const UserController = {
```

```
  . . .
```

```
  getAll(req, res) {
```

```
    User.findAll({
```

```
      include: [Post]
```

```
    })
```

```
    .then(users => res.send(users))
```

```
    .catch(err => {
```

```
      console.log(err)
```

```
      res.status(500).send({ message: 'Ha habido un problema al cargar las publicaciones' })
```

```
    })
```

```
  },
```

```
}
```

con él includes le
decimos que nos traiga el
User con todos sus posts

```
module.exports = UserController
```

Rutas Users

Añadimos la nueva ruta en nuestro archivo **users.js** que nos traerá **todos** los **users** junto a **sus posts**:

```
const express = require('express');
const router = express.Router();
const UserController = require('../controllers/UserController')

router.post('/', UserController.create)
router.get('/', UserController.getAll)

module.exports = router;
```

Controlador Post por id

Ahora crearemos un nuevo método en **PostController.js** que nos traerá el **post por el id** que le pasemos junto al **usuario** que hizo el post:

```
const { Post, User } = require('../models/index.js');

const PostController = {
  ...
  getById(req, res) {
    Post.findBvPk(req.params.id,
      include: [{ model: User, attributes: ['name'] }])
      .then(post => res.send(post))
  },
}

module.exports = PostController
```

con el `findByPk`, nos buscará el post por su Primary Key y nos devolverá el mismo

Rutas Posts

Añadimos la nueva ruta en nuestro archivo **posts.js** que nos traerá el **post por id**:

```
const express = require('express');
const router = express.Router();
const PostController = require('../controllers/PostController')

router.post('/', PostController.create)
router.get('/', PostController.getAll)
router.get('/id/:id', PostController.getById)

module.exports = router;
```

findOne

El método **findOne** obtiene la primera entrada que encuentra (que cumple con las opciones de consulta, si se proporcionan).

```
const { Post, User, Sequelize } = require('../models/index.js')
```

```
const { Op } = Sequelize
```

```
const PostController = {
```

```
  . . .
```

```
  getOneByName(req, res) {
```

```
    Post.findOne({
```

```
      where: {
```

```
        title: {
```

```
          [Op.like]: `>${req.params.title}<`,
```

```
        },
```

```
      },
```

```
      include: [User],
```

```
    }).then((post) => res.send(post))
```

```
  },
```

```
}
```

Aquí definimos las
diferentes
opciones/condiciones

Rutas Posts

Añadimos la nueva ruta en nuestro archivo **posts.js** que nos traerá el **post por nombre**:

```
const express = require('express');
const router = express.Router();
const PostController = require('../controllers/PostController')

router.post('/', PostController.create)
router.get('/', PostController.getAll)
router.get('/:id', PostController.getById)
router.get('/title/:title', PostController.getOneByName)

module.exports = router;
```

destroy

El método **destroy** más el **where** elimina el Post que cumpla la condición.

```
const PostController = {  
  ...  
  async delete(req, res) {  
    await Post.destroy({  
      where: {  
        id: req.params.id  
      }  
    })  
    res.send(  
      'La publicación ha sido eliminada con éxito'  
    )  
  },  
}
```



```
module.exports = PostController
```

Aquí le decimos que nos elimine el Post que su id coincida con el id que le pasamos por parámetro

Rutas Posts

Añadimos la nueva ruta en nuestro archivo **posts.js** que **eliminará** el post por id:

```
const express = require('express');
const router = express.Router();
const PostController = require('../controllers/PostController')

router.post('/', PostController.create)
router.get('/', PostController.getAll)
router.get('/:id', PostController.getById)
router.get('/title/:title', PostController.getOneByName)
router.delete('/id/:id', PostController.delete)

module.exports = router;
```

Eliminar un usuario y sus posts

Con el método **destroy** también podemos **eliminar** un usuario con los posts que tiene

```
const { User, Post } = require('../models/index.js');

const UserController = {
  . . .
  async delete(req, res) {
    await User.destroy({
      where: {
        id: req.params.id
      }
    })
    await Post.destroy({
      where: {
        UserId: req.params.id
      }
    })
    res.send(
      'El usuario ha sido eliminado con éxito'
    )
  },
}

module.exports = UserController
```

le decimos que también borre los posts que tengan de UserId el que le hemos pasado

Rutas Users

Añadimos la nueva ruta en nuestro archivo **users.js** que eliminará al **usuario** junto a sus **posts**:

```
const express = require('express');
const router = express.Router();
const UserController = require('../controllers/UserController')

router.post('/', UserController.create)
router.get('/', UserController.getAll)
router.delete('/id/:id', UserController.delete)

module.exports = router;
```


Update

Con el método **update** también podemos actualizar un **User**

```
const { User, Post } = require('../models/index.js');

const UserController = {
  . . .
  async update(req, res) {
    await User.update(
      { name: req.body.name, email: req.body.email },
      { where: { id: req.params.id } })
    res.send('Usuario actualizado con éxito');
  }
}

module.exports = UserController
```

Rutas Users

Añadimos la nueva ruta en nuestro archivo **users.js** que **actualizará** al usuario:

```
const express = require('express');
const router = express.Router();
const UserController = require('../controllers/UserController')

router.post('/', UserController.create)
router.get('/', UserController.getAll)
router.delete('/:id/:id', UserController.delete)
router.put('/:id/:id', UserController.update)

module.exports = router;
```

“

Seeders

Creando un Seeder

Supongamos que queremos insertar algunos datos en algunas tablas de forma predeterminada.

Los **archivos seeders** son algunos cambios en los datos que se pueden usar para llenar las tablas con **datos de muestra o de prueba**.

```
$ sequelize seed:generate --name demo-user
```

Creando un seeder

En el **archivo seeder** que hemos generado, insertaremos un **usuario de prueba** a nuestra tabla de usuarios.

```
'use strict'

/** @type {import('sequelize-cli').Migration} */
module.exports = {
  async up(queryInterface, Sequelize) {
    return queryInterface.bulkInsert('Users', [
      {
        name: 'John',
        email: 'example@example.com',
        password: '123456',
        role: 'user',
        createdAt: new Date(),
        updatedAt: new Date(),
      },
    ])
  },

  async down(queryInterface, Sequelize) {
  },
}
```

Ejecutando el Seeder

En el paso anterior hemos creado un archivo seeder. Sin embargo, no se ha añadido a la base de datos. Para hacer eso, lanzamos un comando que ejecutará ese archivo inicial y se insertará un usuario de demostración en la tabla users:

```
$ sequelize db:seed:all
```