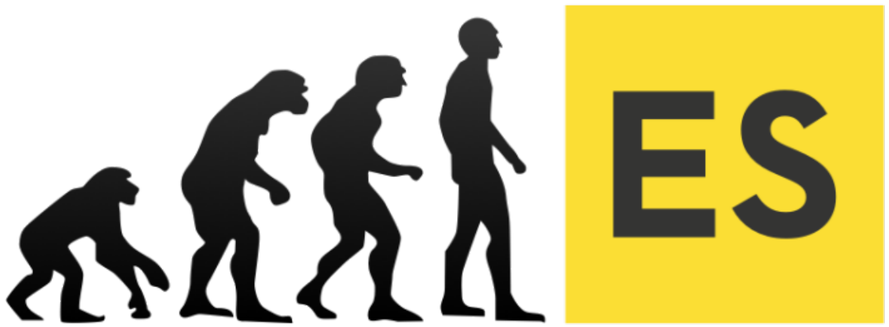
# Promesas
# Bases de datos
# API

Javier Miguel @JavierMiguelG jamg44@gmail.com CTO & Freelance Developer




# Evolución de la asincronía

# Promesas

# Callback Hell

```
// esto no es vida
carga(function(err, valor0) {
  valida(valor0, function(err, valor1) {
    procesa(valor1, function(err, valor2) {
      guarda(valor2, function(err, valor3) {
        comprueba(valor3, function(err, valor4) {
          junta(valor4, function(err, valor5) {
            limpia(valor5, function(err, valor6) {
              avisa(valor6, function(err, valor6) {
                res.json({piramide: true});
              });
            });
          });
        });
      });
    });
  });
});
```

```
174
175         async.series([
176             function(callback) {
177                 if (todos == "true") {
178                     models.PromocionesBase.contarTodos(todos, functi
179                         flag = 1;
180                         resultado = result;
181                         procesarSegmento(segmento_id, function(segme
182                             var id = parseInt(segmento_id);
183                             models.segmentacion.find(id).success(fun
184                                 if (segmento) {
185                                     var obj = {
186                                         cp: segmento.cp,
187                                         edad: segmento.edad,
188                                         sexo: segmento.sexo,
189                                         hijo: segmento.hijo,
190                                         favoritos: segmento.favorito
191                                     };
192                                     models.PromocionesBase.contarSim
193                                         if (err) return callback(err
194                                         return callback(null, result
195                                     }
196                                 }
197                             });
198                         }
199                     });
200                 }
201             },
```

# Promesas

**Una promesa es un objeto que representa una operación que aún no se ha completado, pero que se completará más adelante.**

Antes de ES2015 podíamos usarlas con librerías, pero estas librerías tienen ligeras (o no tan ligeras) diferencias entre ellas. Ahora ya forman parte del estándar y el lenguaje y no necesitamos estas librerías.

# Promesas

Tiene tres estados posibles (https://promisesaplus.com/)

1. Pending
2. Fullfilled(value)
3. Rejected(reason)

## 2.1. Promise States

A promise must be in one of three states: pending, fulfilled, or rejected.

2.1.1. When pending, a promise:

- 2.1.1.1. may transition to either the fulfilled or rejected state.

2.1.2. When fulfilled, a promise:

- 2.1.2.1. must not transition to any other state.
- 2.1.2.2. must have a value, which must not change.

2.1.3. When rejected, a promise:

- 2.1.3.1. must not transition to any other state.
- 2.1.3.2. must have a reason, which must not change.

# Promesas

Cuando una promesa está en uno de los dos estados fulfilled o rejected se le llama settled.

Si la promesa se hubiera cumplimentado (fulfilled) o rechazado (rejected) antes de asignarle un then o catch, cuando se le asignen serán llamados con el resultado o el error.

# Promesas

## Como se hace

```
var promesa = new Promise(function(resolve, reject) {
  // llamo a resolve con el resultado
  // o llamo a reject con el error
});


promesa.then( function(resultado) {

}).catch( function(error) {

});
```

# Promesas

```
promesa.then( function(resultado) {

}).catch( function(error) {

});
```

Es simplemente azúcar sintáctico para la forma:

```
promesa.then(
  function(resultado) { },
  function(error) { }
);
```

# Ejercicio

Hacer el comando sleep(milisegundos)

# Promesas

Podemos encadenar promesas.

```
promesa1
.then(() => promesa2)
.then(() => promesa3)
.then( function(data) { // final
    console.log(data); })
.catch( function(err) {
    console.log('ERROR', err);
});
```

# Promesas

```
var ingredientes = ['sal','pimienta','conejo','gambas'];

// echar() recibe un string y retorna una promesa
var promisedTexts = ingredientes.map(echar);

Promise.all(promisedTexts)
    .then(function (texts) {
       console.log(texts); // han acabado todas
    })
    .catch(function (reason) {
        // llegaremos aqui con el primero que falle
    });
```

# Promesas

Si Promise.all esperaba a que estuvieran todas cumplidas, Promise.race lo hace cuando cumpla la primera, devolviendo su resultado.

```
Promise.race([p1, p2, p3])
    .then(function (textoDelMasRapido) {
        // llegaremos aqui con el primero que acabe
    })
    .catch(function (reason) {
        // llegaremos aqui con el primero que falle
    });
```

# Promesas

El objeto Promise tiene también un par de métodos estáticos que pueden ser útiles:

```
Promise.resolve(valor);
```
Devuelve una promesa resuelta con el valor proporcionado.

```
Promise.reject(razon);
```
Devuelve una promesa resuelta con la razón suministrada. La razón debería ser un error (generalmente una instancia de objeto Error).

# Promesas

```
Promise.resolve("bien!").then(function(value) {
  console.log(value); // "Prueba resolve"
}, function(reason) {
  // not called
});

Promise.reject(new Error("chungo...")).then(function(value) {
  // not called
}, function(error) {
  console.log(error); // Stacktrace
});
```

# async / await

# ECMAScript 2015 (ES6) - async / await

**async** hace que una función devuelva una promesa.

# ECMAScript 2015 (ES6) - async / await

```
async function saluda() {
    return 'hola';
}


console.log(saluda());  // Promise { 'hola' }


saluda().then(res => console.log(res)); // hola
```

# ECMAScript 2015 (ES6) - async / await

**await** consume una promesa

# ECMAScript 2015 (ES6) - async / await

```
async function saluda() {
    const nombre = await row.findName();
    return nombre;
}
```

ejemplos/async-await

# ECMAScript 2015 (ES6) - async / await

```
async function bucleAsincronoEnSerie() {
    for (let i = 0; i < 5; i++) {
      const info = await row.findNext();
      console.log(info.name);
    }
}
```

# ECMAScript 2015 (ES6) - async / await

```
async function asincronoEnParalelo() {
    const prom1 = row.findNext();
    const prom2 = row.findNext();
    const prom3 = row.findNext();
    const list = await Promise.all([prom1, prom2, prom3]);
}
```

# ECMAScript 2015 (ES6) - async / await

## Como usarlo en Express

# ECMAScript 2015 (ES6) - async await + Express

```
router.get(async (req, res, next) => {
    const list = await Orders.find(…);
    // si falla quién gestiona el error?*
})
```

* una pista… nadie

# ECMAScript 2015 (ES6) - async await + Express

```
router.get(async (req, res, next) => {
  try {
    const list = await Orders.find(…);
  } catch (err) {
    next(err);
  }
})
```

# ECMAScript 2015 (ES6) - async await + Express

OPCIONAL

https://github.com/Abazhenov/express-async-handler

```
$ npm install express-async-handler
```

# ECMAScript 2015 (ES6) - async await + Express

https://github.com/Abazhenov/express-async-handler

```
const asyncHandler = require('express-async-handler')

router.get(asyncHandler(async (req, res, next) => {
 const list = await Orders.find(…);
}))
```

# ECMAScript 2015 (ES6) - async await + Express

https://github.com/davidbanham/express-async-errors

```
const express = require('express');
require('express-async-errors');

app.get('/users', async (req, res) => {
  const users = await User.findAll();
  res.send(users);
});
```

# Bases de datos

# Bases de datos

Node.js, a través de módulos de terceros, se puede conectar casi con cualquier base de datos del mercado.

Basta con cargar el driver (módulo) adecuado y establecer la conexión.

```
$ npm install mysql
```

```
$ npm install mongoskin
```

# SQL

# Bases de datos - MySQL (docker)

## Desplegar BD MySQL local con Docker:

```
docker run -d --name mariadb-kc \
  -v $(PWD)/data:/var/lib/mysql \
  -e MARIADB_ROOT_PASSWORD=my-secret-pw \
  -p 3306:3306 \
  mariadb:latest
```

## Abrir shell al contenedor:

```
docker exec -it mariadb-kc bash

mariadb -p
```

# Bases de datos - MySQL

## Inicializar BD MySQL:

```
CREATE DATABASE cursonode;

SHOW DATABASES;

use cursonode;

CREATE TABLE agentes (agenteid int NOT NULL AUTO_INCREMENT, name varchar(255) NOT NULL,
age int, PRIMARY KEY (agenteid) );
INSERT INTO agentes (name, age) VALUES ('Brown', 21);
INSERT INTO agentes (name, age) VALUES ('Smith', 33);
INSERT INTO agentes (name, age) VALUES ('Jones', 45);
```

# Bases de datos - MySQL

## Conectar con MySQL:

```
var mysql      = require('mysql');
var connection = mysql.createConnection({
   host     : 'localhost',
   user     : 'usuario',
   password : 'pass',
   database : 'cursonode'
});

connection.connect(); // callback opcional

connection.query('SELECT * from agentes', function(err, rows, fields) {
   if (err) throw err;
   console.log(rows);
});
```

# Bases de datos

Para refrescar conceptos de SQL:

https://www.sqlteaching.com/

# Bases de datos - SQL ORMs

Un ORM (Object Relational Mapping) se encarga principalmente de:

- Convertir objetos en consultas SQL para que puedan ser persistidos en una base de datos relacional.

- Traducir los resultados de una consulta SQL y generar objetos.

Esto nos resultará util si el diseño de nuestra aplicación es orientado a objetos (OOP).

# Bases de datos - SQL ORMs

ORMs usados para bases de datos SQL:

- TypeORM https://github.com/typeorm/typeorm
- Prisma https://www.prisma.io/
- Sequelize http://docs.sequelizejs.com/en/latest/
- Mikro-ORM https://github.com/mikro-orm/mikro-orm

Otras alternativas son Knex y Bookshelf.

# Bases de datos - MongoDB

MongoDB en una base de datos no relacional sin esquemas, esto significa principalmente que:

- No tenemos JOIN, tendremos que hacerlo nosotros
- Cada registro podría tener una estructura distinta
- Mínimo soporte a transacciones

A la hora de decidir que base de datos usar para una aplicación debemos pensar como vamos a organizar los datos para saber si nos conviene usar una base de datos relacional o no relacional.

# Bases de datos - MongoDB

Usar una base de datos como MongoDB puede darnos más rendimiento principalmente por alguna de estas razones:

- No tiene que gestionar transacciones
- No tiene que gestionar relaciones
- No es necesario convertir objetos a tablas y tablas a objetos (Object-relation Impedance Mismatch)

# Bases de datos - MongoDB

```
$ npm install mongodb
```

```
const { MongoClient } = require('mongodb');

MongoClient.connect('mongodb://localhost', function(err, client) {
  if (err) throw err;
  const db = client.db('cursonode');
  db.collection('agentes').find({}).toArray(function(err, docs) {
    if (err) throw err;
    console.dir(docs);
    client.close();
  });
});
```

# Bases de datos - MongoDB shell basics

Para acceder a la shell usaremos:

```
~/master/cursonode/mongodb-server/bin/mongo
MongoDB shell version: 3.0.4
connecting to: test
>
```

# Bases de datos - MongoDB shell basics

```
show dbs
use <dbname>
show collections
show users
db.agentes.find().pretty()
db.agentes.insert({name: "Brown", age: 37})
db.agentes.remove({_id: ObjectId("55ead8899123383864857 0dd")})
db.agentes.update({_id: ObjectId("55eadb4191233838648570de")}, {$set: {age: 38}})
——————————————————————————————————————————— cuidado con el $set! ——
```

```
db.coleccion.drop()
db.agentes.createIndex({name:1, age:-1})
db.agentes.getIndexes()
```

Mas operaciones en la referencia rápida a la shell de MongoDB

# Bases de datos - MongoDB queries

```
db.agentes.find({ name : 'Smith'})
db.agentes.find({ _id : ObjectId("55eadb41912338386485 70de")})
db.agentes.find({ age: { $gt: 30}}) // $lt, $gte, $lte, ...
db.agentes.find({ age: { $gt: 30, $lt: 40}});
db.agentes.find({ name: { $in: [ 'Jones', 'Brown']}}) //$nin
db.agentes.find({ name: 'Smith', $or: [
  { age: { $lt: 30}},
  { age: 43 } // 'Smith' and ( age < 30 or age = 43)
] })
```

# Bases de datos - MongoDB queries

```
// subdocuments
db.agentes.find({ 'producer.company': 'ACME'})

// arrays
db.agentes.find({ bytes: [ 5, 8, 9 ]}) // array exact
db.agentes.find({ bytes: 5}) // array contain
db.agentes.find({ 'bytes.0': 5}) // array position
```

http://docs.mongodb.org/manual/reference/method/db.collection.find/#db.collection.find

http://docs.mongodb.org/manual/tutorial/query-documents/

# Bases de datos - MongoDB queries

Ordenar:

```
db.agentes.find().sort({age: -1})
```

Descartar resultados:

```
db.agentes.find().skip(1).limit(1)
db.agentes.findOne({name: 'Brown'}) // igual a limit(1)
```

Contar:

```
db.agentes.find().count() // db.agentes.count()
```

# Bases de datos - MongoDB queries

## Full Text Search

Crear índice por los campos de texto involucrados:

```
db.agentes.createIndex({titulo: 'text', subtitulo: 'text'});
```

Para hacer la búsqueda usar:

```
db.agentes.find({$text:{$search:'smith jones'});
```

# Bases de datos - MongoDB queries

## Full Text Search

Frase exacta:

```
db.agentes.find({$text:{$search:'smith jones "el elegido"'}});
```

Excluir un término:

```
db.agentes.find({$text:{$search:'smith jones -mister'});
```

Más info:

https://docs.mongodb.com/v3.2/text-search/

https://docs.mongodb.com/v3.2/tutorial/specify-language-for-text-index/

# Bases de datos - MongoDB Geo

https://docs.mongodb.com/manual/applications/geospatial-indexes/

```
db.productos.createIndex({ubicacion: '2dsphere'})
```

```
db.productos.insert({
  “ubicacion": {
    "coordinates":[ -73.856077, 40.848447 ],
    "type": "Point"
  }
})
```

# Bases de datos - MongoDB Geo

https://docs.mongodb.com/manual/applications/geospatial-indexes/

```
db.productos.createIndex({ubicacion: '2dsphere'})


db.productos.insert({
   “ubicacion": {
      "coordinates":[ -73.856077, 40.848447 ],
      "type": "Point"
   }
})
```

Es necesario crear un índice geoespacial

# Bases de datos - MongoDB Geo

https://docs.mongodb.com/manual/applications/geospatial-indexes/

```
db.productos.createIndex({ubicacion: '2dsphere'})
```

```
db.productos.insert({
   “ubicacion": {
     "coordinates":[ -73.856077, 40.848447 ],
     "type": "Point"
   }
})
```

El orden es longitud, latitud

# Bases de datos - MongoDB Geo

```
const meters = parseFloat(req.params.meters); // 105 * 1000
const longitude = parseFloat(req.params.lng); // -73
const latitude = parseFloat(req.params.lat); // 40

db.productos.find({
  ubicacion: {
    $nearSphere: {
      $geometry: {
        type: 'Point',
        coordinates: [longitude, latitude]
      },
      $maxDistance: meters
    }
  }
})
```

# Bases de datos - MongoDB transacción

`findAndModify` es una operación atómica.

```
db.agentes.findAndModify({
     query: { name: "Brown"},
     update: { $inc: { age: 1}}
})
```

Lo busca y si lo encuentra lo modifica, no permitiendo que otro lo cambie antes de modificarlo.

# Bases de datos - MongoDB transacción

A partir de la versión 4 hay mejor soporte a transacciones (distributed & multi-document).

https://www.mongodb.com/docs/manual/core/transactions/

Paper detallado: https://webassets.mongodb.com/MongoDB_Multi_Doc_Transactions.pdf

# mongoose

# Mongoose

Mongoose es una herramienta que nos permite persistir objetos en MongoDB, recuperarlos y mantener esquemas de estos fácilmente.

Este tipo de herramientas suelen denominarse ODM (Object Document Mapper).

# Mongoose

Instalación como siempre:

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

# Mongoose

Conectar a la base de datos:

```
var mongoose = require('mongoose');
var conn = mongoose.connection;

conn.on('error', (err) =>
  console.error('mongodb connection error', err) );
conn.once('open', () =>
  console.info('Connected to mongodb.') );

mongoose.connect('mongodb://localhost/diccionario');
```

# Mongoose

Crear un modelo:

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

var agenteSchema = mongoose.Schema({
    name: String,
    age: Number
});

mongoose.model('Agente', agenteSchema);
```

# Mongoose

Guardar un registro:

```
var agente = new Agente({name: 'Smith', age: 43});

agente.save(function (err, agenteCreado) {
    if (err) throw err;
    console.log('Agente ' + agenteCreado.name + ' creado');
});
```

# Mongoose

Eliminar registros:

```
Agente.deleteMany({ [filters] }, function(err) {
    if (err) return cb(err);
    cb(null);
});
```

# Mongoose

Crear un método estático a un modelo:

```
agenteSchema.statics.deleteAll = function(cb) {
    Agente.remove({}, function(err) {
        if (err) return cb(err);
        cb(null);
    });
};
```

# Mongoose

Crear un método de instancia a un modelo:

```
agenteSchema.methods.findSimilarAges = function (cb) {
  return this.model('Agente').find({ age: this.age }, cb);
}
```

# Mongoose

Listando registros:

```
agenteSchema.statics.list = function(cb) {
  var query = Agente.find({});
  query.sort('name');
  query.skip(500);
  query.limit(100);
  query.select('name age');
  return query.exec(function(err, rows) {
      if (err) { return cb(err);}
      return cb(null, rows);
  });
});
```

# Mongoose

Crear un modelo con datos geoespaciales:

```
var productoSchema = mongoose.Schema({
    name: String,
    location: {
        type: { type: String},
        coordinates: [Number]
    }
});

schema.index({location: '2dsphere'});

mongoose.model('Producto', productoSchema);
```

# Mongoose

Buscar por cercanía:

```
Product.find({ location: {
   $nearSphere: {
      $geometry: {type: 'Point', coordinates: [longitude, latitude]},
      $maxDistance: meters
   }
}});
```

# Consumir APIs de terceros

# Consumir APIs

Uno de los módulos más usados para esto es *request*.

https://github.com/request/request

npm install request --save

# Consumir APIs

```
var options = {
  method: 'GET',
  url: 'https://api.punkapi.com/v2/beers/random',
  //headers: {'User-Agent': '...'},
  json: true
};

request(options, function (err, response, body) {
  if (err || response.statusCode >= 400) {
    console.error(err, response.statusCode);
    return;
  }
 // body tendrá nuestro contenido
});
```

# Documentación

# OpenAPI

Especificación
https://swagger.io/specification/

Editor
https://swagger.io/tools/swagger-editor/

Visor de doc
https://swagger.io/tools/swagger-ui/

# OpenAPI en Express

Generar especificación desde código o YAML

- https://github.com/Surnet/swagger-jsdoc

Middleware de Express con Swagger UI

- https://github.com/scottie1984/swagger-ui-express

# OpenAPI

Alternativa interesante como visor de doc OpenAPI

- https://github.com/Redocly/redoc