DigitalHouse > Coding School

Programación Web Full Stack 

Modelos y consultas básicas

Práctica integradora

Objetivo

Finalmente es hora de aprender a conectar nuestro proyecto de Express a nuestra **base de datos**. De esta manera buscamos que la información que vemos en nuestro sitio web muestre información real proveniente de una fuente de información veloz y escalable. El objetivo será el de únicamente listar y mostrar la información, pero para ello deberemos establecer la conexión con la base de datos previamente.

¡Buena suerte! 

1

Micro desafío - Paso 1:

Utilizaremos de base el siguiente **proyecto creado con Express**, junto a esta base de datos: **movies_db**.

Una vez realizada la instalación de todas las dependencias del proyecto, debemos efectuar lo siguiente:

1. **Instalar el paquete Sequelize:** para lograr el objetivo es fundamental lograr instalar Sequelize y los paquetes relacionados necesarios.

```
npm install sequelize-cli -

g npm install sequelize npm

install mysql2
```

2. Una vez instalados los paquetes, será fundamental la creación del archivo **.sequelizerc** (lo podemos crear en la raíz del proyecto) con la siguiente estructura:

```
const path = require('path')

module.exports = { config:
  path.resolve('./src/database/config', 'config.js'),
  'models-path': path.resolve('./src/database/models'),
  'seeders-path': path.resolve('./src/database/seeders'),
  'migrations-path': path.resolve('./src/database/migrations'),
}
```

3. Luego, es fundamental correr el comando **sequelize init**.
4. Finalmente no debemos olvidarnos de modificar el archivo **/database/config/config.js** agregando *module.exports* al principio del archivo y configurando los datos de conexión a la base de datos.

2

Micro desafío - Paso 2:

Algo fundamental al inicializar un proyecto es explicarle a **sequelize** las tablas que tiene nuestra base de datos. Para esto debemos crear los **modelos** para las tablas **movies y genres** (Películas y Géneros).

No debemos olvidarnos de aclarar el nombre de la tabla, si usa timestamps y todas sus columnas con su tipo. Recomendamos aclarar los datos como columnas que aceptan nulo así como clave primaria y autoincremental.

```
module.exports = (sequelize, dataTypes) => {
    let alias = 'Movie';
    let cols = {
        id: {
            type: dataTypes.INTEGER,
            primaryKey: true,
            autoIncrement: true
        },
        title: {
            type: dataTypes.STRING
        },
        rating: {
            type: dataTypes.INTEGER
        },
        length: {
            type: dataTypes.INTEGER
        },
        awards: {
            type: dataTypes.INTEGER
        },
        release_date: {
            type: dataTypes.DATE
        }
    };
    let config = {
        tableName: 'movies',
        timestamps: false
    };
    const Movie = sequelize.define(alias, cols, config)

    return Movie
}
```

```
module.exports = (sequelize, dataTypes) => {
    let alias = 'Genre';
    let cols = {
        id: {
            type: dataTypes.INTEGER,
            primaryKey: true,
            autoIncrement: true
        },
        name: {
            type: dataTypes.STRING
        },
        ranking: {
            type: dataTypes.INTEGER
        }
    };
    let config = {
        tableName: 'genres',
        timestamps: false
    };
    const Genre = sequelize.define(alias, cols, config)

    return Genre
}
```

Micro desafío - Paso 3:

Acceder al controlador → controllers/moviesController.js

Es fundamental que el controlador importe la variable **db** de **./database/models/index.js** ya que esta variable tendrá nuestra conexión a la base de datos.

Para la construcción de esta versión del sitio web, el cliente espera contar con la posibilidad de acceso a las siguientes URLs: (**Las mismas ya están creadas, solo debes quitarle el comentario a medida que vas trabajando**)

- /movies (GET)
  - Se deberán listar todas las películas de la base de datos. Cada título de película deberá ser un hipervínculo para ver el detalle de la misma.

(Todas las vistas, ya están creadas)

- El controlador deberá utilizar la conexión a base de datos y el modelo de **Movie** ya creado. Con eso en mente, el método **findAll** permitirá obtener todas las películas de la base de datos.

```
list: (req,res) => {
    db.Movie.findAll()
        .then (movies => {
            res.render('moviesList.ejs', {movies})
        }).catch((error) => {
            if (error) throw error;
            })
},
```

- /movies/detail/:id (GET)
    - Detalle de la película. Se deberá mostrar la película correspondiente al id que aparezca en la URL. Cada película deberá listar sus datos (Título, rating, premios, duración y fecha de estreno).
    - El controlador deberá utilizar la conexión a la base de datos y el modelo de **Movie** ya creado. Con eso en mente, el método **findByPk** permitirá obtener la película buscada. Recordemos utilizar **req.params** para obtener el id de la URL.

```
detail: (req,res) => {
    db.Movie.findByPk(req.params.id)
        .then (movie => {
            res.render('moviesDetail.ejs', {movie})
        }).catch((error) => {
            if (error) throw error;
            })
},
```

- /movies/new (GET)
    - El controlador deberá utilizar la conexión a base de datos y el modelo de **Movie** ya creado. Con eso en mente el método **findAll**, junto con el parámetro de configuración **order** permitirá obtener todas las películas de la base de datos.

```
    new: (req,res) => {
        db.Movie.findAll({
            order: [
                ['title', 'ASC']
            ]
        }).then (movies => {
            res.render('moviesList.ejs', {movies})
        }).catch((error) => {
            if (error) throw error;
            })
    },
```

- /movies/recommended (GET)
  - Deberá mostrar las últimas 5 películas ordenadas según su fecha de estreno.
  - El controlador deberá utilizar la conexión a la base de datos y el modelo de **Movie** ya creado. Con eso en mente, el método **findAll** permitirá obtener todas las películas de la base de datos. En este caso necesitaremos agregar el parámetro de configuración **where** y utilizar los operadores de Sequelize para resolver el desafío.

```
    recomended: (req,res) => {
        db.Movie.findAll({
            order: [
                ['release_date']
            ],
            limit: 5
        }).then (movies => {
            res.render('moviesList.ejs', {movies})
        }).catch((error) => {
            if (error) throw error;
            })
    },
```

Acceder al controlador → controllers/genresController.js

No olvidar importar en el controlador la variable **db** de **./database/models/index.js** ya que esta variable tendrá nuestra conexión a la base de datos.

Para la construcción de esta versión del sitio web, el cliente espera contar con la posibilidad de acceso a las siguientes URLs:

- /genres (GET)
  - Se deberán listar todos los géneros de la base de datos. Cada género deberá ser un hipervínculo para ver el detalle del mismo.
  - El controlador deberá utilizar la conexión a la base de datos y el modelo de **Genre** ya creado. El método **findAll** permitirá obtener todos los géneros de la base de datos.

```
list: (req,res) => {
    db.Genre.findAll()
        .then (genres => {
            res.render('genresList.ejs', {genres})
        }).catch((error) => {
            if (error) throw error;
            })
},
```

- /genres/detail/:id (GET)
  - Detalle del género. Se deberá mostrar del género correspondiente al id que aparezca en la URL. Cada género deberá listar sus datos (Id, name, ranking).
  - El controlador deberá utilizar la conexión a la base de datos y el modelo de **Genre** ya creado. De esta manera, el método **findByPk** permitirá obtener el género buscado. Recordemos utilizar **req.params** para obtener el id de la URL.

```
detail: (req,res) => {
    db.Genre.findByPk(req.params.id)
        .then (genre => {
            res.render('genresDetail.ejs', {genre})
        }).catch((error) => {
            if (error) throw error;
            })
},
```

Bonus track:

Si logramos realizar toda la práctica, una buena idea es replicar el proceso, pero con el **modelo Actores**.

```
module.exports = (sequelize, dataTypes) => {
    let alias = 'Actor';
    let cols = {
        id: {
            type: dataTypes.INTEGER,
            primaryKey: true,
            autoIncrement: true
        },
        first_name: {
            type: dataTypes.STRING
        },
        last_name: {
            type: dataTypes.STRING
        },
        rating: {
            type: dataTypes.DECIMAL(3,1)
        },
        favorite_movie_id: {
            type: dataTypes.INTEGER(10).UNSIGNED
        }
    };
    let config = {
        tableName: 'actors',
        timestamps: false,
    };
    const Actor = sequelize.define(alias, cols, config)

    return Actor
}
```

```
const db = require('../database/models/index.js');
const Movies = db.Movie;

const actorsController = {
    list: (req,res) => {
        db.Actor.findAll()
            .then (actors => {
                res.render('actorsList.ejs', {actors})
            }).catch((error) => {
```

```
                if (error) throw error;
                })
    },
    detail: (req,res) => {
        db.Actor.findByPk(req.params.id)
            .then (actor => {
                res.render('actorsDetail.ejs', {actor})
            }).catch((error) => {
                if (error) throw error;
                })
    },

}

module.exports = actorsController
```

Conclusión

Hasta ahora no habíamos obtenido información de una base de datos formal, es por ello que resulta fundamental tener muy claro la identificación de cada una de las entidades, así como cada uno de los campos de nuestras tablas. Eso nos resultará de mucha utilidad para construir en nuestra aplicación cada uno de los modelos de las mismas, para luego lograr ubicar información en ellas, las cuales serán enviadas desde nuestros controladores a las vistas.

Como podrán notar Sequelize, nos ofrece todo un conjunto de métodos para lograr manipular nuestras tablas.

¡Hasta la próxima!