Coding Bootcamp

Sprint 2

Temario

Temario

- ORM.
- Sequelize.
  - Métodos más importantes
  - Otras funcionalidades.

ORM

ORM

Un ORM (Object-Relational Mapping) es una técnica de programación que permite **mapear** modelos de un sistema con tablas de una base de datos relacional.

Al usar un ORM, podemos interactuar con objetos en lugar de tener tener que escribir consultas SQL “a mano”.

Al crear, editar o eliminar un objeto, el ORM construye y ejecuta “por atrás” las consultas necesarias en la base de datos.

En general, los ORM pueden funcionar con más de un DBMS, fácil de cambiar entre uno u otro.

Sequelize

Sequelize (1/5)

Es un ORM para Node.js que funciona con los siguientes DBMS:

- Postgres.
- MySQL.
- MariaDB.
- SQLite.
- Microsoft SQL Server.

Es decir, gracias a Sequelize podremos interactuar con cualquiera de estos DBMS sin necesidad de escribir consultas SQL.

Está basado en promesas, ideal para evitar *Callback Hells*.

Documentación: https://sequelize.org.

Sequelize (2/5)

Instalación:

```
npm i sequelize
npm i mysql2
```

En caso de usar otro DBMS será necesario instalar otro módulo

Luego importarlo con:

```
const { Sequelize, Model, DataTypes } = require("sequelize");
```

Sequelize (3/5)

Ejemplo básico. Primero que nada hay que crear una instancia de Sequelize:

```
const sequelize = new Sequelize(
  process.env.DB_DATABASE, // Ej: hack_academy_db
  process.env.DB_USERNAME, // Ej: root
  process.env.DB_PASSWORD, // Ej: root
  {
    host: process.env.DB_HOST, // Ej: 127.0.0.1
    dialect: process.env.DB_CONNECTION, // Ej: mysql
  }
);
```

En caso de usar otro DBMS será necesario instalar otro módulo

Recordar: `process.env` es donde se guardan las variables de entorno, luego de usar el módulo dotenv.

Sequelize (4/5)

Luego es necesario crear un modelo:

```
class User extends Model {}
User.init(
  {
    fullname: DataTypes.STRING,
    birthday: DataTypes.DATE,
  },
  { sequelize, modelName: "user" }
);
```

Los modelos son la esencia de Sequelize. Son una abstracción que representan una tabla de la Base de Datos.

Ver tipos de datos disponibles.

Sequelize (5/5)

Finalmente se utiliza el método `sync` para crear la tabla a partir del modelo.

```
sequelize.sync({ force: true }).then(() => {
  console.log(`¡Las tablas fueron creadas!`);
});
```

👉 En necesario tener cuidado con este método ya que, en caso de que una tabla exista previamente, la misma se borra y se crea de nuevo, perdiendo todos los datos que allí estaban. En un ambiente de desarrollo este no suele ser un problema. Para un ambiente de producción, usar migraciones en lugar de `sync`.

Métodos más importantes

Sequelize – Obtener registros

Ejemplo: obtener todos lo usuarios de la BD:

```
User.findAll().then((users) => {
  console.log(users);
});
```

Ejemplo: obtener el usuario con id = 123.

```
User.findByPk(123).then((user) => {
  console.log(user);
});
```

Sequelize – Crear un registro

Ejemplo: insertar un usuario en la BD:

```
User.create({
  firstname: "María",
  lastname: "Pérez",
  email: "mariaperez@gmail.com",
}).then((user) => {
  console.log(user);
});
```

Sequelize – Eliminar un registro

Ejemplo: eliminar a todos los usuarios llamados “Pablo” de la BD:

```
User.destroy({
  where: {
    firstname: "Pablo",
  },
}).then(() => {
  console.log("¡Usuarios eliminados!");
});
```

Sequelize – Editar varios registros

Ejemplo: modificar el apellido de todos los “Gómez” a “Pérez”:

```
User.update(
  { lastname: "Pérez" },
  {
    where: {
      lastname: "Gómez",
    },
  }
).then(() => {
  console.log("¡Usuarios actualizados!");
});
```

Sequelize – Editar un registro

Ejemplo: modificar el email del usuario con id = 123.

```
User.findByPk(123).then((user) => {
 user
   .update({
     email: "mperez@gmail.com",
   })
   .then((user) => {
     console.log(user);
   });
});
```

Otras funcionalidades

Otras funcionalidades

Recomendamos que investiguen sobre estas otras funcionalidades de Sequelize:

- Validaciones.
- Asociaciones / Relaciones entre entidades.
- Migraciones.
- Sequelize CLI.