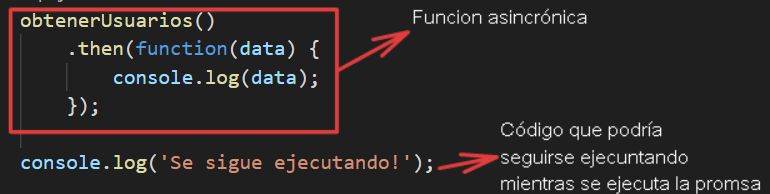
Introducción, modelos, y consultas

Promesas

Las promesas son funciones que permiten ejecutar **código asincrónico** de forma eficiente.

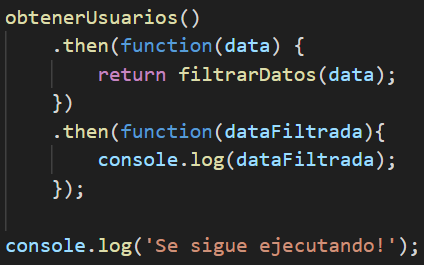
Un **pedido asincrónico** es un conjunto de instrucciones que se ejecutan mediante un mecanismo específico, como puede ser un **callback**, una **promesa**, o un **evento**. Esto hace posible que la respuesta sea procesada en otro momento.

Este comportamiento es **no bloqueante**, ya que el pedido se ejecuta en paralelo con el resto del código.



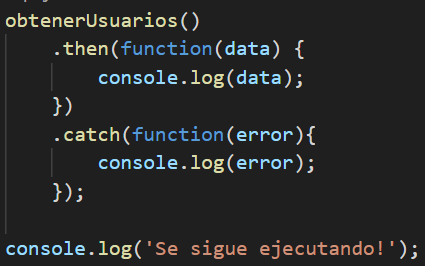
En este caso, el **console.log(data)** se ejecutará SOLO SI obtenerUsuarios() devuelve un resultado. Este resultado lo recibe el **.then()** dentro de su callback.

Hay casos donde tendremos **pedidos anidados**, es decir, que los .then() tengan promesas dentro. En este caso, utilizamos otro .then() que entre en ejecución apenas se resuelva el anterior

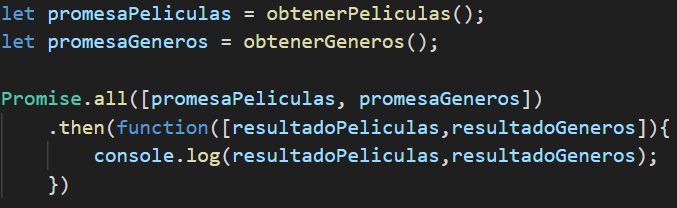


Aquí, **dataFiltrada** es la información que devuelve el return del primer .then().

Por último, en caso de NO obtener un resultado, se genera un error. Podemos interceptarlo con **.catch()**.Es dentro de este método donde decidiremos que hacer con el error. Este llega como un parámetro dentro del callback del .catch().



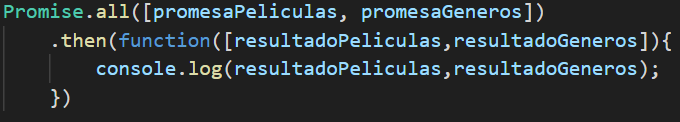
A veces necesitaremos que dós o más promesas se resuelvan para hacer una acción. Para eso usamos **Promise.all()**. Se usa de la siguiente forma:



Primero, debemos guardar en variables todas las promesas a cumplir:



Luego, usamos el Promise.all() para evaluarlas todas, y en el callback del .then() enviamos un array con los resultados de las promesas como parámetro (y ejecutamos el código que precisemos)



Documentación respecto a promesas:

<https://developer.mozilla.org/es/docs/Web/JavaScript/Guide/Using_promises>

<https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global_Objects/Promise/all>

Sequelize

Sequelize es un ORM (Object Relational Mapper) que nos ayuda a conectarnos e interactuar con bases de datos relacionales. Esto nos permite escribir comandos en javascript en vez de tener que escribir querys para acceder a la base de datos

Para instalarlo:

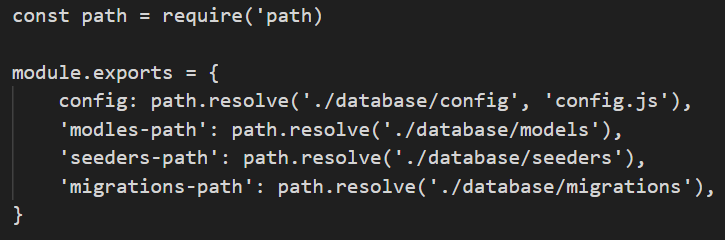
npm install -g sequelize sequelize-cli

npm install sequelize

Debemos instalar un paquete para decirle a sequelize que tipo de motor de base de datos estamos utilizando, y se pueda conectar al mismo:

npm install mysql2

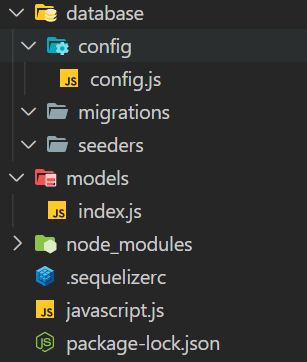
Luego, debemos crear dentro de la carpeta **src** del proyecto el archivo **.sequelizerc**, con lo siguiente (archivo txt en la carpeta de este word):

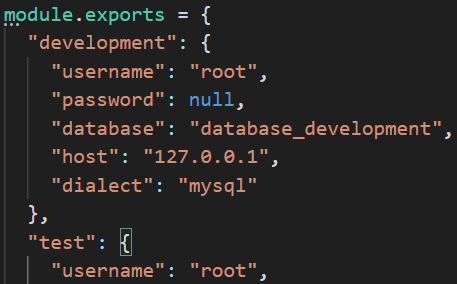


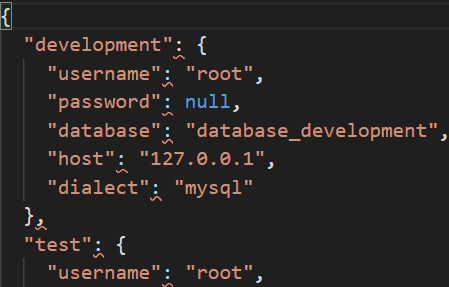
A continuación, con el comando cd nos paramos en **src ,** yen la terminal escribimos:

sequelize init

Esto creara las carpetas **database** y **models**, cada una con varias cosas adentro.



Es importante ir a **database > config > config.js** y cambiar el código agregando module.exports al principio:



Aquí mismo debemos configurar todos los campos de acuerdo a nuestra base de datos.

Por último, en **models > index.js** al final se exporta una variable llamada **db.** Esta es la variable que utilizaremos en el proyecto para interactuar con la base de datos.

Documentación:

<https://sequelize.org/>

Modelos

El modelo es la M en el patrón de diseño MVC. Este contiene únicamente los datos puros de aplicación. No contienen lógica que describa como pueden presentarse los datos a un usuario. Lo ideal es que el modelo sea independiente del sistema de almacenamiento.

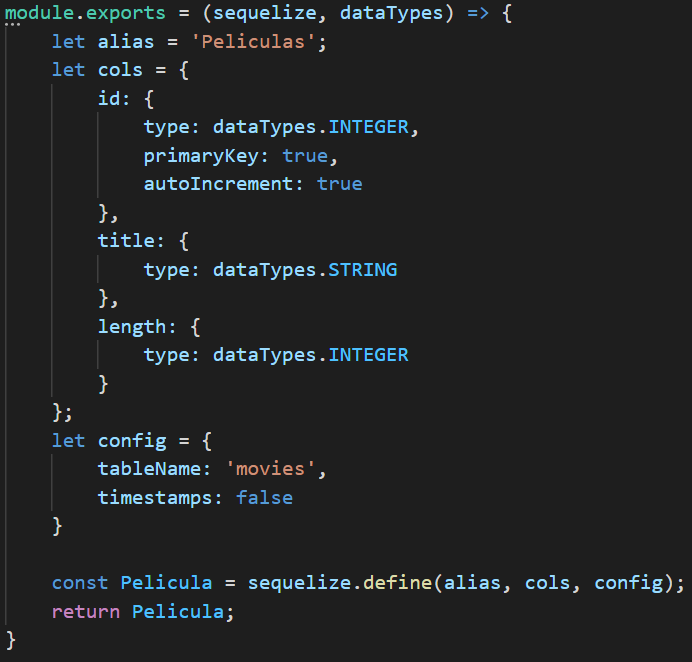
En resumen, un modelo es la **representación de nuestra tabla en código**. Así podremos realizar consultas e interacciones con la base de datos de manera simplificada, en este caso, con Sequelize.

# Crear un modelo para una tabla:

Siempre el modelo debemos crearlo en la ruta **/database/models**. Es un archivo .js,y su nombre debe estar escrito en UpperCamelCase y en singular.



Dentro de este archivo deberemos hacer lo siguiente (choclazo incoming):



Bueno, a desmenuzarlo.

Primero, debemos exportar una función que tendrá dos parámetros, **sequelize**, y **dataTypes:**



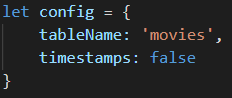
Dentro de esta función, tenemos que crear una constante que será la que se exportará, con tres parámetros: **alias, cols,** y **config (opcional).** Importante aclarar que esta constante debe tener el nombre de la tabla en singular y comenzando con mayúscula:



Bueno, estos tres parámetros los tenemos definidos previamente. Vamos uno por uno, del mas sencillo al más complejo (alias, config, cols).

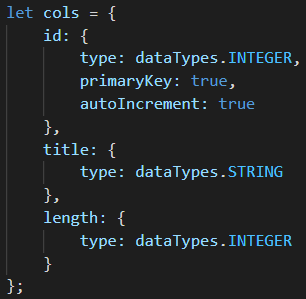


En **alias** se suele poner el nombre del modelo en plural.

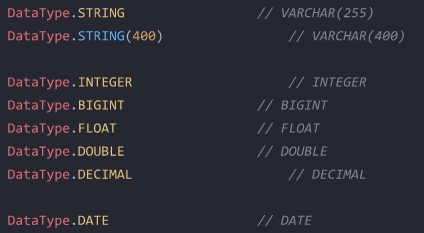


**config** es opcional pero recomendado. El **tableName** no es necesario ya que sequelize tomará el plural del nombre del archivo y lo usará para referirse a la tabla. Si por algún motivo la tabla se llamara distinto, aquí debemos aclararlo.

Por otro lado, **timestamps** sirve para aclarar si usamos o no dos columnas llamadas **createdAt,** y **updatedAt**. Estas sirven para llevar registro de las modificaciones en la tabla. Sequelize asume que las usamos, así que si no es el caso, deberemos aclarárselo con **timestamps: false.**



Bien, aquí le definimos la estructura de nuestra tabla. **id**, **title,** **length** son los nombres de las tablas en la base de datos. Dentro es OBLIGATORIO escribir el type, para que sepa qué tipo de dato se guarda. Pueden ser los siguientes:



Luego se pueden aclarar otras cosas, como si es un primary key, una foreign key, si debe ser no nulo, etc (ver apuntes clase 29 creo).

findAll(), findByPk() y findOne()

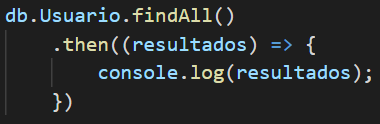
Para utilizar estos métodos debemos siempre importar donde queramos usarlo el modelo:



Luego escribiremos db.nombreTabla.funcion para interactuar con la tabla que queramos, por ejemplo:

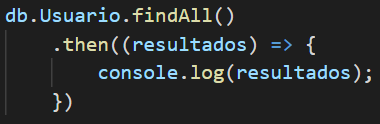


Esto es una promesa, asi que tenemos que utilizarlo con el .then()



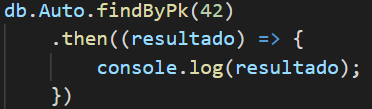
# findAll()

Es el equivalente a SELECT \* en mysql. Se hace de la siguiente forma:



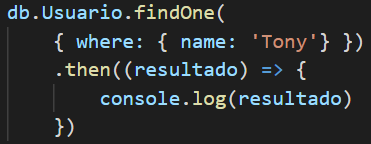
# findByPk()

Sirve para buscar un registro cuya clave primaria sea el valor del parámetro pasado:



# findOne()

Permite buscar resultados que coincidan con los atributos indicados en el objeto literal que recibe el método:

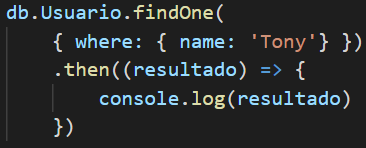


Documentación:

[https://sequelize.org/master/manual/model-querying-basics.html#simple-select-queries](https://sequelize.org/master/manual/model-querying-basics.html" \l "simple-select-queries)

Where y operadores

El where de mysql se hace pasando un objeto literal (lo vimos en el findOne) con la consulta:



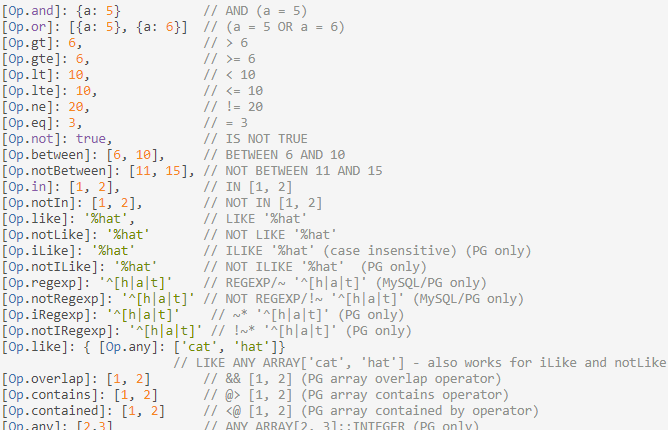
Esto solo nos sirve para buscar por “igual”. Para buscar con estilo LIKE, mayor, menor, etc, debemos primero importar los operadores:



Luego, para usar un LIKE deberíamos hacerlo así (apellidos que tengan una s):

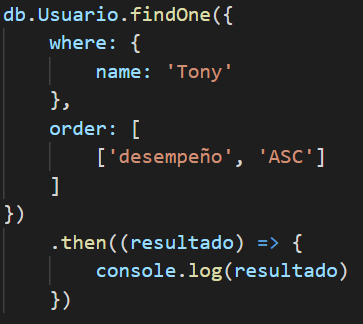


Aquí tenemos todos los operadores (atentos con la versión): [https://sequelize.org/v4/manual/tutorial/querying.html#operators](https://sequelize.org/v4/manual/tutorial/querying.html" \l "operators)



Order y Limit

El ORDER se implementa de la siguiente manera:



LIMIT y OFFSET son bastante directos:

