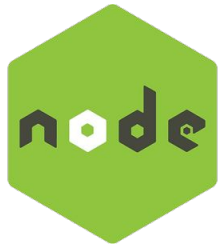
NodeJS



Prof. Celso Henrique Masotti

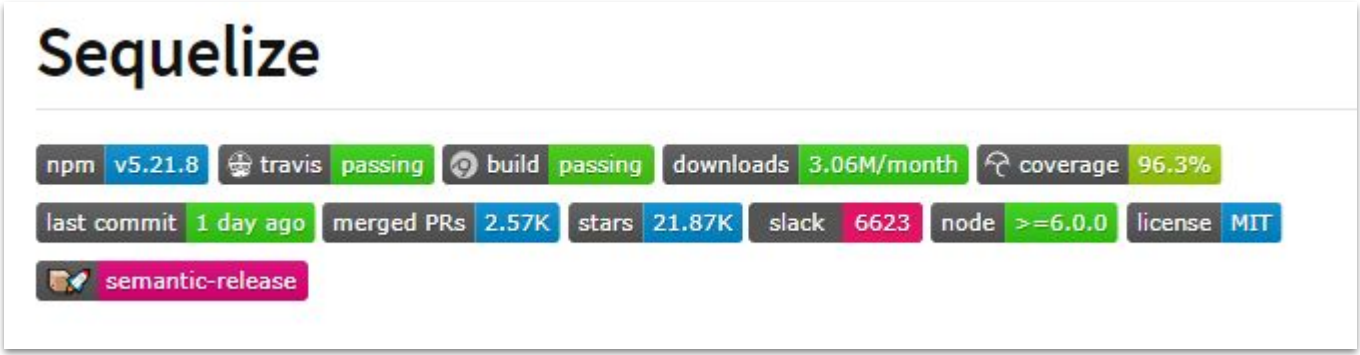
Você verá nesta aula:

Objetivo: Usar o Sequelize.

Sequelize
Sequelize - Instalação
Sequelize - Promises
Sequelize - then e catch
Sequelize - Conectando o BD
Sequelize - models
Sequelize - models - tipagem
Sequelize - models - estrutura
Sequelize - Inserindo Registro

Sequelize

Sequelize é um pacote **nodeJS** para administração de bancos de dados. Com este instalado tudo se torna mais fácil para manipular dados evitando queries longas e que podem dar bugs.



Sequelize

Sequelize é um ORM (Object-relational mapping), ou seja, uma técnica de desenvolvimento utilizada para reduzir a impedância da programação orientada aos objetos utilizando bancos de dados relacionais.

Trata-se de um pacote facilitador intermediando o SGBD e as necessidades do desenvolvedor.

Sequelize - Instalação

A instalação do pacote Sequelize deve ser realizada em duas etapas. A primeira deve-se instalar o pacote “sequelize” propriamente dito e a segunda trata-se do pacote contendo os recursos para o SGBD utilizado no projeto.

Installation

```
$ npm install --save sequelize # This will install v5

# And one of the following:
$ npm install --save pg pg-hstore # Postgres
$ npm install --save mysql2
$ npm install --save mariadb
$ npm install --save sqlite3
$ npm install --save tedious # Microsoft SQL Server
```

Sequelize - Instalação

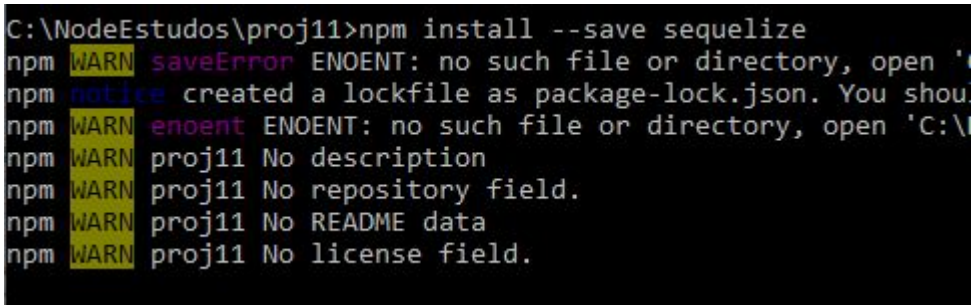
Os procedimentos são:

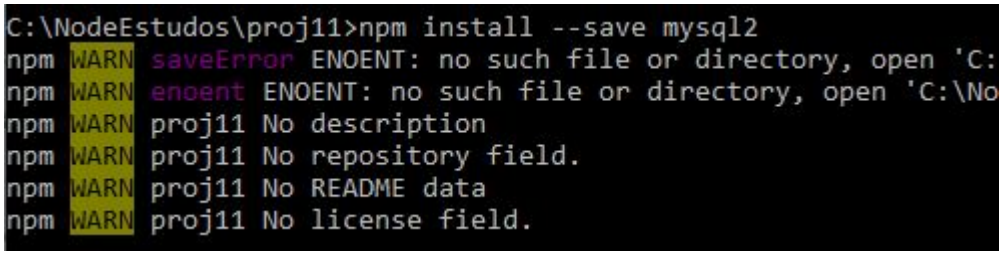
Vá até sua pasta de projetos e digite no console:

```
npm install --save sequelize
```

Depois de instalado, solicite o pacote do SGBD:

```
npm install --save mysql2
```





Sequelize - Promises

As *promises* não eram nativas do JavaScript até o ES6, quando houve uma implementação oficial na linguagem, antes delas, a maioria das funções usavam callbacks.

Promises são um padrão de desenvolvimento que visam representar a conclusão de operações assíncronas. Ou melhor: trata-se de um objeto que representa a eventual conclusão ou falha de uma operação assíncrona. Como a maioria das pessoas consomem *promisses* já criadas, este guia explicará o consumo de *promisses* devolvidas antes de explicar como criá-las.

Essencialmente, uma promise é um objeto retornado para o qual você adiciona callbacks, em vez de passar callbacks para uma função.

Sequelize - Promises

Uma Promise também possuí diferentes estados, sendo alguns deles:

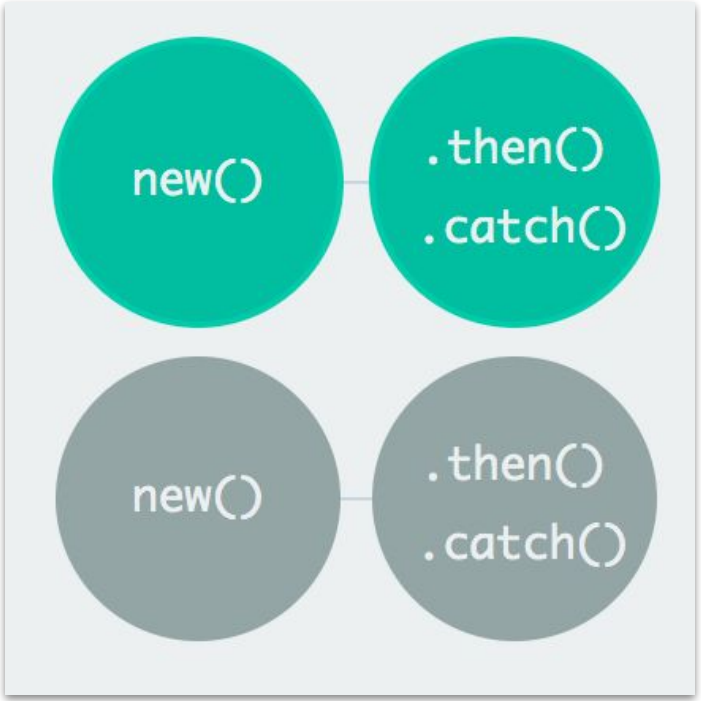
- Pendente (Pending).
- Resolvida (Resolved) (não está na documentação, mas gosto de definir esse estado também).
- Rejeitada (Rejected).
- Realizada (Fulfilled).
- Estabelecida (Settled).

Geralmente os estados mais utilizados são dois (2), sendo eles: Resolvida e Rejeitada.

Sequelize - then e catch

“then” e “catch” são *promises* bastante utilizadas na recepção callback de conexões de bancos de dados.

Uma conexão só pode ter dois retornos: sucesso ou fracasso. Quando a conexão com o banco de dados foi realizada com sucesso usamos o “then” para alguma ação, caso contrário usamos o “catch”.



Sequelize - Conectando o BD

A conexão com o Banco de Dados utilizando o Sequelize possui dois blocos de códigos. Primeiro bloco trabalha o módulo e o objeto que efetivamente faz a conexão com o banco de dados, para isso passamos as informações de de conexão:

- nome do banco de dados,
- usuário,
- senha de acesso.

e em json, informamos o endereçamento do SGBD e qual SGBD desejamos, já que o sequelize trabalha com uma grande quantidade de SGBDs.

```
const Sequelize = require('sequelize');
const sequelize = new Sequelize('sistemadecadastro','root','12345',{
  host: "localhost",
 dialect: 'mysql'
})
```

Sequelize - Conectando o BD

Segundo bloco trabalha com a comunicação com o usuário através de *promises* “then” (retornando aviso de conexão) e “catch” (retornando aviso de falha de conexão), tendo a seguinte estrutura linear:

`sequelize.authenticate().then().catch()`

```
sequelize.authenticate().then(function(){
    console.log("Conectado ao Banco de Dados com sucesso!")
}).catch(function(erro){
    console.log("Falha ao se conectar: " + erro)
})
```

Sequelize - Conectando o BD

Visão completa do código com comentários.

proj11 > JS connect.js > ...

```
1   const Sequelize = require('sequelize');
2
3   // inserir dados de conexão: nome BD; usuário; senha; abrir json.
4   const sequelize = new Sequelize('sistemadecadastro','root','12345',{
5    // inserir endereço do SGBD e tipo de SGBD
6    host: "localhost",
7    dialect: 'mysql'
8   })
9
10  // certificando a autenticação
11  // usando promise 'then' e 'catch'
12  sequelize.authenticate().then(function(){
13      console.log("Conectado ao Banco de Dados com sucesso!")
14  }).catch(function(erro){
15      console.log("Falha ao se conectar: " + erro)
16  })
```

Sequelize - Conectando o BD

Ao chamarmos nosso arquivo pelo node, se tudo estiver correto, o callback será capturado pelo “then” e imprimirá no console o texto “Conectado ao Banco de Dados com sucesso!”

```
C:\NodeEstudos\proj11>node connect.js
Executing (default): SELECT 1+1 AS result
Conectado ao Banco de Dados com sucesso!
```

Sequelize - models

Como modelos (models) podemos criar tabelas em um banco de dados. Para sua funcionalidade fazemos uso:

- sequelize.define(); que define as características estruturais da tabela que precisamos;
- sequelize.sync(); que realiza a estrutura acima definida no banco de dados.

Sequelize - models - tipagem

Tipagem de campos utilizados no *sequelize.define()*

```
STRING                      // VARCHAR(255)
STRING(1234)                // VARCHAR(1234)
STRING.BINARY               // VARCHAR BINARY
TEXT                        // TEXT
TEXT('tiny')                // TINYTEXT

INTEGER                     // INTEGER
BIGINT                      // BIGINT
BIGINT(11)                  // BIGINT(11)

FLOAT                       // FLOAT
FLOAT(11)                   // FLOAT(11)
FLOAT(11, 12)               // FLOAT(11,12)
```

Sequelize - models - tipagem

Tipagem de campos utilizados no *sequelize.define()*

```
REAL                        // REAL         PostgreSQL only.
REAL(11)                    // REAL(11)     PostgreSQL only.
REAL(11, 12)                // REAL(11,12)  PostgreSQL only.

DOUBLE                      // DOUBLE
DOUBLE(11)                  // DOUBLE(11)
DOUBLE(11, 12)              // DOUBLE(11,12)

DECIMAL                     // DECIMAL
DECIMAL(10, 2)              // DECIMAL(10,2)

DATE                        // TIMESTAMP WITH TIME ZONE for postgres
DATE(6)                     // DATETIME(6) for mysql 5.6.4+. F
DATEONLY                    // DATE without time.
BOOLEAN                     // TINYINT(1)
```

Sequelize - models - tipagem

Tipagem de campos utilizados no *sequelize.define()*

```
ENUM('value 1', 'value 2')  // An ENUM with allowed values 'value 1' and 'value 2'

ARRAY(Sequelize.TEXT)        // Defines an array. PostgreSQL only.
ARRAY(Sequelize.ENUM)        // Defines an array of enum. PostgreSQL only.

JSON                         // JSON column. PostgreSQL only.
JSONB                        // JSONB column. PostgreSQL only.

BLOB                         // BLOB (bytea for PostgreSQL)
BLOB('tiny')                 // TINYBLOB (bytea for PostgreSQL. Other options are medium and long)

UUID                         // UUID datatype for PostgreSQL and SQLite, CHAR(36) BINARY for MySQL
(use defaultValue: Sequelize.UUIDV1 or Sequelize.UUIDV4 to make sequelize generate the ids
automatically)
```

Sequelize - models - tipagem

Tipagem de campos utilizados no *sequelize.define()*

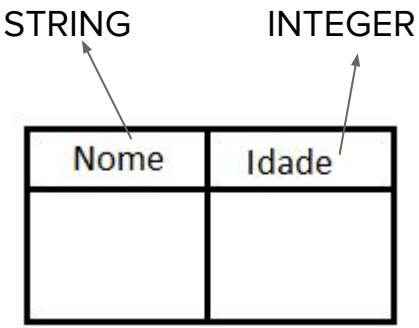
```
RANGE(Sequelize.INTEGER)      // Defines int4range range. PostgreSQL only.
RANGE(Sequelize.BIGINT)       // Defined int8range range. PostgreSQL only.
RANGE(Sequelize.DATE)         // Defines tstzrange range. PostgreSQL only.
RANGE(Sequelize.DATEONLY)     // Defines daterange range. PostgreSQL only.
RANGE(Sequelize.DECIMAL)      // Defines numrange range. PostgreSQL only.

ARRAY(Sequelize.RANGE(Sequelize.DATE))  // Defines array of tstzrange ranges. PostgreSQL only.

GEOMETRY                      // Spatial column.  PostgreSQL (with PostGIS) or MySQL only.
GEOMETRY('POINT')             // Spatial column with geometry type.  PostgreSQL (with PostGIS) or
MySQL only.
GEOMETRY('POINT', 4326)       // Spatial column with geometry type and SRID.  PostgreSQL (with
PostGIS) or MySQL only.
```

Sequelize - models - estrutura

sequelize.define()



Nome da Constante.

Nome da Tabela

Campos

```
const Contato = sequelize.define('contato',{
    nome: {
        type: Sequelize.STRING
    },
    idade: {
        type: Sequelize.INTEGER
    }
});
```

Sequelize - models - estrutura

```
18  const Contato = sequelize.define('contato',{
19      nome: {
20          type: Sequelize.STRING
21      },
22      idade: {
23          type: Sequelize.INTEGER
24      }
25  });
26
27
28  Contato.sync({force: true});
```

Sincroniza e cria o conteúdo da constante “Contato”

proj11 > JS connect.js > ...

```
1   const Sequelize = require('sequelize');
2
3   // inserir dados de conexão: nome BD; usuário; senha; abrir json.
4   const sequelize = new Sequelize('sistemadecadastro','root','12345',{
5    // inserir endereço do SGBD e tipo de SGBD
6    host: "localhost",
7    dialect: 'mysql'
8   })
9
10  // certificando a autenticação
11  // usando promise 'then' e 'catch'
12  sequelize.authenticate().then(function(){
13      console.log("Conectado ao Banco de Dados com sucesso!")
14  }).catch(function(erro){
15      console.log("Falha ao se conectar: " + erro)
16  })
17
18  const Contato = sequelize.define('contato',{
19      nome: {
20          type: Sequelize.STRING
21      },
22      idade: {
23          type: Sequelize.INTEGER
24      }
25  });
26
27
28  Contato.sync({force: true});
```

Sequelize - models - estrutura

```
C:\NodeEstudos\proj11>node connect.js
Executing (default): SELECT 1+1 AS result
Executing (default): DROP TABLE IF EXISTS `contatos`;
Conectado ao Banco de Dados com sucesso!
Executing (default): CREATE TABLE IF NOT EXISTS `contatos` (`id` INTEGER NOT NULL auto_increment , `nome` VARCHAR(255),
`idade` INTEGER, `createdAt` DATETIME NOT NULL, `updatedAt` DATETIME NOT NULL, PRIMARY KEY (`id`)) ENGINE=InnoDB;
Executing (default): SHOW INDEX FROM `contatos`
```

```
mysql> show tables;
+-----------------------------+
| Tables_in_sistemadecadastro |
+-----------------------------+
| contatos                    |
| usuarios                    |
+-----------------------------+
2 rows in set (0.00 sec)
```

```
mysql> describe contatos;
+-----------+--------------+------+-----+---------+----------------+
| Field     | Type         | Null | Key | Default | Extra          |
+-----------+--------------+------+-----+---------+----------------+
| id        | int          | NO   | PRI | NULL    | auto_increment |
| nome      | varchar(255) | YES  |     | NULL    |                |
| idade     | int          | YES  |     | NULL    |                |
| createdAt | datetime     | NO   |     | NULL    |                |
| updatedAt | datetime     | NO   |     | NULL    |                |
+-----------+--------------+------+-----+---------+----------------+
5 rows in set (0.00 sec)
```

Sequelize - models - estrutura

Toda tabela deve ter uma chave e o Sequelize criou a coluna “**id**” automaticamente.

Também foi criado a coluna “**createdAt**” que armazena a data do registro e “**updatedAt**” que armazena a última modificação do registro.

```
mysql> describe contatos;
+-----------+--------------+------+-----+---------+----------------+
| Field     | Type         | Null | Key | Default | Extra          |
+-----------+--------------+------+-----+---------+----------------+
| id        | int          | NO   | PRI | NULL    | auto_increment |
| nome      | varchar(255) | YES  |     | NULL    |                |
| idade     | int          | YES  |     | NULL    |                |
| createdAt | datetime     | NO   |     | NULL    |                |
| updatedAt | datetime     | NO   |     | NULL    |                |
+-----------+--------------+------+-----+---------+----------------+
5 rows in set (0.00 sec)
```

id	Nome	Idade	createdAt	updatedAt

Sequelize - models - estrutura

Assim que “rodar” o arquivo construtor de tabelas recomendo que comente a linha “sync()”, caso contrário, corre-se o risco das tabelas serem novamente criadas e/ou sobrescritas.

```
// Contato.sync({force: true});
```

Sequelize - Inserindo Registro

Para inserir registro basta fazer uso da constante + função “create()”.

```
Contato.create({
    nome: "Celso Masotti",
    idade: 60
});
```

```
C:\NodeEstudos\proj11>node connect.js
Executing (default): SELECT 1+1 AS result
Executing (default): INSERT INTO `contatos` (`id`,`nome`,`idade`,`createdAt`,`updatedAt`) VALUES (DEFAULT,?,?,?,?);
Conectado ao Banco de Dados com sucesso!
```

```
mysql> select * from contatos;
+----+---------------+-------+---------------------+---------------------+
| id | nome          | idade | createdAt           | updatedAt           |
+----+---------------+-------+---------------------+---------------------+
|  1 | Celso Masotti |    60 | 2020-05-13 17:55:29 | 2020-05-13 17:55:29 |
+----+---------------+-------+---------------------+---------------------+
1 row in set (0.00 sec)
```

Obrigado

Prof. Celso Henrique Masotti