Modelos



Índice

- 1. Creando un modelo
- 2. <u>Método .define()</u>
- 3. <u>Tipos de datos</u>
- 4. <u>Timestamps</u>
- 5. <u>Configuraciones adicionales</u>



A la hora de hacer una consulta a la base de datos, un **modelo** va a retornarnos información en un formato útil y cómodo para luego operar con la misma.







Modelos

En los patrones de diseño MVC (Modelo - Vista - Controlador), los **modelos** contienen únicamente los datos puros de aplicación. No contienen lógica que describa cómo pueden presentarse los datos a un usuario. Este puede acceder a la capa de almacenamiento de datos. Lo ideal es que el modelo sea independiente del sistema de almacenamiento.

Modelos

Es decir, un modelo es la **representación de nuestra tabla en código**. Con esto obtenemos recursos que nos permiten realizar consultas e interacciones con la base de datos de manera simplificada usando, en este caso, **Sequelize**.

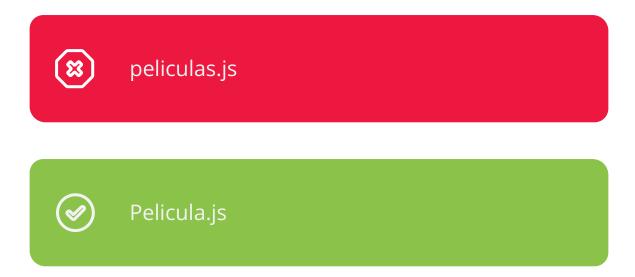
Siempre creamos un modelo para cada tabla de nuestra base de datos. La ruta donde los almacenamos es /database/models.



Los modelos son archivos JS, por lo tanto deben ser creados con esa extensión.



Los nombres de los modelos deben estar escritos en UpperCamelCase y en singular.



Un modelo es naturalmente una función que debemos definir y luego exportar con module.export. Esta función recibe dos parámetros. En primer lugar, el objeto sequelize para poder acceder a su método define() y, en segundo lugar, necesitamos traer al objeto DataTypes que nos dará la posibilidad de decirle a nuestras columnas qué tipo de datos permitirán.



Recordemos que al estar tratarse de parámetros no es necesario que se llamen así, pero solemos hacerlo para entender por qué los usamos.

2 Método .define()

Método .define()

El método **define()** nos permite definir asignaciones entre un modelo y una tabla. Este recibe **3 parámetros**. El primero es un alias que identifica al modelo, el segundo es un objeto con la configuración de las columnas en la base de datos y el tercero es otro objeto con configuraciones adicionales (parámetro opcional). Lo que devuelva **define()** será almacenado en una variable con el nombre del modelo para luego ser retornada por la función que creamos.

```
const Pelicula = sequelize.define(alias, cols, config);
return Pelicula;
```

Alias

Como mencionamos en el slide anterior, el primero es un alias que utiliza Sequelize para **identificar al modelo**. No es algo determinante. Solemos asignarle el mismo nombre del modelo como **String**.

```
const Pelicula = sequelize.define("Pelicula", cols, config);
return Pelicula;
```

3 Tipos de datos

Tipos de datos en Sequelize

Dentro de nuestro segundo parámetro que llamamos **cols** se encuentra un objeto que nos permite, en el segundo parámetro del **define()**, definir qué tipos de datos deben recibir las columnas en la base de datos.

```
cols = {
    id: {
        type: DataTypes.INTEGER
    },
    name: {
        type: DataTypes.STRING
    },
    admin: {
        type: DataTypes.BOOLEAN
    }
}
```

Ejemplos más utilizados

```
DataType.STRING
                                     // VARCHAR(255)
     DataType.STRING(400)
                                          // VARCHAR(400)
     DataType.INTEGER
                                          // INTEGER
     DataType.BIGINT
                                     // BIGINT
{}
     DataType.FLOAT
                                     // FLOAT
     DataType.DOUBLE
                                     // DOUBLE
     DataType.DECIMAL
                                          // DECIMAL
     DataType.DATE
                                     // DATE
```

Contamos con más tipos de datos en la documentación oficial de Sequelize. Para acceder a ella, podemos hacer clic en el siguiente <u>link</u>.

4 Timestamps

Timestamps

```
module.exports = (sequelize, DataTypes) => {
  const Usuario = sequelize.define("Usuario", {
    email: {
      type: DataTypes.STRING
      },
      createdAt: {
```

Los **timestamps** no son obligatorios, pero la mayoría de las tablas suelen tenerlos y forman parte del estándar. Estos deben llamarse de la misma forma que se ve en el ejemplo.

```
createdAt: {
    type: DataTypes.DATE
    },
    updatedAt: {
    type: DataTypes.DATE
    },
});
return Usuario;
}
```

Campos que guardan la fecha de creación y última edición.

5 Configuraciones adicionales

Configuraciones adicionales

Dentro de nuestro tercer parámetro del **define()** podemos configurar cosas adicionales. Por ejemplo, si el nombre de nuestra tabla está en inglés y el de nuestro modelo en español, deberíamos aclararle al modelo que esto es así mediante un objeto literal, como en el ejemplo de la siguiente diapositiva.

```
module.exports = (sequelize, DataTypes) => {
         const Pelicula = sequelize.define("Pelicula",
      // Configuraciones de las columnas.
            },
               tableName: 'movies',
{}
      //Si el nombre de la tabla no coincide con el del modelo
               timestamps: false,
      //Si no tengo timestamps
            });
        return Pelicula;
```

Documentación



Para más información acerca de DataTypes y qué cosas se pueden configurar de una misma columna en la DB ingresa al siguiente: <u>link</u>.

DigitalHouse>