Sequelize
Modelos

DigitalHouse>

Índice

> Ao fazer uma consulta ao banco de dados, um **modelo** retornará as informações em um formato útil e confortável e então operará com elas.

Sequelize - Models

Em padrões de projeto MVC (Model - View - Controller), os **modelos** contêm apenas os dados puros de aplicação; não há lógica que descreva como os dados podem ser apresentados a um usuário. Isso pode acessar a camada de armazenamento de dados. Idealmente, o modelo deve ser independente do sistema de armazenamento.

Sequelize - Models

Quer dizer...

Um **modelo** é a representação de nossa tabela em código. Com isso, obtemos recursos que nos permitem fazer consultas e interações com o banco de dados de forma simplificada, utilizando, neste caso, o sequelize.

Models - Criando um model

Sempre criamos um modelo para cada tabela em nosso banco de dados. O caminho onde o armazenamos é **/database/models.**

Os modelos são arquivos JS, portanto, devem ser criados com essa extensão.

Os nomes dos modelos devem ser escritos em UpperCamelCase e no singular.

Models - Criando um model

filmes.js

Filme.js

Models - Criando um modelo

Um modelo é naturalmente uma função que devemos definir e exportar com **module.export**. Essa função recebe dois parâmetros. Primeiramente, ele recebe o objeto **sequelize** para poder acessar seu método **define()**, e em segundo lugar, precisamos trazer o objeto **DataTypes**, que nos dará a possibilidade de dizer às nossas colunas que tipo de dados elas permitirão.

Lembre-se de que, ao lidar com parâmetros, não é necessário que sejam chamados assim, mas normalmente fazemos isso para entender por que os usamos.

Models - Método .define()

O método **define()** nos permite definir atribuições entre um modelo e uma tabela, que recebe **3 parâmetros**. O primeiro é um alias que identifica o model, o segundo é um objeto com a configuração das colunas do banco de dados e o terceiro é outro objeto com configurações adicionais (parâmetro opcional). O que **define()** retorna será armazenado em uma variável com o nome do model, para posteriormente ser retornado pela função que criamos.

```
{}
const Filme = sequelize.define(alias, cols, config);
return Filme;
```

Models - Alias

Como mencionamos no slide anterior, o primeiro é um alias que o Sequelize usa para **identificar o modelo**. Não é algo determinante. Normalmente, atribuímos o mesmo nome do modelo como **String**.

```
{}
const Filme = sequelize.define("Filme", cols, config);
return Filme;
```

Models - Tipos de dados no sequelize

Dentro de nosso segundo parâmetro, que chamamos de **cols**, existe um objeto que nos permite definir quais tipos de dados as colunas devem receber no banco de dados.

{}

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

DataTypes - Exemplos mais 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
```

Você encontrará mais tipos de dados na documentação oficial do Sequelize. Para acessá-la, clique no seguinte **Link**.

Model - Timestamps

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

  return Usuario;
}
```

Os **timestamps** não são obrigatórios, mas a maioria das tabelas geralmente os contém e fazem parte do padrão. Eles devem ser nomeados da mesma forma que no exemplo.

Campos que salvam a data de criação e última edição.

Models - Configuraçõe adicionais

Dentro de nosso terceiro parâmetro de **define()**, podemos configurar coisas adicionais. Por exemplo, se o nome de nossa tabela está em inglês e o de nosso modelo em português, devemos deixar claro para o modelo que este é o caso por meio de um objeto literal, como no exemplo do próximo slide.

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

    const Filme = sequelize.define("Filme",
        {
  // Configurações das colunas.
        },
        {
{}          tableName: 'movies',
  //Se o nome da tabela coincide com o model
            timestamps: false,
  //Se tenho timestamps
        });

    return Filme;
  }
```

Sequelize - Documentação

Para obter mais informações sobre DataTypes e o que pode ser configurado na mesma coluna no DB, siga o seguinte Link.

DigitalHouse>