Curso de NodeJS

Unidad Didáctica 06: ORM

Ayuntamiento
de Vitoria-Gasteiz
Vitoria-Gasteizko
Udala

http://cursosdedesarrollo.com/

Índice de contenidos

- Introducción
- Drivers
- MongoDB
- ORM Mongoose
- Mysql
- ORM Sequelize
- Conclusiones

Introducción

¿Qué es ORM?

Introducción

ORM es un sistema que permite la interacción con una base de datos y una aplicación mediante un intercambio de objeto y un sistema de mapeo de entidades a tablas

http://cursosdedesarrollo.com/

Drivers

Para poder conectar con los distintos sistemas de bases de datos es necesario manejar los drivers de conexión a dichas bases de datos desde node

MongoDB

Es una base de datos orienta a objetos que almacena la información en almacenes y tiene una estructura jerárquica de objetos

Mongoose

Es una framework de ORM orientado a su uso con Mongo DB

http://cursosdedesarrollo.com/

Mongoose

Se instala con

```
npm install mongoose --save
```

Mongoose

```
var mongoose = require('mongoose');

mongoose.connect('mongodb://localhost/test');

//Get the default connection

var db = mongoose.connection;

//Bind connection to error event (to get notification of
connection errors)

db.on('error', console.error.bind(console, 'MongoDB
connection error:'));
```

Mongoose

Al serán ORM necesita hacer uso de los esquemas par realizar el mapeo con la BBDD

Mongoose

```
//Define a schema

var Schema = mongoose.Schema;


var SomeModelSchema = new Schema({

a_string          : String,

a_date            : Date

});
```

http://cursosdedesarrollo.com/

Mongoose

Después de la definición del esquema es necesario capturar el esquema como un modelo accesible

Mongoose

```
// Define schema

var Schema = mongoose.Schema;


var SomeModelSchema = new Schema({

    a_string: String,

    a_date: Date

});


// Compile model from schema

var SomeModel = mongoose.model('SomeModel', SomeModelSchema );
```

http://cursosdedesarrollo.com/

Mongoose

En el esquema pueden definirse aquellos campos de una manera compleja

Mongoose

```
var schema = new Schema(
{
  name: String,
  binary: Buffer,
  living: Boolean,
  updated: { type: Date, default: Date.now },
  age: { type: Number, min: 18, max: 65, required: true },
  mixed: Schema.Types.Mixed,
  _someId: Schema.Types.ObjectId,
  array: [],
  ofString: [String], // You can also have an array of each of the other types too.
  nested: { stuff: { type: String, lowercase: true, trim: true } }
})
```

http://cursosdedesarrollo.com/

Mongoose

Para utilizar los modelos es necesario manejamos variables por objeto

Mongoose

```
// Create an instance of model SomeModel

var awesome_instance = new SomeModel({ name: 'awesome' });

// Save the new model instance, passing a callback

awesome_instance.save(function (err) {

  if (err) return handleError(err);

  // saved!

});
```

http://cursosdedesarrollo.com/

Mongoose

Podemos crear nuevos objetos

```
SomeModel.create({ name: 'also_awesome' },
function (err, awesome_instance) {

if (err) return handleError(err);

// saved!

});
```

Mongoose

Se pueden hacer búsquedas de objetos

```
// find all athletes who play tennis, selecting the 'name' and
'age' fields

Athlete.find({ 'sport': 'Tennis' }, 'name age', function (err,
athletes) {

if (err) return handleError(err);

// 'athletes' contains the list of athletes that match the criteria.

})
```

http://cursosdedesarrollo.com/

Mongoose

Se pueden manejar las consultas con el objeto Query

```
// find all athletes that play tennis

var query = Athlete.find({ 'sport': 'Tennis' });

// selecting the 'name' and 'age' fields

query.select('name age');

// limit our results to 5 items

query.limit(5);

// sort by age

query.sort({ age: -1 });

// execute the query at a later time

query.exec(function (err, athletes) {

  if (err) return handleError(err);

  // athletes contains an ordered list of 5 athletes who play Tennis

})
```

http://cursosdedesarrollo.com/

Mongoose

```
Athlete.
  find().
  where('sport').equals('Tennis').
  where('age').gt(17).lt(50).  //Additional where query
  limit(5).
  sort({ age: -1 }).
  select('name age').
  exec(callback); // where callback is the name of our callback function
```

http://cursosdedesarrollo.com/

Mongoose

Tenemos también otras funciones interesantes:

findById()

findOne()

findByIdAndRemove()

findByIdAndUpdate()

findOneAndRemove()

findOneAndUpdate()

Mongoose

Las entidades pueden relacionarse unas con contras

http://cursosdedesarrollo.com/

Mongoose

```
var mongoose = require('mongoose')
, Schema = mongoose.Schema
var authorSchema = Schema({
name    : String,
stories : [{ type: Schema.Types.ObjectId, ref: 'Story' }]
});
var storySchema = Schema({
author : { type: Schema.Types.ObjectId, ref: 'Author' },
title    : String,
});
var Story  = mongoose.model('Story', storySchema);
var Author = mongoose.model('Author', authorSchema);
```

Mongoose

```
var bob = new Author({ name: 'Bob Smith' });

bob.save(function (err) {

  if (err) return handleError(err);

  //Bob now exists, so lets create a story

  var story = new Story({

    title: "Bob goes sledding",

    author: bob._id    // assign the _id from the our author Bob. This ID is created by default!

  });

  story.save(function (err) {

    if (err) return handleError(err);

    // Bob now has his story

  });

});
```

http://cursosdedesarrollo.com/

Mongoose

```
Story

.findOne({ title: 'Bob goes sledding' })

.populate('author') //This populates the author id with actual author
information!

.exec(function (err, story) {

  if (err) return handleError(err);

  console.log('The author is %s', story.author.name);

  // prints "The author is Bob Smith"

});
```

http://cursosdedesarrollo.com/

Mysql

Es uno de los motores de bases de datos más conocidos en software libre

http://cursosdedesarrollo.com/

Mysql

Tiene un paquete asociado instalable

```
nom install mysql --save
```

http://cursosdedesarrollo.com/

Sequelize

Es un ORM preparado para funcionar contra bases de datos relacionales: Mysql, Postgresql, MariaDB, Sqlite y MSSQL

Sequelize

Soporta gestión de transacciones, relaciones, replicación de lectura, etc…

http://cursosdedesarrollo.com/

Sequelize

Instalación

```
$ npm install --save sequelize

# y uno de los siguientes dependiendo de la bbdd a conectar

$ npm install --save pg pg-hstore

$ npm install --save mysql // For both mysql and mariadb dialects

$ npm install --save sqlite3

$ npm install --save tedious // MSSQL
```

Sequelize

Gestión de la conexión:

```
var sequelize = new Sequelize('database', 'username', 'password', {
  host: 'localhost',
  dialect: 'mysql'|'mariadb'|'sqlite'|'postgres'|'mssql',
  pool: {
    max: 5,
    min: 0,
    idle: 10000
  },
  // SQLite only
  storage: 'path/to/database.sqlite'
});
// Or you can simply use a connection uri
var sequelize = new Sequelize('postgres://user:pass@example.com:5432/dbname');
```

http://cursosdedesarrollo.com/

Sequelize

Definición de Modelos:

```
var User = sequelize.define('user', {
  firstName: {
    type: Sequelize.STRING,
    field: 'first_name' // Will result in an attribute that is firstName when user facing but first_name in the database
  },
  lastName: {
    type: Sequelize.STRING
  }
}, {
  freezeTableName: true // Model tableName will be the same as the model name
});
```

http://cursosdedesarrollo.com/

Sequelize

Uso de Modelos:

```
User.sync({force: true}).then(function () {
  // Table created
  return User.create({
    firstName: 'John',
    lastName: 'Hancock'
  });
});
```

http://cursosdedesarrollo.com/

Sequelize

Uso de consultas a través de promesas:

```
User.findOne().then(function (user) {
    console.log(user.get('firstName'));
});
```

Sequelize

Tipos de Datos:

```
Sequelize.STRING                    // VARCHAR(255)

Sequelize.STRING(1234)              // VARCHAR(1234)

Sequelize.STRING.BINARY             // VARCHAR
BINARY

Sequelize.TEXT                      // TEXT

Sequelize.TEXT('tiny')              // TINYTEXT
```

Sequelize

Tipos de Datos:

Sequelize.INTEGER	// INTEGER
Sequelize.BIGINT	// BIGINT
Sequelize.BIGINT(11)	// BIGINT(11)

http://cursosdedesarrollo.com/

Sequelize

Tipos de Datos:

```
Sequelize.FLOAT                  // FLOAT

Sequelize.FLOAT(11)              // FLOAT(11)

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

http://cursosdedesarrollo.com/

Sequelize

Tipos de Datos:

Sequelize.REAL // REAL PostgreSQL only.

Sequelize.REAL(11) // REAL(11) PostgreSQL only.

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

http://cursosdedesarrollo.com/

Sequelize

Tipos de Datos:

```
Sequelize.DOUBLE                    // DOUBLE
Sequelize.DOUBLE(11)                  // DOUBLE(11)
Sequelize.DOUBLE(11, 12)              // DOUBLE(11,12)
Sequelize.DECIMAL                   // DECIMAL
Sequelize.DECIMAL(10, 2)              // DECIMAL(10,2)
```

http://cursosdedesarrollo.com/

Sequelize

Tipos de Datos:

Sequelize.DATE // DATETIME for mysql / sqlite, TIMESTAMP WITH TIME ZONE for postgres

Sequelize.DATE(6) // DATETIME(6) for mysql 5.6.4+. Fractional seconds support with up to 6 digits of precision

Sequelize.DATEONLY // DATE without time.

Sequelize.BOOLEAN // TINYINT(1)

http://cursosdedesarrollo.com/

Sequelize

Tipos de Datos:

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

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

Sequelize.JSON   // JSON column. PostgreSQL only.

Sequelize.JSONB   // JSONB column. PostgreSQL only.

Sequelize.BLOB   // BLOB (bytea for PostgreSQL)

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

Sequelize.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)

http://cursosdedesarrollo.com/

Sequelize

Tipos de Datos:

Sequelize.RANGE(Sequelize.INTEGER)   // Defines int4range range. PostgreSQL only.

Sequelize.RANGE(Sequelize.BIGINT)   // Defined int8range range. PostgreSQL only.

Sequelize.RANGE(Sequelize.DATE)   // Defines tstzrange range. PostgreSQL only.

Sequelize.RANGE(Sequelize.DATEONLY)   // Defines daterange range. PostgreSQL only.

Sequelize.RANGE(Sequelize.DECIMAL)   // Defines numrange range. PostgreSQL only.

Sequelize

Tipos de Datos:

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

Sequelize.GEOMETRY // Spatial column. PostgreSQL (with PostGIS) or MySQL only.

Sequelize.GEOMETRY('POINT') // Spatial column with geometry type. PostgreSQL (with PostGIS) or MySQL only.

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

Sequelize

Tipos de Datos:

```
Sequelize.INTEGER.UNSIGNED              // INTEGER UNSIGNED

Sequelize.INTEGER(11).UNSIGNED          // INTEGER(11) UNSIGNED

Sequelize.INTEGER(11).ZEROFILL          // INTEGER(11) ZEROFILL

Sequelize.INTEGER(11).ZEROFILL.UNSIGNED // INTEGER(11) UNSIGNED ZEROFILL

Sequelize.INTEGER(11).UNSIGNED.ZEROFILL // INTEGER(11) UNSIGNED ZEROFILL
```

Sequelize

Validaciones:

http://docs.sequelizejs.com/en/v3/docs/models-definition/#validations

Permite validar distintos tipos dependiendo de la biblioteca validator.js

https://github.com/chriso/validator.js

Sequelize

```
var Pub = Sequelize.define('pub', {
  name: { type: Sequelize.STRING },
  address: { type: Sequelize.STRING },
  latitude: {type: Sequelize.INTEGER, allowNull: true, defaultValue: null, validate: { min: -90, max: 90 } },
  longitude: { type: Sequelize.INTEGER, allowNull: true, defaultValue: null, validate: { min: -180, max: 180 } },
}, {
  validate: {  bothCoordsOrNone: function() {
      if ((this.latitude === null) !== (this.longitude === null)) {
        throw new Error('Require either both latitude and longitude or neither')
      }
    }
  }
})
```

http://cursosdedesarrollo.com/

Sequelize

Consultas:

```
Model.findAll({
 attributes: ['foo', 'bar']
});
```

SELECT foo, bar ...

http://cursosdedesarrollo.com/

Sequelize

Consultas:

```
Model.findAll({

attributes: ['foo', ['bar', 'baz']]

});

SELECT foo, bar AS baz ...
```

Sequelize

```
Model.findAll({

attributes: [[sequelize.fn('COUNT',
sequelize.col('hats')), 'no_hats']]

});
```

```
SELECT COUNT(hats) AS no_hats ...
```

Sequelize

```
Post.findAll({
where: {
authorId: 2
}
});
// SELECT * FROM post WHERE authorId = 2
```

http://cursosdedesarrollo.com/

Sequelize

```
Post.findAll({
where: {
authorId: 12,
status: 'active'
}
});
// SELECT * FROM post WHERE authorId = 12 AND status = 'active';
```

http://cursosdedesarrollo.com/

Sequelize

```
Post.findAll({
where: {
authorId: 12,
status: 'active'
}
});
// SELECT * FROM post WHERE authorId = 12 AND status = 'active';
```

http://cursosdedesarrollo.com/

Sequelize

```
Post.findAll({

where: sequelize.where(sequelize.fn('char_length',
sequelize.col('status')), 6)

});

// SELECT * FROM post WHERE char_length(status)
= 6;
```

Sequelize

```
Post.update({
updatedAt: null,
}, {
where: {
deletedAt: {
$ne: null
}
}
});
// UPDATE post SET updatedAt = null WHERE deletedAt NOT NULL;
```

http://cursosdedesarrollo.com/

Sequelize

```
Post.destroy({
where: {
status: 'inactive'
}
});
// DELETE FROM post WHERE status = 'inactive';
```

http://cursosdedesarrollo.com/

Sequelize

Operadores:

```
$and: {a: 5}            // AND (a = 5)
$or: [{a: 5}, {a: 6}]   // (a = 5 OR a = 6)
$gt: 6,                 // > 6
$gte: 6,                // >= 6
$lt: 10,                // < 10
$lte: 10,               // <= 10
$ne: 20,                // != 20
```

http://cursosdedesarrollo.com/

Sequelize

Operadores:

```
$not: true,            // IS NOT TRUE
$between: [6, 10],     // BETWEEN 6 AND 10
$notBetween: [11, 15], // NOT BETWEEN 11 AND 15
$in: [1, 2],           // IN [1, 2]
$notIn: [1, 2],        // NOT IN [1, 2]
$like: '%hat',         // LIKE '%hat'
$notLike: '%hat'       // NOT LIKE '%hat'
$iLike: '%hat'         // ILIKE '%hat' (case insensitive) (PG only)
$notILike: '%hat'      // NOT ILIKE '%hat'  (PG only)
$like: { $any: ['cat', 'hat']}
// LIKE ANY ARRAY['cat', 'hat'] - also works for iLike and notLike
```

http://cursosdedesarrollo.com/

Sequelize

Operadores:

$overlap: [1, 2]	// && [1, 2] (PG array overlap operator)
$contains: [1, 2]	// @> [1, 2] (PG array contains operator)
$contained: [1, 2]	// <@ [1, 2] (PG array contained by operator)
$any: [2,3]	// ANY ARRAY[2, 3]::INTEGER (PG only)

$col: 'user.organization_id' // = "user"."organization_id", with dialect specific column identifiers, PG in this example

http://cursosdedesarrollo.com/

Sequelize

Paginación:

```
// Fetch 10 instances/rows

Project.findAll({ limit: 10 })


// Skip 8 instances/rows

Project.findAll({ offset: 8 })


// Skip 5 instances and fetch the 5 after that

Project.findAll({ offset: 5, limit: 5 })
```

http://cursosdedesarrollo.com/

Sequelize

Ordenación:

```
something.findOne({

order: 'username DESC'

})
```

http://cursosdedesarrollo.com/

Conclusiones

http://cursosdedesarrollo.com/

Datos de Contacto

http://www.cursosdedesarrollo.com
david@cursosdedesarrollo.com

http://cursosdedesarrollo.com/

Licencia

David Vaquero Santiago

http://cursosdedesarrollo.com/