

Servidor API REST con Node.js

Adolfo Sanz De Diego

Octubre 2013

Contents

1	Acerca de	7
1.1	El GUL	7
1.2	¿Dónde encontrarnos?	7
1.3	Adolfo Sanz De Diego	7
1.4	Hackalover	8
1.5	Tweets Sentiment	9
1.6	¿Donde encontrarme?	9
1.7	Créditos	10
1.8	Licencia	10
1.9	Fuentes	10
2	APIs ¿Para qué?	11
2.1	Aplicación estándar	11
2.2	Introducimos API	12
2.3	Separación Roles	13
2.4	¿Y ahora qué?	14
2.5	Servicios externos	15
2.6	Apps clientes	16
2.7	Apps de servicios	17
2.8	Apps mixtas	18
2.9	Plataforma	19
2.10	¿Quien expone APIs?	20
2.11	¿Quien expone APIs?	21

2.12	¿Quién expone APIs?	22
2.13	Exponlas tú	23
2.14	Exponlas tú	24
2.15	Exponlas tú	25
3	APIs RESTful	27
3.1	¿Qué es REST?	27
3.2	Ejemplo API	27
3.3	Manejo de errores	28
3.4	¿Por qué REST?	28
3.5	REST vs RESTful	28
3.6	REST vs RESTful	29
4	Node.js	31
4.1	Introducción	31
4.2	Ejecución	31
4.3	¿Dónde usarlo?	31
4.4	Otros conceptos	32
4.5	Primero pasos	32
4.6	Aburrido	33
5	Código	35
5.1	Aplausos	35
5.2	package.json	35
5.3	npm install	35
5.4	app.js	35
5.5	app.js	36
5.6	Directorios	36
5.7	models	37
5.8	models/talkModel.js	37
5.9	routes	37
5.10	routes/talkRoute.js	37
5.11	services	38
5.12	services/talkService.js	38

<i>CONTENTS</i>	5
6 Demo	41
7 ¿Alguna pregunta?	43

Chapter 1

Acerca de

1.1 El GUL

El GUL es el **Grupo de Usuarios de Linux de la UC3M**.

Grupo de personas con inquietudes en torno a la informática.

Con la idea común de la utilización y promoción del **Software Libre**.

Quedamos de vez en cuando y organizamos actividades sobre todo esto.

El punto de unión es la **lista de correo** que está abierta a todo el mundo.

1.2 ¿Dónde encontrarnos?

Twitter: <http://twitter.com/guluc3m>

Lista: gul@gul.uc3m.es

Ftp: <ftp://ftp.gul.uc3m.es>

Web: <http://www.gul.uc3m.es>

Podcast: <http://holamundo.gul.es/>

Blog: <http://planeta.gul.uc3m.es/>

Linkedin: <http://www.linkedin.com/groups?gid=3451836>

1.3 Adolfo Sanz De Diego

Antiguo programador web JEE

Hoy en día:

- **Profesor de FP de informática:**
 - Hardware, Sistemas Operativos
 - Redes, Programación
- **Formador Freelance:**
 - Java, Android
 - JavaScript, jQuery
 - JSF, Spring, Hibernate
 - Groovy & Grails
- **Me gusta programar**

1.4 Hackalover



Para los amantes de los hackathones

- **Meetup:** <http://www.meetup.com/Hackathon-Lovers/>
- **Twitter:** <http://twitter.com/HackathonLovers>
- **Blog:** <http://hackathonlovers.tumblr.com/>
- **LinkedIn:** <http://www.linkedin.com/groups/Hackathon-Lovers-6510465>
- **YouTube:** <http://www.youtube.com/channel/UCRwSe7jK-y62BMvIiNBV1qw>

1.5 Tweets Sentiment



Es un **analizador de tweets** que extrae información semántica para conocer si el sentimiento general de los tweets de un determinado tema es positivo o negativo.

- Web: <http://tweetssentiment.com/>
- Twitter: <http://twitter.com/TweetsSentiment>

1.6 ¿Donde encontrarme?

Mi nick: **asanzdiego**

- AboutMe: <http://about.me/asanzdiego>
- GitHub: <http://github.com/asanzdiego>
- Twitter: <http://twitter.com/asanzdiego>
- Blog: <http://asanzdiego.blogspot.com.es>
- LinkedIn: <http://www.linkedin.com/in/asanzdiego>
- Google+: <http://plus.google.com/+AdolfoSanzDeDiego>

1.7 Créditos

Agradecimientos a **Carlos Azustre** (<http://twitter.com/carlosazaustre>)

- Cómo crear una API REST usando Node.JS
- <http://carlosazaustre.es/blog/como-crear-una-api-rest-usando-node-js/>

Estas **transparencias** están hechas con:

- <https://github.com/asanzdiego/markdownslides>

1.8 Licencia

Estas **transparencias** están bajo una licencia:

- [Creative Commons Reconocimiento-CompartirIgual 3.0](#)

El **código** de los programas están bajo una licencia:

- [GPL 3.0](#)

1.9 Fuentes

Transparencias:

- [SlideShare](#)
- [Deck Slides](#)
- [Reveal Slides](#)
- [Plain HTML](#)

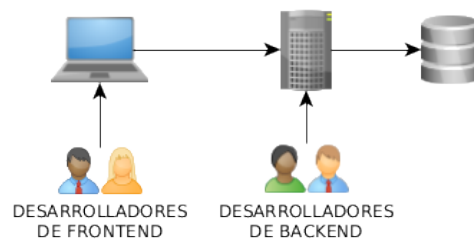
Código:

- <https://github.com/asanzdiego/curso-api-restful-nodejs-server-2013/tree/master/src>

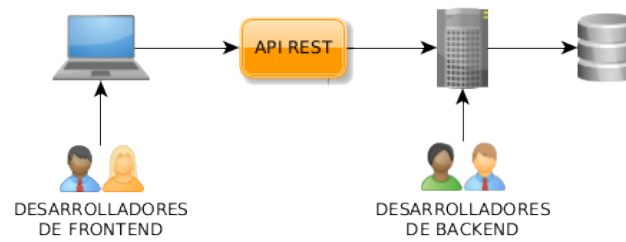
Chapter 2

APIs ¿Para qué?

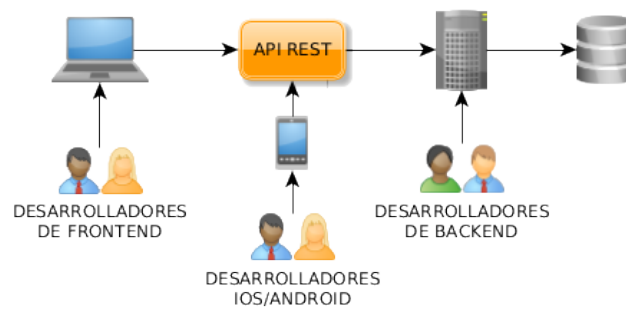
2.1 Aplicación estándar



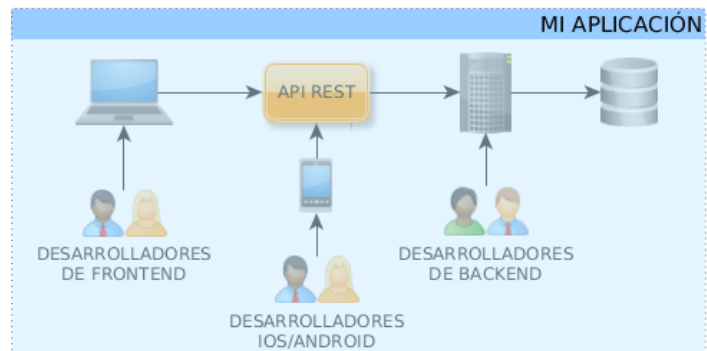
2.2 Introducimos API



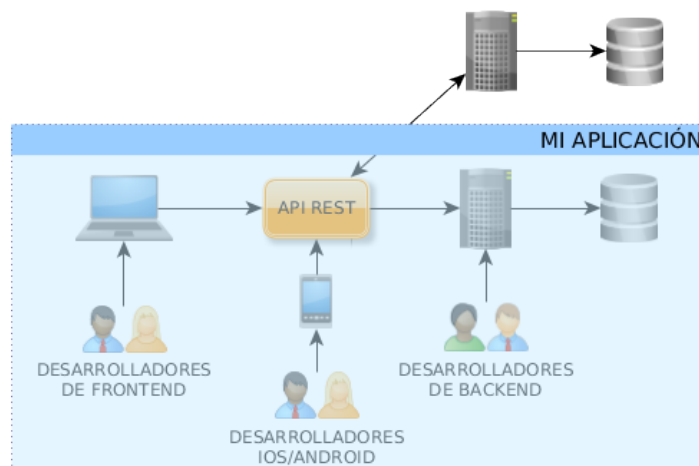
2.3 Separación Roles



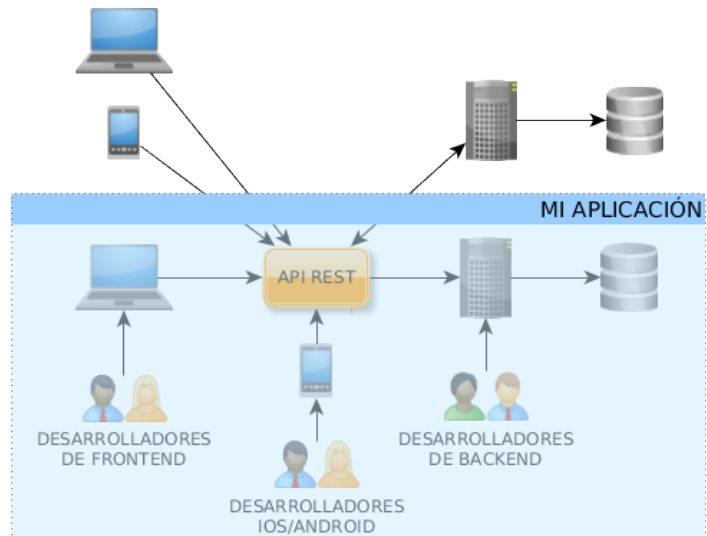
2.4 ¿Y ahora qué?



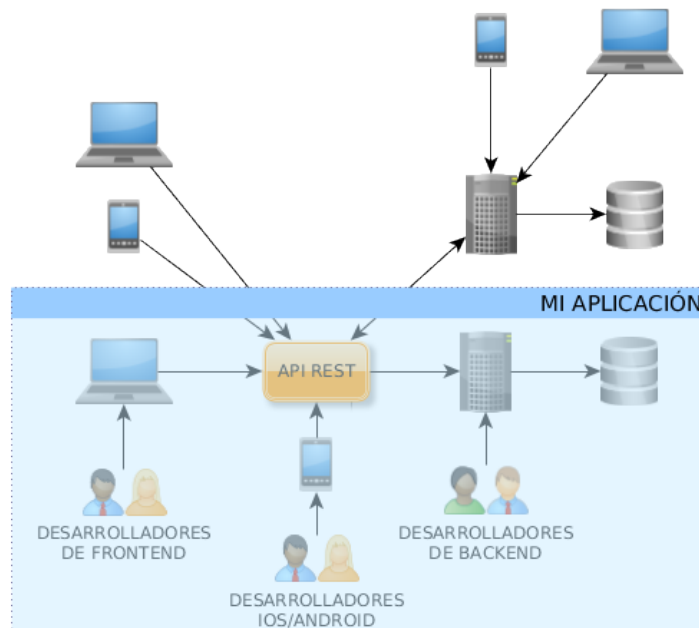
2.5 Servicios externos



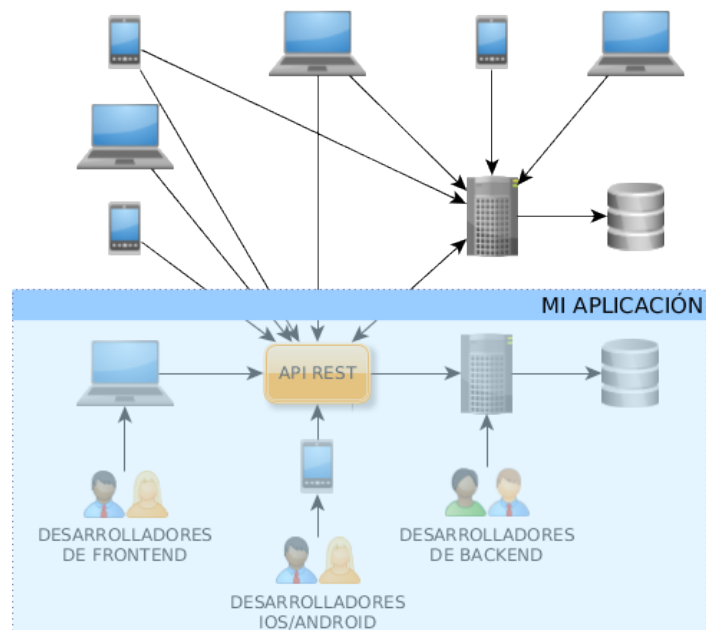
2.6 Apps clientes



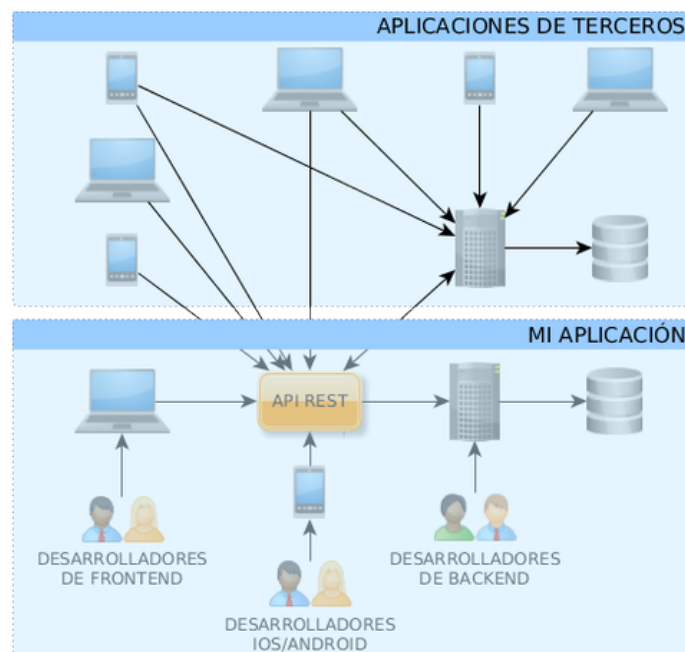
2.7 Apps de servicios



2.8 Apps mixtas



2.9 Plataforma



2.10 ¿Quien expone APIs?



2.11 ¿Quien expone APIs?



2.12 ¿Quien expone APIs?



2.13 Exponlas tú



2.14 Exponlas tú



2.15 Exponlas tú



Chapter 3

APIs RESTful

3.1 ¿Qué es REST?

REST (**R**epresentational **S**tate **T**ransfer) es una técnica de arquitectura de software para sistemas hipertexto distribuidos como la World Wide Web.

En REST una **URL** (Uniform Resource Locator) representa un **recurso**.

Se puede acceder al recurso o modificarlo mediante los **métodos del protocolo HTTP**:

GET, POST, PUT, DELETE

3.2 Ejemplo API

`http://myhost.com/talk`

- GET > Devuelve todas las charlas.
- POST > Crear una nueva charla.

`http://myhost.com/talk/123`

- GET > Devuelve la charla con id=123
- PUT > Actualiza la charla con id=123
- DELETE > Borra la charla con id=123

3.3 Manejo de errores

Se pueden utilizar los errores del protocolo HTTP:

- 200 Successful
- 201 Created
- 202 Accepted
- 301 Moved Permanently
- 400 Bad Request
- 401 Unauthorised
- 402 Payment Required
- 403 Forbidden
- 404 Not Found
- 405 Method Not Allowed
- 500 Internal Server Error
- 501 Not Implemented

3.4 ¿Por qué REST?

Es **más sencillo** (tanto la API como la implementación).

Es **más rápido** (peticiones más ligeras que se pueden cachear).

Es **multiformato** (HTML, XML, JSON, etc.).

Se complementa muy bien con **AJAX**.

3.5 REST vs RESTful

REST se refiere a un tipo de arquitectura de software

- Se utiliza como **nombre**
- Se utiliza como por ejemplo: success = éxito.

Si un servicio web es **RESTful** indica que implementa dicha arquitectura.

- Se utiliza como **adjetivo**
- Se utiliza como por ejemplo: success**ful** = exitoso).

3.6 REST vs RESTful

A veces el **ful** se confunde con **full** = completo.

- Y se refiere a los servicios web REST**full**

Aquellos que implementan una API con todos los métodos del protocolo HTTP.

- Y se refiere a los servicios web REST (**sin el full**)

Aquellos que NO implementan una API con todos los métodos del protocolo HTTP.

Chapter 4

Node.js

4.1 Introducción



Node.js permite programar en **Javascript** del lado del servidor.

Pensado para un manejo de E/S orientada a eventos.

4.2 Ejecución

Ejecución **concurrente**

- Muchos tareas

Pero **NO** paralelo

- Una única hebra

4.3 ¿Dónde usarlo?

Cuando hay **mucha E/S**

- y por tanto mucha CPU inactiva por tarea.

Y hay **muchos clientes**

- que compensan esa inactividad de la CPU.

Pensado para **la creación de programas de red altamente escalables**.

4.4 Otros conceptos

npm <http://npmjs.org/>:

- es el gestor de paquetes de Node.js.

expressjs <http://expressjs.com/>:

- es una librería para Node.js de desarrollo web.

mongoosejs <http://mongoosejs.com/>:

- es una librería para Node.js de modelado de objetos de MongoDB <http://www.mongodb.org/>

4.5 Primero pasos

Instalar **Node.js**

- <http://nodejs.org/download/>

Instalar **MongoDB**

- <http://docs.mongodb.org/manual/installation/>

4.6 Aburrido



Chapter 5

Código

5.1 Aplausos

5.2 package.json

Define las **dependencias** de nuestro proyecto.

```
{
  "name" : "api-restful-nodejs-server",
  "version" : "0.0.1",
  "dependencies" : {
    "express" : "3.x",
    "mongoose" : "3.6.20"
  }
}
```

5.3 npm install

Este comando instalará en la **carpeta node_modules** las dependencias de nuestro proyecto.

```
npm install
```

5.4 app.js

Es el fichero principal.

El nombre es lo de menos.

A veces también se le suele llamar `server.js`

Para ejecutar una aplicación de Node.js:

```
node app.js
```

5.5 app.js

```
// modulos requeridos
var http, express, mongoose, app, server ...

// configuramos app
app.configure(function () {
  // config...
});

// importamos las rutas
var routes = require('./routes/talkRoute')(app);

// conectamos con la base de datos
mongoose.connect('mongodb://localhost/gul', function(err, res) {
  // console.log('Connected to GUL MongoDB Database');
});

// arrancamos el servidor
server.listen(3000, function() {
  // console.log("Server running on http://localhost:3000");
});
```

5.6 Directorios

Puedes usar la estructura de directorios que quieras.

Yo he usado esta:

- **models:**
- **routes:**
- **services:**

5.7 models

Directorio con los modelos que se van a guardar en base de datos.

Yo creo un fichero js para cada colección.

5.8 models/talkModel.js

```
// modulos requeridos
var mongoose = require('mongoose');
var Schema   = mongoose.Schema;

// definimos el modelo 'talk' con sus restricciones
// también podemos definir relaciones con otros modelos
// aquí no se ha hecho por simplificar
var talkSchema = new Schema({
  talkName:      { type: String, required: true },
  talkDate:      { type: Date,   required: true },
  talkSpeaker:   { type: String, required: true },
  talkSpeakerMail: { type: String, required: true,
    match: /^\\w+@[a-zA-Z_]+?\\.[a-zA-Z]{2,3}$/ },
  talkPoints:    { type: Number, required: true, default: 0 }
});

// exportamos el modelo
module.exports = mongoose.model('Talk', talkSchema);
```

5.9 routes

Directorio con los mapeos de las rutas de la API RESTful.

Aquí sólo gestiono la 'request' y el 'response'.

Transformo la 'request' en un objeto 'options' y se le paso a un servicio.

Lo que devuelva el servicio lo meto en el 'response'.

Yo creo un fichero js para cada colección.

5.10 routes/talkRoute.js

```
module.exports = function(app) {
```

```

var TalkService = require('../services/talkService.js');

var findTalks = function(req, res) {
  TalkService.findAllTalks({...});
};
var findTalk = function(req, res) {
  var talkId = req.params.talkId;
  TalkService.findTalkById({...});
};
var addTalk = function(req, res) {...};
var updateTalk = function(req, res) {...};
var deleteTalk = function(req, res) {...};

// mapeamos método y URL a una función
app.get(    '/talk',          findTalks);
app.get(    '/talk/:talkId',  findTalk);
app.post(   '/talk',          addTalk);
app.put(    '/talk/:talkId',  updateTalk);
app.delete( '/talk/:talkId',  deleteTalk);
}

```

5.11 services

Aquí están los servicios que acceden a base de datos.

Aquí no hay ni request ni response.

Las funciones reciben un objeto 'options' con lo que necesita.

Normalmente deben gestionar al menos un 'onSuccess' y un 'onError'.

Hay funciones que además gestionan un 'onNotFound'.

Un servicio puede llamar a otros servicios.

Yo creo un fichero js para cada colección.

5.12 services/talkService.js

```

//importamos el modelo
var Talk = require('../models/talkModel.js');

var findAllTalks = function(options) {
  Talk.find(function(error, talks) {...});
};
var findTalkById = function(options) {

```

```
    Talk.findById(options.talkId, function(error, talk) {...});
};
var saveTalk = function(options) {...};
var findTalkByIdAndUpdate = function(options) {...};
var findTalkByIdAndRemove = function(options) {...});

// exportamos los servicios
exports.findAllTalks      = findAllTalks;
exports.findTalkById      = findTalkById;
exports.saveTalk          = saveTalk;
exports.findTalkByIdAndUpdate = findTalkByIdAndUpdate;
exports.findTalkByIdAndRemove = findTalkByIdAndRemove;
```


Chapter 6

Demo

Chapter 7

¿Alguna pregunta?