Servidor API REST con Node.js

Adolfo Sanz De Diego

Octubre 2013

# Acerca de

## 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.

## ¿Dónde encontrarnos?

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

Lista: [gul@gul.uc3m.es](mailto: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>

## 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**

## 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>

## 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>

## ¿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>

## 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>

## Licencia

Estas **transparencias** están bajo una licencia:

* [Creative Commons Reconocimiento-CompartirIgual 3.0](http://creativecommons.org/licenses/by-sa/3.0/es/)

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

* [GPL 3.0](http://www.viti.es/gnu/licenses/gpl.html)

## Fuentes

Transparencias:

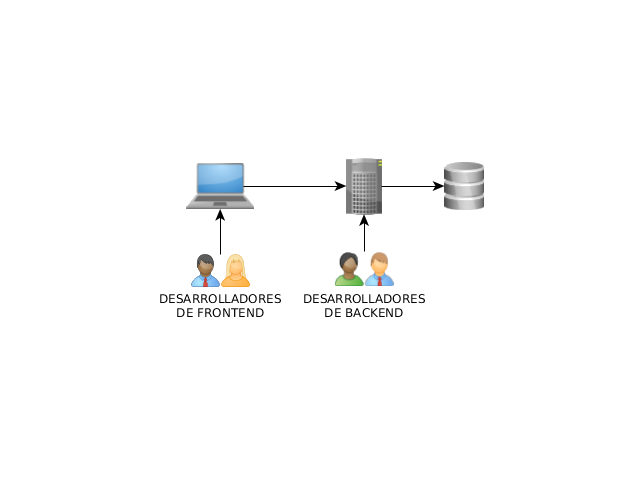
* [SlideShare](http://www.slideshare.net/asanzdiego/servidor-api-rest-con-nodejs)
* [Deck Slides](http://asanzdiego.github.io/curso-api-restful-nodejs-server-2013/slides/export/servidor-api-rest-con-nodejs-deck-slides.html)
* [Reveal Slides](http://asanzdiego.github.io/curso-api-restful-nodejs-server-2013/slides/export/servidor-api-rest-con-nodejs-reveal-slides.html)
* [Plain HTML](http://asanzdiego.github.io/curso-api-restful-nodejs-server-2013/slides/export/servidor-api-rest-con-nodejs.html)

Código:

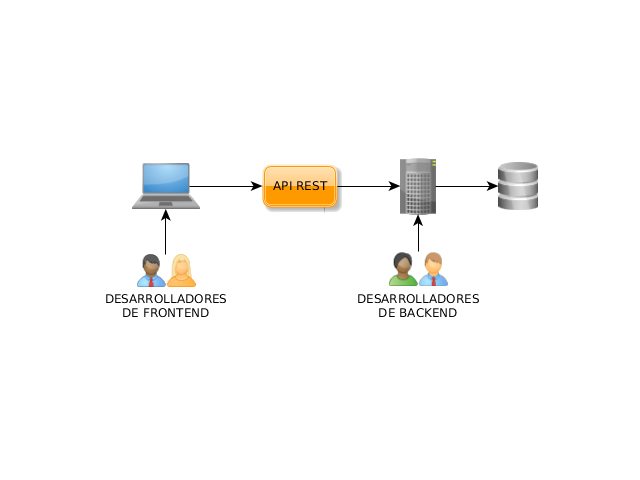
* <https://github.com/asanzdiego/curso-api-restful-nodejs-server-2013/tree/master/src>

# APIs ¿Para qué?

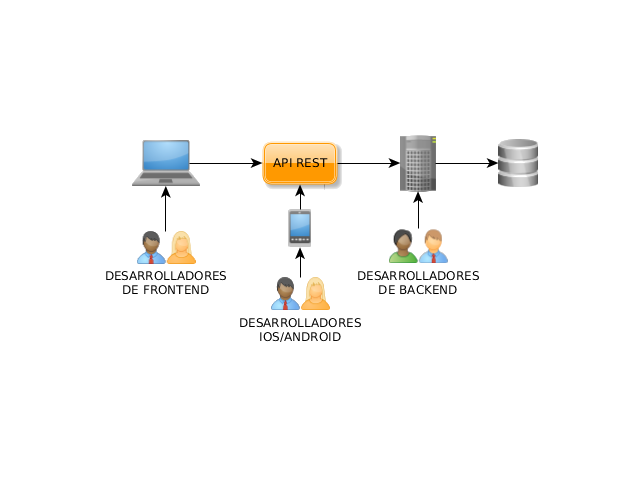
## Aplicación estándar



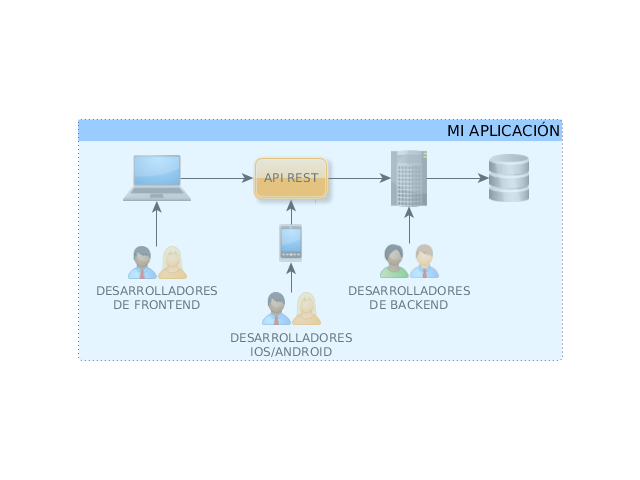
## Introducimos API



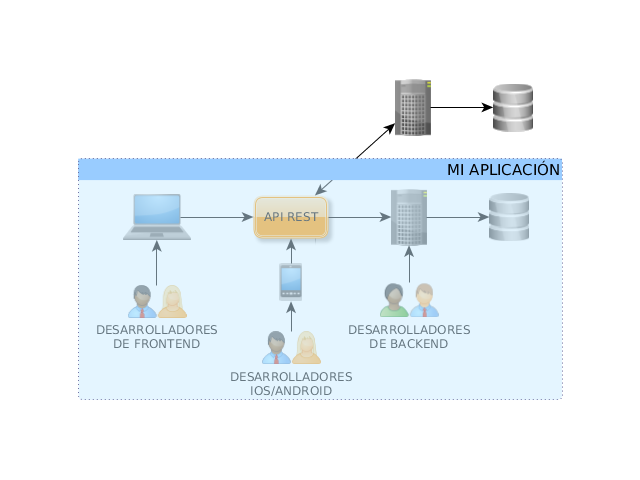
## Separación Roles



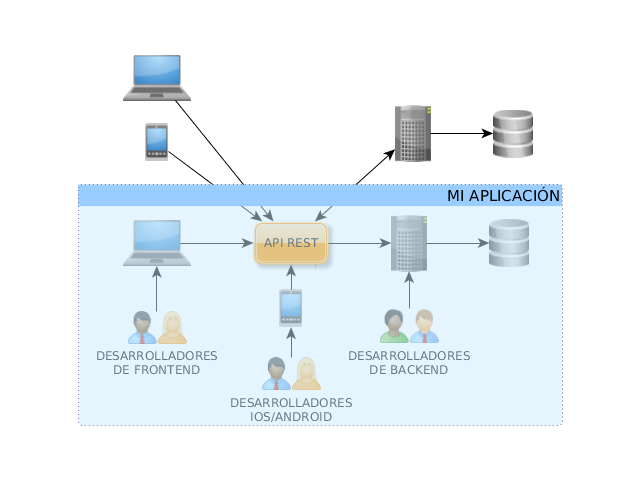
## ¿Y ahora qué?



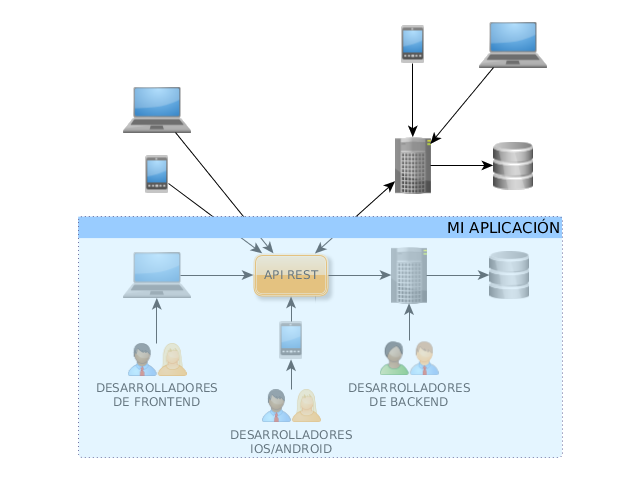
## Servicios externos



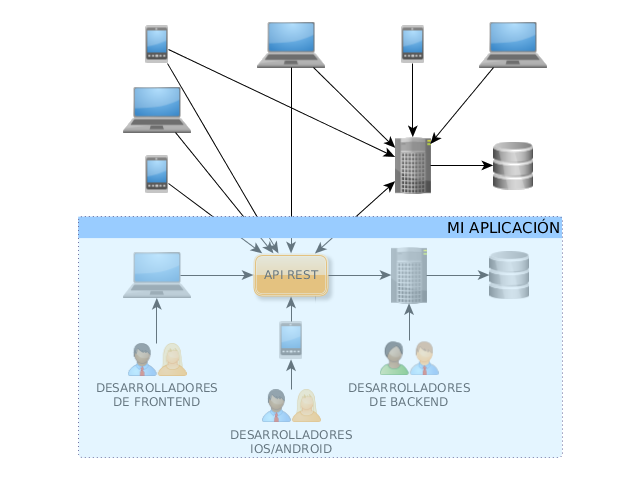
## Apps clientes



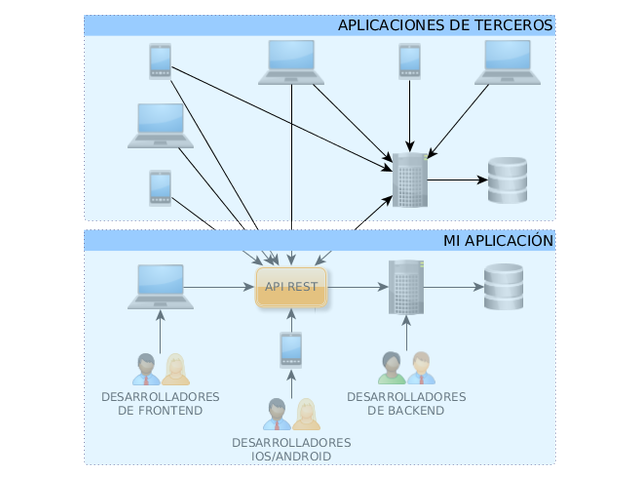
## Apps de servicios



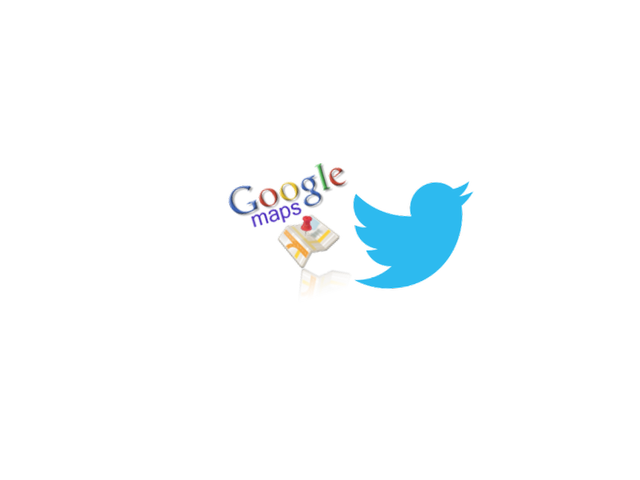
## Apps mixtas



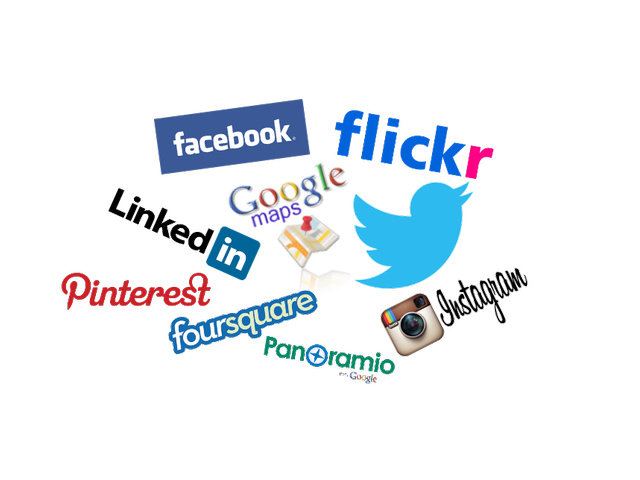
## Plataforma



## ¿Quien expone APIs?



## ¿Quien expone APIs?



## ¿Quien expone APIs?



## Exponlas tú



## Exponlas tú



## Exponlas tú



# APIs RESTful

## ¿Qué es REST?

REST (**Representational State Transfer**) es una técnica de arquitectura de software para sistemas hipermedia 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

## 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

## 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

## ¿Por qué REST?

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

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

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

Se complementa muy bien con **AJAX**.

## 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 REST**ful** indica que implementa dicha arquitectura.

* Se utiliza como **adjetivo**
* Se utiliza como por ejemplo: success**ful** = éxito**so**).

## 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 protócolo HTTP.
* Y se refiere a los servicios web REST (**sin el full**)
* Aquellos que NO implementan una API con  
   todos los métodos del protócolo HTTP.

# Node.js

## Introducción



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

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

## Ejecución

Ejecución **concurrente**

* Muchos tareas

Pero **NO paralelo**

* Una única hebra

## ¿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**.

## 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/>

## Primero pasos

Instalar **Node.js**

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

Instalar **MongoDB**

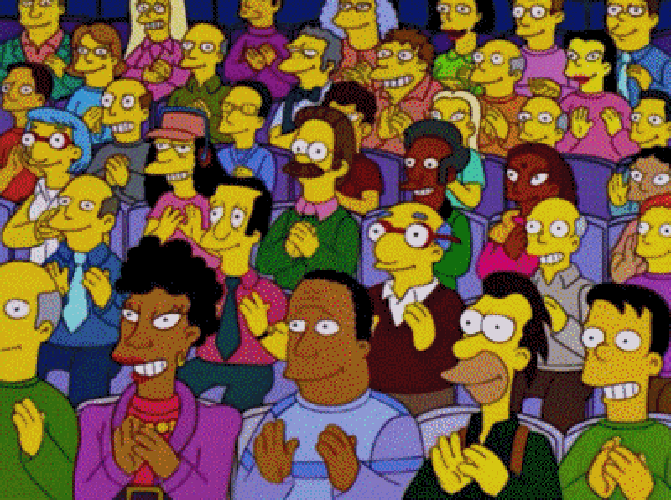
* <http://docs.mongodb.org/manual/installation/>

## Aburrido



# Código

## Aplausos



## 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"  
 }  
}

## npm install

Este comando instalará en la **carpeta node\_modules** las dependencias de nuestro proyecto.

npm install

## 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

## 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");  
});

## Directorios

Puedes usar la estructura de directorios que quieras.

Yo he usado esta:

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

## models

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

Yo creo un fichero js para cada colección.

## 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);

## 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.

## 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);  
}

## 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.

## 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;

# Demo

# ¿Alguna pregunta?